AERE - R 8269

C1

United Kingdom Atomic Energy Authority

HARWELL

Fortran subroutines for handling sparse linear programming bases

J. K. Reid
Computer Science and Systems Division
AERE Harwell, Oxfordshire
January 1976

PRICE £1.50 net from H.M. Stationery Office

C13

A.E.R.E.-R.8269

FORTRAN SUBROUTINES FOR HANDLING SPARSE LINEAR PROGRAMMING BASES

by

J.K. Reid

Abstract

In this report we present subroutines that implement a sparse variant (Reid,1975) of the Bartels-Golub algorithm for handling linear programming bases. There are separate subroutines for factorization, solution of linear systems using such a factorization and updating the factorization to correspond to the replacement of one of the columns of the basis.

Computer Science and Systems Division,
Building 8.9, A.E.R.E., Harwell,
Didcot, Oxfordshire.

January, 1976

HL76/111 (C.13)

ISBN-0-70-580176-4

C O N T E N T S

1. Introduction

The purpose of this report is to present three subroutines that implement a sparse variant (Reid,1975) of the Bartels-Golub algorithm. The subroutines are

a) LA05A, which produces a factorization

$$A = LU \tag{1.1}$$

of a given sparse nxn matrix A, where L is a matrix whose inverse is held as the product

$$L^{-1} = M_r M_{r-1} \ldots M_1 \tag{1.2}$$

of r matrices M_i that differ from I in just one element, and U is a permutation of an upper triangular matrix

$$\tilde{U} = PUQ \; ; \tag{1.3}$$

b) LA05B, which solves sets of equations

$$Ax = b \tag{1.4}$$

and

$$A^T x = b \tag{1.5}$$

using the factorization (1.1); and

c) LA05C which revises the factorization (1.1) when one of the columns of A is altered.

We expect the main application of these subroutines to be for solving linear programming problems (see for example Goldfarb and Reid,1976) but they are also available as a basic tool for use in other optimization problems.

The algorithm is described by Reid (1975) and some test results on medium scale problems are also given by Reid (1975). Here we concentrate on describing the code itself. Although the reader may find it helpful to read the earlier paper, it is our intention that this report should be self-contained. In order to make the code more understandable we describe the algorithm slightly differently and use slightly different notation.

The subroutines presented here are in standard Fortran, and have been checked by the Bell Telephone Laboratories' Fortran verifier (Ryder,1973), although those placed in the Harwell library are variants which use INTEGER*2 arrays. To aid readability of the code we avoid any backward jumps, all loops being programmed with the DO statement; this rule gives "well-structured" code without departing from standard Fortran. Occasionally we have needed a "DO WHILE" loop, for instance when looping along a linked list. In such cases we index the DO loop with an integer called IDUMMY (or JDUMMY, KDUMMY,...) to indicate that it is not used within the loop.

The code itself and its specification sheet are included in Section 6. So far as seems reasonable we have used comments in the code to explain its action. The underlying data structure is described in Section 2 and Sections 3,4 and 5 describe the methods used in LA05A, LA05B and LA05C. Besides the code and specification sheets we include in Section 6 some comments on runs under a preprocessor which made execution counts available.

The author would like to thank I.S. Duff for reading a draft of this report and making some helpful suggestions regarding the presentation.

2. Data structure

In order to explain our choice of data structure we need to sketch the algorithm used. Further details of the algorithm are given in later sections. Immediately after the original factorization (LA05A) L has a symmetric permutation

$$\tilde{L} = PLP^{-1} \tag{2.1}$$

which is a unit lower triangular matrix, but later (see next paragraph) it can only be said to be of the form (1.2). U is always a permuted upper triangular matrix (see (1.3)).

If column m of A is changed to give the matrix $\bar{A}$ then the factorization becomes

$$\bar{A} = LV \tag{2.2}$$

where V differs from U only in column m, and this is L^{-1} times column m of $\bar{A}$. V may already be a permuted upper triangular matrix, in which case we take $\bar{U}=V$, perhaps revising the permutation matrices P and Q. If it is not, then further elementary row operations $M_{r+1},\ldots,M_{\bar{r}}$ are used to reduce it to such a form. This gives the new factorization

$$\bar{A} = \bar{L}\,\bar{U} \tag{2.3}$$

of the same form as previously, and associated permutation matrices $\bar{P}$ and $\bar{Q}$. Notice that $\bar{L}^{-1}$ differs from L^{-1} only in the possible addition of more elementary row operation matrices $M_{\bar{r}},M_{\bar{r}-1},\ldots,M_{r+1}$ whereas $\bar{U}$ differs from U in having a column replaced and elementary row operations applied.

The factorization (1.1) enables the sets of equations (1.4) and (1.5) to be written as

$$x = U^{-1}(L^{-1}b) = U^{-1}(M_r\ldots M_1)b \tag{2.4}$$

and

$$x = L^{-T}(U^{-T}b) = M_1^T\ldots M_r^T(U^{-T}b) \tag{2.5}$$

It follows that at no stage is access required to the matrices M_i except in forward or reverse order and that updating need be nothing more complicated than adding additional matrices M_i. Therefore a sequential file which can be accessed forwards and backwards is a suitable storage mode and we have chosen to use the end of the arrays A and IND. The number of matrices M_i is held in LENL of COMMON/LA05D/ and they themselves are held as the operators "add A(K) times component IND(K,1) of a given vector to component IND(K,2)", for K = IA,IA-1,...IA-LENL+1.

A more complicated storage mode is needed for U because of the unpredictability of the extra non-zeros (fill-ins) that the row operations may produce. An early version (LA03) held each row of U as a linked list. This enables fill-ins to be handled easily but

i) with INTEGER*2 links on the IBM370 we were limited to about 32000 non-zeros,

ii) accessing a row is likely to involve accessing well-separated items in the store, undesirable on a paged machine, and

iii) the column structure is not available so that replacing column m requires a scan of the whole of U, for instance.

Because of these disadvantages we have decided to follow Gustavson (1972) in holding the rows of U as a file of packed vectors and the pattern of the columns of U as a file of lists of integers. The non-zeros of row I, say, are held in A(K),A(K+1),... and corresponding column numbers are held in IND(K,2),IND(K+1,2),..., with K held in IP(I,1) and the number of elements (non-zeros) held in IW(I,1). We use a completely separate file for the columns of U; here we do not store the numerical values but just the row numbers themselves. For column J these are in IND(K,1), IND(K+1,1),... where K is in IP(J,2) and the number of entries is in IW(J,2). When a row operation is applied to U we open a new entry for

the changed row and (temporarily) waste the space that the old row occupied. Any element in the row that changes to have modulus less than a given small number (SMALL of COMMON/LA05D/) is regarded as a zero and its entry in the column file is removed after having been interchanged with the last entry so that space at the end is released. Any fill-ins mean that an additional entry in the column structure is needed; this is placed at the end of the stored column if this is possible and if not a fresh entry is opened for the column and the old space is (temporarily) wasted. For both files there is a need for occasional "compression" to release wasted storage. This is done by subroutine LA05E, which is called to compress just one of the files whenever this is necessary. They are not treated together because usually the file of rows will need to be compressed more frequently since a new entry is made for every row operation applied to U. Spaces inside the file of rows are indicated by zero values of IND(K,2) and similarly zeros values of IND(K,1) indicate spaces in the column file. The overall length of the files, including internal spaces, are stored in LROW and LCOL of COMMON/LA05D/ and a count of the number of compresses (of either file) since last entry to LA05A is held in NCP of COMMON/LA05D/. We have found it convenient to hold the elements that are on the diagonal of the upper triangular matrix $\tilde{U} = PUQ$ as first in their rows and columns; otherwise the entries are in arbitrary order within the rows and columns.

The actual compression in LA05E is done by scanning every element of the old file, including dummies, and moving just the genuine elements forward. Details of how the pointers are adjusted are given in comments. It is perhaps unfortunate that we have to look at all the dummies but the computer time taken here will be small compared with the time taken in other subroutines when the element is genuine, and this method leads to a

simple subroutine that does not require a sort of the pointers to starts of entries.

Finally in this section we describe how the permutations P and Q are held. IW(I,3) holds the row number in U of row I of $\tilde{U}$ = PUQ and IW(J,4) holds the column number of column J of $\tilde{U}$.

3. Subroutine LA05A, which performs the original factorization

The original factorization is performed using Gaussian elimination with the pivotal strategy of Markowitz (1957), but subject to the stability requirement that no pivot be less than u (a user-set parameter for which we have found the value 0.1 satisfactory) times the largest element in its row. Markowitz' strategy is to use a non-zero with least product of number of other non-zeros in its row and of other non-zeros in its column. To make the search for such an element fast we hold doubly linked lists of all rows having the same number of non-zeros and of all columns having the same number of non-zeros. This enables us to search columns of length 1, then rows of length 1 then columns of length 2, etc., stopping whenever we know that there cannot be a non-zero later in the sequence with better Markowitz cost than the best so far found. Normally this search will terminate very early. Once the pivot has been chosen we remove from their linked lists all the rows which have a non-zero in the pivot column and all the columns that have a non-zero in the pivot row because these may have their numbers of non-zeros changed. Once the elimination is complete they are inserted in the list corresponding to their new numbers of non-zeros. We choose to search columns first to give a bias towards placing non-zeros in U rather than L, because L can only grow in length whereas at a change of basis a column of U is removed; also the column coming into U at a change of basis has been produced from the corresponding column of A by operating with L^{-1}, so is likely to have more non-zeros if L has more non-zeros.

The data structure used during elimination is similar to that used finally for U (see last section). As each elementary row operation is performed it is stored in its final position in the sequential file for L (using arrays A,IND from the end forwards). Initially A is stored in exactly the same way as U is eventually stored. During the elimination the row file holds all the rows but the column file contains only the submatrix of elements that have not been in a pivotal row or column (i.e. the "active" submatrix). Once the elimination is complete the column file for U is reconstructed from its row file.

For the user's convenience we require a different format for initial input of A. The non-zeros are passed in any order in the array A, with their row and column numbers in corresponding positions of arrays IND(·,1) and IND(·,2). The actual sorting of the non-zeros is done by subroutine MC20A, whose specification sheet is included in the appendix. It is fast since it handles each item that needs moving exactly three times. Although designed to order sparse matrices by columns we have been able to use it here by switching the roles of row and columns.

We do not permit any elements held in the row file to have the value zero because this leads to unnecessary work and use of storage. If any element has modulus less than SMALL (of COMMON/LA05D/ with default value zero) in the original matrix then it is removed before the row and column files are created. If any element with modulus less than SMALL is created during a row operation it is again removed. We recommend the user to reset SMALL to a positive value if he can since this will make underflows much less common and will save some storage.

The column file is used when choosing the pivot and in order to know which rows are active in the elimination itself. In neither case is there any need for the elements to be in order within their entry. We therefore make no attempt to order them, always adding any extra element to

the end of the entry and removing elements by overwriting the unwanted entry by the old last entry. It might be thought that the entries in the row file should be ordered so that during a row operation the two rows can be scanned in phase. We find it convenient to bring the elements of the pivot column to the front of both rows, but by using a full work vector of reals we can produce elegant and efficient code without the need for the remaining elements to be in any particular order. The full vector w is initialized to zero and is restored to zero after each use. For each row operation we

a) load the pivot row, excluding the pivot, into w.

b) scan the non-pivot row adding the required multiple of the appropriate component of w to each element and resetting the component of w to zero after its use.

c) scan the pivot row again to see which elements are still in w. Each that is gives rise to a fill-in.

4. Subroutine LA05B, which solve sets of equations

Our data structure was chosen with one of its aims that the solution of sets of equations $Ax=b$ and $A^Tx=b$ should be straightforward and fast. It very often happens in linear programming applications that b is very sparse indeed and even x may have few non-zeros, and we want to exploit this feature. This leads to slightly more complicated code for solving $Ax=b$ so we describe the solution of $A^Tx=b$ first.

Because the permutation $\tilde{U} = PUQ$ of U is an upper-triangular matrix, we may solve $U^Tw=b$ by a forward substitution process. The presence of the permutations makes it convenient to use a work vector, so we first load the input vector b into w and set b to zero, then solve $U^Tb=w$ by the forward substitution

$$b_{p_i} = (w_{q_i} - \sum_{j \neq p_i} u_{jq_i} b_j)/u_{p_i q_i} , \quad i=1,2,\ldots,n. \tag{4.1}$$

As each non-zero b_{p_i} is calculated we run through row p_i of U (column p_i of U^T) performing the operation

$$w_k \leftarrow w_k - u_{p_i k} b_{p_i} , \quad k \neq q_i , \tag{4.2}$$

which can of course be skipped for zero b_{p_i}. This ensures that at stage i of the process w_{q_i} has been updated fully and we find b_{p_i} by dividing by the pivot $u_{p_i q_i}$. Since b has been initialized to zero we do not even have to look up the array containing p_i (p_i is held in IW(I,3)) if w_{q_i} is zero (q_i is held in IW(I,4)). Notice also how convenient it is to have the pivot $u_{p_i q_i}$ stored at the row start. Once this process is complete we apply the sequence of elementary row operations that comprise L^{-T} to b so that it finally contains the required solution. Here we avoid null operations but for each operation we need an array look-up and to test for a zero vector component.

When solving Ax=b we begin by applying L^{-1} to b as a sequence of elementary row operations and again avoid null operations. Because of the permutations P and Q we next load b into w and set b to zero as in the last paragraph. Then we use the back-substitution

$$b_{q_i} = (w_{p_i} - \sum_{j \neq q_i} u_{p_i j} b_j)/u_{p_i q_i} , \quad i=n,n-1,\ldots,1. \tag{4.3}$$

Unfortunately we cannot exploit zero components of b directly because U is held by rows but we do have the sparsity patterns of the columns of U. Therefore after each non-zero component b_{q_i} is found we run through the pattern of column q_i of U marking those w_k, other than w_{p_i}, for which

$u_{kq_i} \neq 0$. At a later stage i we know that $\sum_{j \neq q_i} u_{p_i j} b_j$ is zero if w_{p_i} is unmarked and therefore do not need to calculate this sum. The marking is actually performed by negating the pointers IP(·,1). Notice that w is left unchanged as $L^{-1}b$. This is precisely the vector that is required by LA05C later if b is to replace an existing column of the basis. It is not inefficient to use this as the only means of specifying the incoming column because in linear programming it always happens that $A^{-1}b$ is wanted before it is known which column is to leave the basis matrix A.

The great sparsity of the vectors that arise in typical linear programming applications led us to consider a condensed storage mode for them and we even wrote such a variant of LA05B. However we eventually decided against using it because the code is significantly more complicated and would execute faster only when the number of non-zeros in the output vector is less than about $\sqrt{n}$ because we have either to keep the non-zeros in order or perform a search whenever one is wanted. The loops of length n that are avoided are simple and so likely to execute rapidly. We did find (see Section 6) that they were executed a great number of times, often giving the dominant execution count.

5. Subroutine LA05C, which updates the factorization

Subroutine LA05C updates the factorization following the replacement of column m (MM in the code) of A by b. We begin by removing column m of U and then inserting the vector $L^{-1}b$, calculated on a previous entry to LA05B, and stored in array W, as new column m to make the matrix V of equation (2.2). The permuted matrix

$$S = PVQ$$

is upper triangular except for one column (column M) which corresponds to the changed column and which we call the spike. The spike is read into our data structure in row order within S so that its last row number (in S) may be placed in the variable LAST. We refer to rows and columns M to LAST as "the bump". An example is shown in Figure 1 (page 15).

Our first aim is to alter the permutations in such a way that the length of the spike (size of the bump) is reduced. We begin by searching columns M+1,M+2,...,LAST looking for a column (which we call a singleton column) having only one non-zero in the bump. If column J is such a column and we apply the symmetric permutation in which rows and columns M,M+1,...,J-1 are all moved forward one place and row and column J becomes the new row and column M then the new matrix has exactly the same form as the old but now the spike is in column M+1 and is shorter by one component. In our example column 5 is such a column and the permuted matrix is shown in Figure 2. We can now look for a further singleton, apply another symmetric permutation and continue until none are found. It is more efficient, however, to find all the singletons then apply the composite symmetric permutation. This can be done by marking (by setting W(J)=1) column M and those of columns M+1,...,LAST which are not singletons. We begin by marking column M itself and then marking all columns which have a non-zero in row M, since these certainly cannot be singletons. We then look at the marks for columns M+1,M+2,... and for each marked column J we mark the columns having non-zeros in row J since again these cannot be singletons. In preparation for the next stage we also perform the symmetric permutation that places the spike at the end of the bump without altering the relative order of the other row and columns. The new order of columns is therefore i) singletons, in unchanged order among themselves,

ii) non-singletons, in unchanged order among themselves and

iii) spike column. Our example has only one column singleton and its form at this stage is shown in Figure 3. Because we hold pivots (diagonal elements of the upper triangular matrix $\tilde{U} = PUQ$) first in their row and column we do not need to store both P and Q explicitly and here we are performing a symmetric permutation so pivots remain pivots. We therefore revise only P and use the storage for Q as workspace.

These permutations leave us with a matrix $S = P_1VQ_1$ that is upper triangular apart from row LAST (which we now call the spike row) and this has non-zeros starting at column M1, say, with M1≥M. We refer to rows and columns M1 to LAST as "the bump", and apply an exactly similar process to rows LAST-1, LAST-2,...,M1 looking for rows that are singletons in the bump as previously when looking for column singletons, except that here we leave the spike where it is. This leaves us with a spike in row LAST1, say, with LAST1≤LAST, which commences in column M1. Our example (see Figure 3) begins with a row singleton in row 6 (row 7 of original matrix) and dealing with this makes row 4 into a singleton. Dealing with both of these gives the matrix shown in Figure 4.

We may be able to reduce the length of the spike still further if column LAST1 of this new matrix $S_2 = P_2VQ_2$ is a singleton in the bump rows M1 to LAST1. Unless we have the trivial case M1=LAST1 such a singleton must have its non-zero in row LAST1-1 since otherwise this would be a row singleton (and row singletons were all removed by the previous set of permutations). We therefore perform the unsymmetric permutation that makes row LAST1-1 become row M1, column LAST1 become column M1 and the intervening rows and columns (rows M1+1,...,LAST1-2 and columns M1+1,...,LAST1-1) all move forward one place. The new matrix

has the same structure but the spike now extends from column M1+1 of row LAST1 (see Figure 5). We repeat the process (see Figure 6), continuing until the trivial case is reached or a non-singleton is discovered. Again we may delay the permutations and perform them together. Rows I of U that correspond to rows M1 to LAST1 of S_2 are initially marked with W(I)=3. , so we can test the last column for being a singleton. If it is, then we revise JM to point to the new last column and reset W(I) to 2. for the row being moved forward ready for the next singleton test. Similar action is taken as later singletons are found. The permutation P, stored in IW(·,3) is revised when the process terminates, and M1 is increased to point to the new start of the spike row.

If these permutations have not reduced the matrix to upper triangular form (M1=LAST1), then the task is completed by application of a sequence of elementary Gaussian elimination steps. Each pivotal step begins with a search of the spike row for a non-zero in the pivot column. If one is found it is brought to the front of the spike. Next the pivot row is exchanged with the spike row if the stability test demands it (leading element of the pivot row less in modulus than u times that the other leading element) or if the sparseness suggests it (pivot row has greater number of non-zeros) while the stability test permits it (leading element of non-pivot row greater than u times leading element of pivot row). The actual row elimination is performed by code identical with that of LA05A.

Finally the permutation $\overline{Q}$ is constructed from $\overline{P}$, now stored in IW(·,3), by using the fact that pivotal elements are always first in the stored rows.

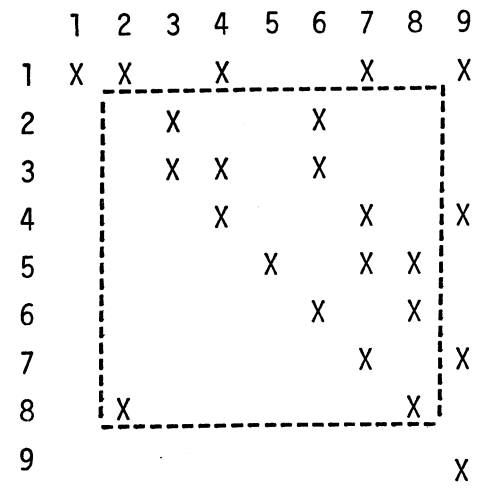


Figure 1. Original matrix

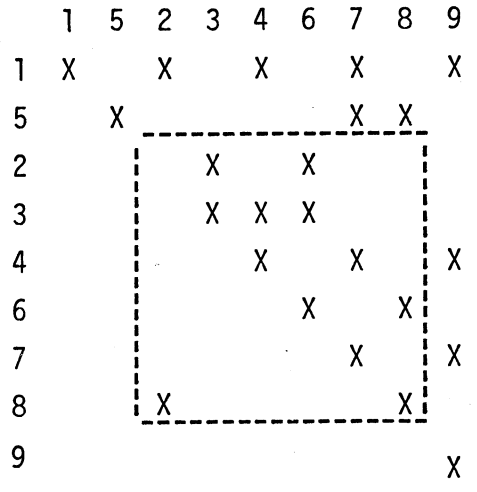


Figure 2. After column singleton moved to front of bump

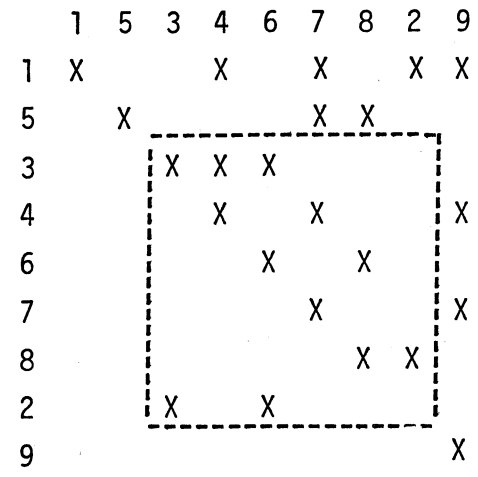


Figure 3. After spike moved to end of bump

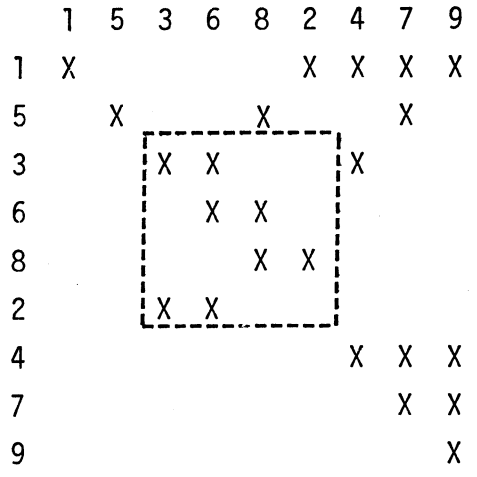
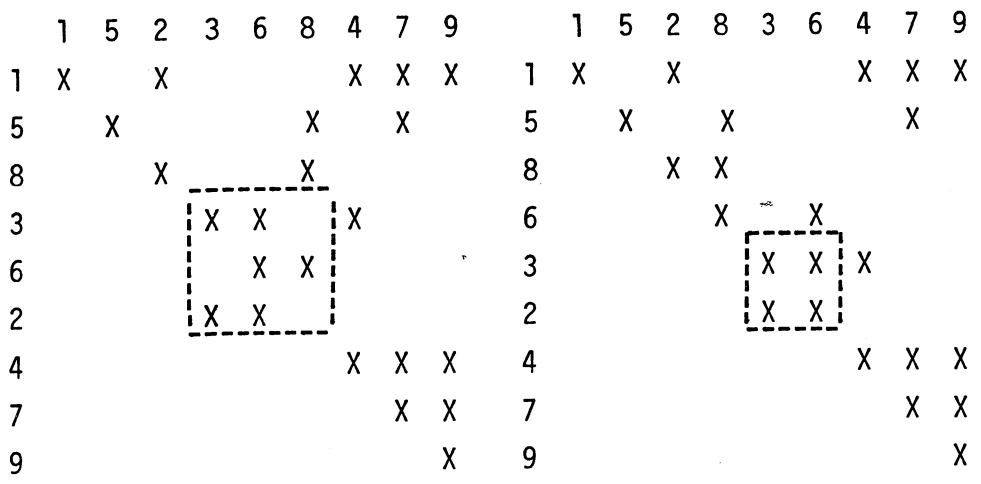


Figure 4. After treating row singletons



Figures 5 and 6. After treating first and second singleton spike column

6. Code and specification sheets

Given in this section is the code itself and its specification sheet. Also given is the specification sheet of the Harwell subroutine MC20A which is called by LA05A to sort the non-zeros by columns.

The code given is the single-length standard Fortran version. It contains comments which allow the Harwell subroutine OE04 to convert to double-length standard Fortran (version labelled D), double-length IBM Fortran (version I) or single-length IBM Fortran (version J). These comments consist of statements labelled with the version to which they apply. Each is an alternative to the Fortran statement that immediately follows it.

We have run some linear programming problems, including most of those by Goldfarb and Reid (1975), after treating the code by a preprocessor (Harwell subroutine OE02) that inserts additional statements for counting the number of executions of each statement. We used the steepest edge simplex algorithm, as described by Goldfarb and Reid (1975). The highest counts were invariably in LA05B. The simple

loops of length N (i.e. DO 113, DO 140, DO 303, DO 315) were always very prominent and particularly so in the sparser cases. That it is worthwhile to test for zero components in the vectors being operated upon, even in the relatively complicated case of applying U^{-1}, was amply demonstrated. In LA05C, rather surprising, the heaviest count was usually in the beginning of the loop DO 110, that is the simple search for non-zeros in the incoming column. Another high count in LA05C, but small compared with those of LA05B, was often in the simple loop DO 270 that sets the vector w to zero prior to its use for a sequence of eliminations. The success of the singleton selection process was indicated by the much higher counts obtained in the first scan of the bump (looking for singleton columns) than in later scans (looking for singleton rows and dealing with singleton spikes). All the LA05A counts were much lower than the high counts of LA05B and LA05C. If execution speed is very important it might be worthwhile to machine code LA05B, or parts of it, and the loop that finds the non-zeros in the incoming column for LA05C.

Harwell Subroutine Library

1. Purpose

This package of subroutines will factorize a matrix, solve corresponding systems of linear equations and update the factorization when a column of the matrix is altered, exploiting sparsity in all cases. Its primary application is likely to be for handling linear programming bases. It has three entries:

(a) LA05A factorizes a given matrix A.

(b) LA05B subsequently calculates $A^{-1}b$ or $A^{-T}b$ for a given vector b, using the factorized A.

(c) LA05C modifies the factorization to correspond with the replacement of a column of the matrix by the vector b of a previous LA05B entry which calculated $A^{-1}b$.

2. Argument Lists

```
CALL LA05A(A,IND,NZ,IA,N,IP,IW,W,G,U)
CALL LA05B(A,IND,   IA,N,IP,IW,W,G,B,TRANS)
CALL LA05C(A,IND,   IA,N,IP,IW,W,G,U,M)
```

A is a REAL(DOUBLE PRECISION for the D version) array of length IA. On entry to LA05A it must be set to contain, in any order, the non-zeros of A. On exit from LA05A and on entry to and exit from LA05B/C it contains the factors of the current matrix A. It is altered by LA05A and C, and must not be altered by the user except prior to an LA05A entry.

IND is an INTEGER(INTEGER*2 for IBM versions) array of dimensions (IA,2). On entry to LA05A IND(K,1),IND(K,2) must be set to contain the row and column number of the non-zero held in A(K) for K=1,2,...NZ. It is altered by LA05A and C, and must not be altered by the user except prior to an LA05A entry.

NZ (INTEGER) must be set by the user to the number of non-zeros in A. It is used by LA05A only and is not altered by it.

IA (INTEGER) must be set by the user to indicate the size of arrays A and IND. Advice on the choice of the size is given in §3. It is not altered by LA05A/B/C.

N (INTEGER) must be set by the user to the order of A. For the IBM versions it may not exceed 32767 because of the use of INTEGER*2 arrays. It is not altered by LA05A/B/C.

IP is an INTEGER work array of length N*2. It must not be altered by the user except prior to an LA05A entry.

IW is an INTEGER(INTEGER*2 for IBM versions) work array of length N*2 the first half of which must not be altered by the user except prior to an LA05A entry. The second half is not used by LA05B/C.

W is a REAL (or DOUBLE PRECISION in the D version) working array of length at least N. It is used to transmit information about an incoming column between an LA05B entry with TRANS=.FALSE.and a subsequent LA05C entry, and therefore should not be altered between two such entries.

G (REAL or DOUBLE PRECISION in the D version) is used to output information about the stability of the factorization and error conditions. After a successful entry G is positive and equal to the modulus of the largest element in any of the reduced matrices. This is explained further in section 6. After an unsuccessful entry it is set negative (see §4 for details).

U (REAL or DOUBLE PRECISION in the D version) is a number set by the user in the range $0<U\leq 1$ to control the choice of pivots; if $U>1$ it is reset to 1 and if $U\leq 0$ it is reset to the relative floating-point accuracy. When searching for a pivot any element less than U times the largest element in its row is excluded. Thus decreasing U biases the algorithm towards maintaining sparsity at the expense of stability and vice-versa. The value 0.1 has been found satisfactory in test examples. It is used only by LA05A and LA05C.

B is a REAL (or DOUBLE PRECISION in the D version) array of length N used by LA05B to input b and output $A^{-1}b$ (TRANS=.FALSE.) or $A^{-T}b$ (TRANS=.TRUE.). It is used by LA05B only.

TRANS is a LOGICAL variable which must be set to .FALSE. if $A^{-1}b$ is required from LA05B and to .TRUE. if $A^{-T}b$ is wanted. It is used only by LA05B and is not altered by it.

M (INTEGER) is the column number in A of the column to be replaced in an LA05C entry. It is used by LA05C only and is not altered by it.

3. Storage considerations

The matrix is factorized into a product LU. The matrix L is stored as a product of matrices L_i differing from the unit matrix I on only one element. Each matrix L_i is stored in one position of A and IND and LENL in COMMON (see §5) holds the number of such matrices. U is a permutation of an upper triangular matrix and its number of non-zeros is held in LENU of COMMON (see §5). It is held in A and IND as a file containing separate ordered lists for each row and column. This file will need occasional compression to release space used by altered rows and columns. This compression (actually performed by the subroutine LA05E) will not add a significant overhead to the computational cost if it happens less often than, say, alternate calls of LA05C. If it is required more than twenty times in a single call of LA05C then this call is aborted and a diagnostic is printed. If such a call follows a long sequence of LA05C calls then it can probably be corrected by a fresh

LA05A call. Both L and U are held in A and IND so IA (the size of A and IND) must exceed LENL+LENU (the number of non-zeros in L and U) by a margin sufficient to avoid overfrequent compression of the file holding U. The adequacy of the length IA may be judged by monitoring NCP (in COMMON, see §5) which accumulates the number of times the U file is compressed since the last entry to LA05A.

4. Error diagnostics

After an unsuccessful entry a message is output on the line printer (unless suppressed or switched to another stream, see §5) and G is set negative to indicate one of the following conditions.

-1 'N is not positive'.

-2 'Row (or col) j has no elements'.

-3 'Element k is in row i and column j' (one of which is out of range).

-4 'There is more than one entry in row i and column j'.

-5 'The matrix was found singular in pivotal step k. Row (or col) j is dependent on rows (or cols) k,ℓ,...'.

-6 'Singular matrix created by replacement of col m'.

-7 'IA is too small'.

Diagnostics 1 to 5 may result from an entry into LA05A and 6 or 7 may result from an LA05C entry. Also error returns (with G unchanged) may result from LA05B or LA05C if G is negative, indicating a previous error return.

5. Use of Common

The subroutines contain the following common block

```
COMMON/LA05D/SMALL,LP,LENL,LENU,NCP,LROW,LCOL
```

(called LA05DD in the D version).

SMALL is a REAL variable (DOUBLE PRECISION in the D version) given the default value zero by BLOCK DATA. Its purpose is explained in §6.

LP is an INTEGER variable, given the default value 6 by BLOCK DATA, and used for stream number for diagnostic messages. Messages are suppressed if LP=0.

LENL,LENU,NCP are INTEGER variables giving information about use of the store (see §3).

LROW and LCOL are INTEGER variables used internally by the LA05 subroutines to hold the lengths of the files holding U by rows and its structure by columns.

If the user includes a common statement of the above form in his program then he may alter SMALL and LP from their default values and he may inspect LENL, LENU and NCP.

None of the variables LENL,...,LCOL may be altered by the user except prior to an LA05A entry.

6. Method and general notes

LA05A decomposes A into triangular factors using sparse matrix techniques similar to those of MA18A, documented in AERE Report R.6844.

Changing a column of A corresponds to changing a column of the upper triangular factor so that it is no longer a permutation of a triangular matrix and further row operations and/or permutations are needed to restore it to this form.

To control stability all pivots are chosen so that the multiples of a row that are added to another are always less than 1/U and stability is monitored by the parameter G, which is set to the modulus of the largest element in A or any of the upper triangular matrices to which it is reduced. If ε is the relative accuracy of the computation in use then the solutions obtained will have errors comparable with those of a perturbed system with matrix $A+\delta A$, elements of δA being less than a small multiple of εG. Any elements of the upper triangular factor that are less than SMALL (of COMMON, see §4) are reset to zero; this has an effect comparable with that of making a perturbation to A whose elements have size about SMALL. We recommend the user to reset this to a positive value if he can, because this will save most underflow interrupts and some storage.

An LA05A call is normally followed by a long sequence of calls of LA05B and C. The time taken by LA05B will grow steadily as the number of non-zeros in the factors of A grows and eventually it will be more economic to call LA05A with the current matrix A and continue from this. A further call of LA05A may also be needed because of instability; large values of G are an indication of trouble but a better test is to calculate $r=Ax-b$ (or A^Tx-b) where x is the result of a LA05B call and compare r_i with $\sum_j |a_{ij}x_j|$ (or $\sum_j |a_{ji}x_j|$).

7. Other subroutines

This is in fact a package of subroutines whose names are LA05A, LA05B, LA05C and LA05E.

December, 1975

1. Purpose

a) MC20A: To sort the non-zeros of a sparse matrix from arbitrary order to column order, unordered within each column.

b) MC20B: To sort the non-zeros within each column of a sparse matrix stored by columns.

2. Argument lists

CALL MC20A(NC,MAXA,A,INUM,JPTR,JNUM,JDISP)

CALL MC20B(NC,MAXA,A,INUM,JPTR)

NC (INTEGER) must be set by the user to the number of matrix columns, and for the IBM versions it must not exceed 32767+JDISP. It is not altered by MC20A/B.

MAXA (INTEGER) must be set by the user to the number of matrix non-zeros. It is not altered by MC20A/B.

A is a REAL (DOUBLE PRECISION in D version) array of length MAXA. For entry to MC20A the user must set it to contain the non-zeros in any order. On exit from MC20A they are reordered so that column 1 precedes column 2 which precedes column 3, etc, but the order within columns is arbitrary. This format is required for entry to MC20B. On exit from MC20B the non-zeros are also ordered within each column.

INUM is an INTEGER(INTEGER*2 for IBM versions) array of length MAXA. On entry to and exit from MC20A/B the absolute value of INUM(K) is the row number of the element in A(K). The values, including signs, are moved so the user is at liberty to use these signs as flags attached to the non-zeros.

JPTR is an INTEGER array of length NC. It is not required to be set for entry to MC20A. On exit from MC20A and on entry to and exit from MC20B it contains the position in A of the first element of column J,J=1,2,...NC.

JNUM is an INTEGER(INTEGER*2 for IBM versions) array of length MAXA. On entry to MC20A JNUM(K)+JDISP is the column number of the element held in A(K). It is destroyed by MC20A.

JDISP (INTEGER) must be set by the user to his required displacement for column numbers, in the range [0,32767]. Normally zero will be suitable, but positive values permit matrices with up to 65534 columns to be handled by the IBM versions for which JNUM is an INTEGER*2 array. JDISP is not altered by MC20A.

3. Notes

It is expected that this subroutine will be called by other library subroutines but not by the user directly. There are no checks on the validity of the data and no error exits.

4. Method

MC20A is an in-place sort algorithm which handles each item to be sorted exactly 3 times, so it is of order MAXA. The number of elements in each column is first obtained by a counting pass. The space needed by each column is allocated. Each element in turn is made the "current element" and examined to see if it is in place. If not, it is put into the next location allotted for the column it occurs in, and the element displaced made the current element. This chain of displacing elements continues until the first element examined in the chain is located and stored. Then the next item is examined. A flag prevents an element being moved twice.

MC20B is a pairwise interchange algorithm of maximum order r(r-1)/2, for each column, where r is the number of elements in the column.

November, 1975

```
C I AND J ARE IBM FORTRAN DOUBLE AND SINGLE LENGTH VERSIONS       SJID/
C D AND S ARE STANDARD FORTRAN DOUBLE AND SINGLE LENGTH VERSIONS
C     SUBROUTINE LA05AD(A,IND,NZ,IA,N,IP,IW,W,G,U)                    DI/
      SUBROUTINE LA05A (A,IND,NZ,IA,N,IP,IW,W,G,U)
      INTEGER IP(N,2),RC(2,3)
C     INTEGER*2 IND(IA,2),IW(N,8)                                     IJ/
      INTEGER   IND(IA,2),IW(N,8)
C     DOUBLE PRECISION A(IA),AMAX,AU,AM,G,U,SMALL,W(N)                DI/
                REAL A(IA),AMAX,AU,AM,G,U,SMALL,W(N)
C     COMMON/LA05DD/SMALL,LP,LENL,LENU,NCP,LROW,LCOL                  DI/
C IP(I,1),IP(I,2) POINT TO THE START OF ROW/COL I.
C IW(I,1),IW(I,2) HOLD THE NUMBER OF NON-ZEROS IN ROW/COL I.
C DURING THE MAIN BODY OF THIS SUBROUTINE THE VECTORS IW(,3),IW(,5),
C     IW(,7) ARE USED TO HOLD DOUBLY LINKED LISTS OF ROWS THAT HAVE
C     NOT BEEN PIVOTAL AND HAVE EQUAL NUMBERS OF NON-ZEROS.
C IW(,4),IW(,6),IW(,8) HOLD SIMILAR LISTS FOR THE COLUMNS.
C IW(I,3),IW(I,4) HOLD FIRST ROW/COLUMN TO HAVE I NON-ZEROS
C     OR ZERO IF THERE ARE NONE.
C IW(I,5), IW(I,6) HOLD ROW/COL NUMBER OF ROW/COL PRIOR TO ROW/COL I
C     IN ITS LIST, OR ZERO IF NONE.
C IW(I,7), IW(I,8) HOLD ROW/COL NUMBER OF ROW/COL AFTER ROW/COL I
C     IN ITS LIST, OR ZERO IF NONE.
C FOR ROWS/COLS THAT HAVE BEEN PIVOTAL IW(I,5),IW(I,6) HOLD NEGATION OF
C     POSITION OF ROW/COL I IN THE PIVOTAL ORDERING.
      COMMON/LA05D /SMALL,LP,LENL,LENU,NCP,LROW,LCOL
      DATA RC(1,1),RC(1,2),RC(1,3),RC(2,1),RC(2,2),RC(2,3)
     1 / 1HR,1HO,1HW,1HC,1HO,1HL/
C     DATA EPS/2.3E-16/                                               DI/
C EPS IS THE RELATIVE ACCURACY OF FLOATING-POINT COMPUTATION
      DATA EPS/1.0E-6 /
      IF(U.GT.1.)U=1.
      IF(U.LT.EPS)U=EPS
      IF(N.LT.1)GO TO 520
      G=0.
      DO 5 I=1,N
      W(I)=0.
      DO 5 J=1,5
5     IW(I,J)=0
C
C FLUSH OUT SMALL ENTRIES, COUNT ELEMENTS IN ROWS AND COLUMNS
      L=1
```

```
      LENU=NZ
      DO 20 IDUMMY=1,NZ
      IF(L.GT.LENU)GO TO 25
      DO 10 K=L,LENU
C     IF(DABS(A(K)).LE.SMALL)GO TO 15                                    DI/
      IF( ABS(A(K)).LE.SMALL)GO TO 15
      I=IND(K,1)
      J=IND(K,2)
C     G=DMAX1(DABS(A(K)),G)                                              DI/
      G=AMAX1( ABS(A(K)),G)
      IF(I.LT.1 .OR. I.GT.N)GO TO 540
      IF(J.LT.1 .OR. J.GT.N)GO TO 540
      IW(I,1)=IW(I,1)+1
10    IW(J,2)=IW(J,2)+1
      GO TO 25
15    L=K
      A(L)=A(LENU)
      IND(L,1)=IND(LENU,1)
      IND(L,2)=IND(LENU,2)
20    LENU=LENU-1
C
25    LENL=0
      LROW=LENU
      LCOL=LROW
C MCP IS THE MAXIMUM NUMBER OF COMPRESSES PERMITTED BEFORE AN
C     ERROR RETURN RESULTS.
      MCP=MAXO(N/10,20)
      NCP=0
C CHECK FOR NULL ROW OR COLUMN AND INITIALIZE IP(I,2) TO POINT
C     JUST BEYOND WHERE THE LAST COMPONENT OF COLUMN I OF A WILL
C     BE STORED.
      K=1
      DO 28 IR=1,N
      K=K+IW(IR,2)
      IP(IR,2)=K
      DO 28 L=1,2
      IF(IW(IR,L).LE.0)GO TO 580
 28   CONTINUE
C REORDER BY ROWS
C     CALL MC20AD(N,LENU,A,IND(1,2),IP,IND(1,1),0)                       DI/
C CHECK FOR DOUBLE ENTRIES WHILE USING THE NEWLY CONSTRUCTED
```

```
C      ROW FILE TO CONSTRUCT THE COLUMN FILE. NOTE THAT BY PUTTING
C     THE ENTRIES IN BACKWARDS AND DECREASING IP(J,2) EACH TIME IT
C      IS USED WE AUTOMATICALLY LEAVE IT POINTING TO THE FIRST ELEMENT.
       CALL MC20A (N,LENU,A,IND(1,2),IP,IND(1,1),0)
       KL=LENU
       DO 40 II=1,N
       IR=N+1-II
       KP=IP(IR,1)
       DO 30 K=KP,KL
       J=IND(K,2)
       IF(IW(J,5).EQ.IR)GO TO 500
       IW(J,5)=IR
       KR=IP(J,2)-1
       IP(J,2)=KR
30     IND(KR,1)=IR
40     KL=KP-1
C
C SET UP LINKED LISTS OF ROWS AND COLS WITH EQUAL NUMBERS OF NON-ZEROS.
       DO 100 L=1,2
       DO 100 I=1,N
       NZ=IW(I,L)
       IN=IW(NZ,L+2)
       IW(NZ,L+2)=I
       IW(I,L+6)=IN
       IW(I,L+4)=0
100    IF(IN.NE.0)IW(IN,L+4)=I
C
C
C START OF MAIN ELIMINATION LOOP.
       DO 480 IPV=1,N
C FIND PIVOT. JCOST IS MARKOWITZ COST OF CHEAPEST PIVOT FOUND SO FAR.
C      WHICH IS IN ROW IPP AND COLUMN JP.
       JCOST=N*N
C LOOP ON LENGTH OF COLUMN TO BE SEARCHED
       DO 155 NZ=1,N
       IF(JCOST.LE.(NZ-1)**2)GO TO 183
       J=IW(NZ,4)
C SEARCH COLUMNS WITH NZ NON-ZEROS.
       DO 131 IDUMMY=1,N
       IF(J.LE.0)GO TO 133
       KP=IP(J,2)
```

```
      KL=KP+IW(J,2)-1
      DO 130 K=KP,KL
      I=IND(K,1)
      KCOST=(NZ-1)*(IW(I,1)-1)
      IF(KCOST.GE.JCOST)GO TO 130
      IF(NZ.EQ.1)GO TO 125
C FIND LARGEST ELEMENT IN ROW OF POTENTIAL PIVOT.
      AMAX=0.
      K1=IP(I,1)
      K2=IW(I,1)+K1-1
      DO 120 KK=K1,K2
C     AMAX=DMAX1(AMAX,DABS(A(KK)))                                    DI/
      AMAX=AMAX1(AMAX, ABS(A(KK)))
120   IF(IND(KK,2).EQ.J)KJ=KK
C PERFORM STABILITY TEST.
C      IF(DABS(A(KJ)).LT.AMAX*U)GO TO 130                             DI/
       IF( ABS(A(KJ)).LT.AMAX*U)GO TO 130
125   JCOST=KCOST
      IPP=I
      JP=J
      IF(JCOST.LE.(NZ-1)**2)GO TO 183
130   CONTINUE
131   J=IW(J,8)
C SEARCH ROWS WITH NZ NON-ZEROS.
133   I=IW(NZ,3)
      DO 151 IDUMMY=1,N
      IF(I.LE.0)GO TO 155
      AMAX=0.
      KP=IP(I,1)
      KL=KP+IW(I,1)-1
C FIND LARGEST ELEMENT IN THE ROW
      DO 140 K=KP,KL
C140  AMAX=DMAX1(DABS(A(K)),AMAX)                                     DI/
 140  AMAX=AMAX1( ABS(A(K)),AMAX)
      AU=AMAX*U
      DO 150 K=KP,KL
C PERFORM STABILITY TEST.
C     IF(DABS(A(K)).LT.AU)GO TO 150                                   DI/
      IF( ABS(A(K)).LT.AU)GO TO 150
      J=IND(K,2)
      KCOST=(NZ-1)*(IW(J,2)-1)
```

```
      IF(KCOST.GE.JCOST)GO TO 150
      JCOST=KCOST
      IPP=I
      JP=J
      IF(JCOST.LE.(NZ-1)**2)GO TO 183
150   CONTINUE
151   I=IW(I,7)
155   CONTINUE
C
C PIVOT FOUND.
C REMOVE ROWS AND COLUMNS INVOLVED IN ELIMINATION FROM ORDERING VECTORS.
183   KP=IP(JP,2)
      KL=IW(JP,2)+KP-1
      DO 195 L=1,2
      DO 190 K=KP,KL
      I=IND(K,L)
      IL=IW(I,L+4)
      IN=IW(I,L+6)
      IF(IL.EQ.0)GO TO 185
      IW(IL,L+6)=IN
      GO TO 190
185   NZ=IW(I,L)
      IW(NZ,L+2)=IN
190   IF(IN.GT.0)IW(IN,L+4)=IL
      KP=IP(IPP,1)
195   KL=KP+IW(IPP,1)-1
C STORE PIVOT
      IW(IPP,5)=-IPV
      IW(JP,6 )=-IPV
C ELIMINATE PIVOTAL ROW FROM COLUMN FILE AND FIND PIVOT IN ROW FILE.
      DO 219 K=KP,KL
      J=IND(K,2)
      KPC=IP(J,2)
      IW(J,2)=IW(J,2)-1
      KLC=KPC+IW(J,2)
      DO 215 KC=KPC,KLC
      IF(IPP.EQ.IND(KC,1))GO TO 216
215   CONTINUE
216   IND(KC,1)=IND(KLC,1)
      IND(KLC,1)=0
219   IF(J.EQ.JP)KR=K
```

```
C BRING PIVOT TO FRONT OF PIVOTAL ROW.
      AU=A(KR)
      A(KR)=A(KP)
      A(KP)=AU
      IND(KR,2)=IND(KP,2)
      IND(KP,2)=JP
C
C PERFORM ELIMINATION ITSELF, LOOPING ON NON-ZEROS IN PIVOT COLUMN.
      NZC=IW(JP,2)
      IF(NZC.EQ.0)GO TO 468
      DO 467 NC=1,NZC
      KC=IP(JP,2)+NC-1
      IR=IND(KC,1)
C SEARCH NON-PIVOT ROW FOR ELEMENT TO BE ELIMINATED.
      KR=IP(IR,1)
      KRL=KR+IW(IR,1)-1
      DO 290 KNP=KR,KRL
      IF(JP.EQ.IND(KNP,2))GO TO 300
290   CONTINUE
C BRING ELEMENT TO BE ELIMINATED TO FRONT OF ITS ROW.
300   AM=A(KNP)
      A(KNP)=A(KR)
      A(KR)=AM
      IND(KNP,2)=IND(KR,2)
      IND(KR,2)=JP
      AM=-A(KR)/A(KP)
C COMPRESS ROW FILE UNLESS IT IS CERTAIN THAT THERE IS ROOM FOR NEW ROW.
      IF(LROW+IW(IR,1)+IW(IPP,1)+LENL.LE.IA)GO TO 340
      IF(NCP.GE.MCP .OR. LENU+IW(IR,1)+IW(IPP,1)+LENL.GT.IA)GO TO 600
C     CALL LA05ED(A,IND(1,2),IP,N,IW,IA  ,.TRUE.)                    DI/
      CALL LA05E (A,IND(1,2),IP,N,IW,IA  ,.TRUE.)
      KP=IP(IPP,1)
      KR=IP(IR,1)
340   KRL=KR+IW(IR,1)-1
      KQ=KP+1
      KPL=KP+IW(IPP,1)-1
C PLACE PIVOT ROW (EXCLUDING PIVOT ITSELF) IN W.
      IF(KQ.GT.KPL)GO TO 350
      DO 345 K=KQ,KPL
      J=IND(K,2)
345   W(J)=A(K)
```

```
350    IP(IR,1)=LROW+1
C
C TRANSFER MODIFIED ELEMENTS.
       IND(KR,2)=0
       KR=KR+1
       IF(KR.GT.KRL)GO TO 380
       DO 370 KS=KR,KRL
       J =IND(KS,2)
       AU=A(KS)+AM*W(J)
       IND(KS,2)=0
C IF ELEMENT IS VERY SMALL REMOVE IT FROM U.
C      IF(DABS(AU).LE.SMALL)GO TO 365                                    DI/
       IF( ABS(AU).LE.SMALL)GO TO 365
C      G=DMAX1(G,DABS(AU))                                               DI/
       G=AMAX1(G, ABS(AU))
       LROW=LROW+1
       A(LROW)=AU
       IND(LROW,2)=J
       GO TO 370
365    LENU=LENU-1
C REMOVE ELEMENT FROM COL FILE.
       K=IP(J,2)
       KL=K+IW(J,2)-1
       IW(J,2)=KL-K
       DO 366 KK=K,KL
       IF(IND(KK,1).EQ.IR)GO TO 367
366 .  CONTINUE
367    IND(KK,1)=IND(KL,1)
       IND(KL,1)=0
370    W(J)=0.
C
C SCAN PIVOT ROW FOR FILLS.
380    IF(KQ.GT.KPL)GO TO 435
       DO 430 KS=KQ,KPL
       J=IND(KS,2)
       AU=AM*W(J)
C      IF(DABS(AU).LE.SMALL)GO TO 430                                    DI/
       IF( ABS(AU).LE.SMALL)GO TO 430
       LROW=LROW+1
       A(LROW)=AU
       IND(LROW,2)=J
```

```
      LENU=LENU+1
C
C CREATE FILL IN COLUMN FILE.
      NZ=IW(J,2)
      K=IP(J,2)
      KL=K+NZ-1
C IF POSSIBLE PLACE NEW ELEMENT AT END OF PRESENT ENTRY.
      IF(KL.NE.LCOL)GO TO 390
      IF(LCOL+LENL.GE.IA)GO TO 400
      LCOL=LCOL+1
      GO TO 395
390   IF(IND(KL+1,1).NE.0)GO TO 400
395   IND(KL+1,1)=IR
      GO TO 425
C NEW ENTRY HAS TO BE CREATED.
400   IF(LCOL+LENL+NZ+1.LT.IA)GO TO 410
C COMPRESS COLUMN FILE IF THERE IS NOT ROOM FOR NEW ENTRY.
      IF(NCP.GE.MCP .OR. LENU+LENL+NZ+1.GE.IA)GO TO 600
C     CALL LA05ED(A,IND,IP(1,2),N,IW(1,2),IA  ..FALSE.)                DI/
      CALL LA05E (A,IND,IP(1,2),N,IW(1,2),IA  ..FALSE.)
      K=IP(J,2)
      KL=K+NZ-1
C TRANSFER OLD ENTRY INTO NEW.
410   IP(J,2)=LCOL+1
      DO 420 KK=K,KL
      LCOL=LCOL+1
      IND(LCOL,1)=IND(KK,1)
420   IND(KK,1)=0
C ADD NEW ELEMENT.
      LCOL=LCOL+1
      IND(LCOL,1)=IR
C425  G=DMAX1(G,DABS(AU))                                              DI/
 425  G=AMAX1(G, ABS(AU))
      IW(J,2)=NZ+1
430   W(J)=0.
435   IW(IR,1)=LROW+1-IP(IR,1)
C
C STORE MULTIPLIER
      IF(LENL+LCOL+1.LE.IA)GO TO 450
C COMPRESS COL FILE IF NECESSARY.
      IF(NCP.GE.MCP)GO TO 600
```

```
C     CALL LA05ED(A,IND,IP(1,2),N,IW(1,2),IA   ,.FALSE.)                DI/
      CALL LA05E (A,IND,IP(1,2),N,IW(1,2),IA   ,.FALSE.)
450   K=IA-LENL
      LENL=LENL+1
      A(K)=AM
      IND(K,1)=IPP
      IND(K,2)=IR
      LENU=LENU-1
467   CONTINUE
C
C INSERT ROWS AND COLUMNS INVOLVED IN ELIMINATION IN LINKED LISTS
C     OF EQUAL NUMBERS OF NON-ZEROS.
468   K1=IP(JP,2)
      K2=IW(JP,2)+K1-1
      IW(JP,2)=0
      DO 480 L=1,2
      IF(K2.LT.K1)GO TO 475
      DO 470 K=K1,K2
      IR=IND(K,L)
      IF(L.EQ.1)IND(K,L)=0
      NZ=IW(IR,L)
      IF(NZ.LE.0)GO TO 630
      IN=IW(NZ,L+2)
      IW(IR,L+6)=IN
      IW(IR,L+4)=0
      IW(NZ,L+2)=IR
470   IF(IN.NE.0)IW(IN,L+4)=IR
475   K1=IP(IPP,1)+1
480   K2=IW(IPP,1)+K1-2
C
C RESET COLUMN FILE TO REFER TO U AND STORE ROW/COL NUMBERS IN
C     PIVOTAL ORDER IN IW(,,3),IW(,,4)
      DO 482 I=1,N
      J=-IW(I,5)
      IW(J,3)=I
      J=-IW(I,6)
      IW(J,4)=I
482   IW(I,2)=0
      DO 485 I=1,N
      KP=IP(I,1)
      KL=IW(I,1)+KP-1
```

```
      DO 485 K=KP,KL
      J=IND(K,2)
485   IW(J,2)=IW(J,2)+1
      K=1
      DO 487 I=1,N
      K=K+IW(I,2)
487   IP(I,2)=K
      LCOL=K-1
      DO 490 II=1,N
      I=IW(II,3)
      KP=IP(I,1)
      KL=IW(I,1)+KP-1
      DO 490 K=KP,KL
      J=IND(K,2)
      KN=IP(J,2)-1
      IP(J,2)=KN
490   IND(KN,1)=I
      GO TO 720
C
C     THE FOLLOWING INSTRUCTIONS IMPLEMENT THE FAILURE EXITS.
500   IF(LP.GT.0)WRITE(LP,510)IR,J
510   FORMAT(//34X,35HTHERE IS MORE THAN ONE ENTRY IN ROW,I5,
     1 11H AND COLUMN,I5)
      G=-4.
      GO TO 700
520   IF(LP.GT.0)WRITE(LP,530)
530   FORMAT(//34X,17HN IS NOT POSITIVE)
      G=-1.
      GO TO 700
540   IF(LP.GT.0)WRITE(LP,550)K,I,J
550   FORMAT(//34X,7HELEMENT,I7,10H IS IN ROW,I5,11H AND COLUMN,I5)
      G=-3.
      GO TO 700
580   IF(LP.GT.0)WRITE(LP,590)(RC(L,I),I=1,3),IR
590   FORMAT(//34X,3A1,I5,16H HAS NO ELEMENTS)
      G=-2.
      GO TO 700
600   IF(LP.GT.0)WRITE(LP,610)
610   FORMAT(//34X,15HIA IS TOO SMALL)
      G=-7.
      GO TO 700
```

```
630    IPV=IPV+1
       IW(IPV,1)=IR
       DO 640 I=1,N
       II=-IW(I,L+4)
640    IF(II.GT.0)IW(II,1)=I
       IF(LP.GT.0)WRITE(LP,650)(RC(L,I),I=1,3),(IW(I,1),I=1,IPV)
C650   FORMAT(48H ERROR RETURN FROM LA05AD BECAUSE THE FOLLOWING ,3A1 DI/
C     1,15HS ARE DEPENDENT/(20I5))                                   DI/
 650   FORMAT(48H ERROR RETURN FROM LA05A  BECAUSE THE FOLLOWING ,3A1
      1,15HS ARE DEPENDENT/(20I5))
       G=-5.
       GO TO 720
700    IF(LP.GT.0)WRITE(LP,710)
C710   FORMAT(33H+ERROR RETURN FROM LA05AD BECAUSE )                  DI/
 710   FORMAT(33H+ERROR RETURN FROM LA05A  BECAUSE )
720    RETURN
       END
```

```
C      SUBROUTINE LA05BD(A,IND,IA,N,IP,IW,W,G,B,TRANS)                  DI/
       SUBROUTINE LA05B (A,IND,IA,N,IP,IW,W,G,B,TRANS)
C      DOUBLE PRECISION A(IA),B(N),AM,W(N),G,SMALL                      DI/
                  REAL A(IA),B(N),AM,W(N),G,SMALL
       LOGICAL TRANS
C      INTEGER*2 IND(IA,2),IW(N,4)                                      IJ/
       INTEGER   IND(IA,2),IW(N,4)
       INTEGER IP(N,2)
C      COMMON/LA05DD/SMALL,LP,LENL,LENU,NCP,LROW,LCOL                   DI/
C IP(I,1),IP(I,2) POINT TO START OF ROW/COLUMN I OF U.
C IW(I,1),IW(I,2) ARE LENGTHS OF ROW/COL I OF U.
C IW(..3),IW(..4) HOLD ROW/COL NUMBERS IN PIVOTAL ORDER.
       COMMON/LA05D /SMALL,LP,LENL,LENU,NCP,LROW,LCOL
       IF(G.LT.0.)GO TO 400
       KLL=IA-LENL+1
       IF(TRANS)GO TO 300
C
C      MULTIPLY VECTOR BY INVERSE OF L
       IF(LENL.LE.0)GO TO 112
       L1=IA+1
       DO 110 KK=1,LENL
       K=L1-KK
       I=IND(K,1)
       IF(B(I).EQ.0.)GO TO 110
       J=IND(K,2)
       B(J)=B(J)+A(K)*B(I)
110    CONTINUE
112    DO 113 I=1,N
       W(I)=B(I)
113    B(I)=0.
C
C      MULTIPLY VECTOR BY INVERSE OF U
       N1=N+1
       DO 140 II=1,N
       I=N1-II
       I=IW(I,3)
       AM=W(I)
       KP= IP(I,1)
       IF(KP.GT.0)GO TO 130
       KP=-KP
       IP(I,1)=KP
```

```
      NZ=IW(I,1)
      KL=KP-1+NZ
      K2=KP+1
      DO 120 K=K2,KL
      J=IND(K,2)
120   AM=AM-A(K)*B(J)
130   IF(AM.EQ.0.)GO TO 140
      J=IND(KP,2)
       B(J)=AM/A(KP)
      KPC=IP(J,2)
      KL=IW(J,2)+KPC-1
      IF(KL.EQ.KPC)GO TO 140
      K2=KPC+1
      DO 135 K=K2,KL
      I=IND(K,1)
135   IP(I,1)=-IABS(IP(I,1))
140   CONTINUE
      GO TO 500
C
C     MULTIPLY VECTOR BY INVERSE OF TRANSPOSE OF U
300   DO 303 I=1,N
      W(I)=B(I)
303   B(I)=0.
      DO 315 II=1,N
      I=IW(II,4)
      AM=W(I)
      IF(AM.EQ.0.)GO TO 315
      J=IW(II,3)
      KP=IP(J,1)
      AM=AM/A(KP)
      B(J)=AM
      KL=IW(J,1)+KP-1
      IF(KP.EQ.KL)GO TO 315
      K2=KP+1
      DO 310 K=K2,KL
      I=IND(K,2)
310   W(I)=W(I)-AM*A(K)
315   CONTINUE
C
C     MULTIPLY VECTOR BY INVERSE OF TRANSPOSE OF L
      IF(KLL.GT.IA)RETURN
```

```
      DO 330 K=KLL,IA
      J=IND(K,2)
      IF(B(J).EQ.0.)GO TO 330
      I=IND(K,1)
      B(I)=B(I)+A(K)*B(J)
330   CONTINUE
      GO TO 500
C
400   IF(LP.GT.0)WRITE(LP,410)
C410  FORMAT(// 47H ERROR RETURN FROM LA05BD BECAUSE EARLIER ENTRY   DI/
C     1,18H GAVE ERROR RETURN)                                      DI/
 410  FORMAT(// 47H ERROR RETURN FROM LA05B  BECAUSE EARLIER ENTRY
     1,18H GAVE ERROR RETURN)
500   RETURN
      END
```

```
C      SUBROUTINE LA05CD(A,IND,IA,N,IP,IW,W,G,U,MM)                    DI/
       SUBROUTINE LA05C (A,IND,IA,N,IP,IW,W,G,U,MM)
C      DOUBLE PRECISION A(IA),G,U,AM,W(N),SMALL,AU                     DI/
                  REAL A(IA),G,U,AM,W(N),SMALL,AU
C      INTEGER*2 IND(IA,2),IW(N,4)                                     IJ/
       INTEGER   IND(IA,2),IW(N,4)
       INTEGER IP(N,2)
C      COMMON/LA05DD/SMALL,LP,LENL,LENU,NCP,LROW,LCOL                  DI/
       COMMON/LA05D /SMALL,LP,LENL,LENU,NCP,LROW,LCOL
       IF(G.LT.0.)GO TO 640
       JM=MM
C MCP LIMITS THE VALUE OF NCP PERMITTED BEFORE AN ERROR RETURN RESULTS.
       MCP=NCP+20
C REMOVE OLD COLUMN
       LENU=LENU-IW(JM,2)
       KP=IP(JM,2)
       IM=IND(KP,1)
       KL=KP+IW(JM,2)-1
       IW(JM,2)=0
       DO 40 K=KP,KL
        I=IND(K,1)
       IND(K,1)=0
       KR=IP(I,1)
       NZ=IW(I,1)-1
       IW(I,1)=NZ
       KRL=KR+NZ
       DO 10 KM=KR,KRL
       IF(IND(KM,2).EQ.JM)GO TO 20
10     CONTINUE
20     A(KM)=A(KRL)
       IND(KM,2)=IND(KRL,2)
40     IND(KRL,2)=0
C
C INSERT NEW COLUMN
       DO 110 II=1,N
       I=IW(II,3)
       IF(I.EQ.IM)M=II
C      IF(DABS(W(I)).LE.SMALL)GO TO 110                                DI/
       IF( ABS(W(I)).LE.SMALL)GO TO 110
       LENU=LENU+1
       LAST=II
```

```
      IF(LCOL+LENL.LT.IA)GO TO 50
C COMPRESS COLUMN FILE IF NECESSARY.
      IF(NCP.GE.MCP .OR. LENL+LENU.GE.IA)GO TO 600
C     CALL LA05ED(A,IND,IP(1,2),N,IW(1,2),IA  ,.FALSE.)                DI/
      CALL LA05E (A,IND,IP(1,2),N,IW(1,2),IA  ,.FALSE.)
50    LCOL=LCOL+1
      NZ=IW(JM,2)
      IF(NZ.EQ.0)IP(JM,2)=LCOL
      IW(JM,2)=NZ+1
      IND(LCOL,1)=I
      NZ=IW(I,1)
      KPL=IP(I,1)+NZ
      IF(KPL.GT.LROW)GO TO 55
      IF(IND(KPL,2).EQ.0)GO TO 90
C NEW ENTRY HAS TO BE CREATED.
55    IF(LENL+LROW+NZ.LT.IA)GO TO 60
      IF(NCP.GE.MCP .OR. LENL+LENU+NZ.GE.IA)GO TO 600
C COMPRESS ROW FILE IF NECESSARY.
C     CALL LA05ED(A,IND(1,2),IP,N,IW,IA  ,.TRUE.)                      DI/
      CALL LA05E (A,IND(1,2),IP,N,IW,IA  ,.TRUE.)
60    KP=IP(I,1)
      IP(I,1)=LROW+1
      IF(NZ.EQ.0)GO TO 80
      KPL=KP+NZ-1
      DO 70 K=KP,KPL
      LROW=LROW+1
      A(LROW)=A(K)
      IND(LROW,2)=IND(K,2)
70    IND(K,2)=0
80    LROW=LROW+1
      KPL=LROW
C PLACE NEW ELEMENT AT END OF ROW.
90    IW(I,1)=NZ+1
      A(KPL )=W(I)
      IND(KPL ,2)=JM
110   W(I)=0.
      IF(IW(IM,1).EQ.0 .OR. IW(JM,2).EQ.0 .OR. M.GT.LAST)GO TO 580
C
C FIND COLUMN SINGLETONS, OTHER THAN THE SPIKE. NON-SINGLETONS ARE
C     MARKED WITH W(J)=1. ONLY IW(..3) IS REVISED AND IW(..4) IS USED
C     FOR WORKSPACE.
```

```
      INS=M
      M1=M
      W(JM)=1.
      DO 150 II=M,LAST
      I=IW(II,3)
      J=IW(II,4)
      IF(W(J).EQ.0.)GO TO 140
      KP=IP(I,1)
      KL=KP+IW(I,1)-1
      DO 130 K=KP,KL
      J=IND(K,2)
130   W(J)=1.
      IW(INS,4)=I
      INS=INS+1
      GO TO 150
C PLACE SINGLETONS IN NEW POSITION.
140   IW(M1,3)=I
      M1=M1+1
150   CONTINUE
C PLACE NON-SINGLETONS IN NEW POSITION.
      IJ=M+1
      DO 160 II=M1,LAST
      IW(II,3)=IW(IJ,4)
160   IJ=IJ+1
C PLACE SPIKE AT END.
      IW(LAST,3)=IM
C
C FIND ROW SINGLETONS, APART FROM SPIKE ROW. NON-SINGLETONS ARE MARKED
C     WITH W(I)=2. AGAIN ONLY IW(..3) IS REVISED AND IW(..4) IS USED
C     FOR WORKSPACE.
      LAST1=LAST
      JNS=LAST
      W(IM)=2.
      J=JM
      DO 190 IJ=M1,LAST
      II=LAST  +M1-IJ
      I=IW(II,3)
      IF(W(I).NE.2.)GO TO 180
      K=IP(I,1)
      IF(II.NE.LAST)J=IND(K,2)
      KP=IP(J,2)
```

```
      KL=KP+IW(J,2)-1
      IW(JNS,4)=I
      JNS=JNS-1
      DO 170 K=KP,KL
      I=IND(K,1)
170   W(I)=2.
      GO TO 190
180   IW(LAST1,3)=I
      LAST1=LAST1-1
190   CONTINUE
      DO 195 II=M1,LAST1
      JNS=JNS+1
      I=IW(JNS,4)
      W(I)=3.
195   IW(II,3)=I
C
C DEAL WITH SINGLETON SPIKE COLUMN. NOTE THAT BUMP ROWS ARE MARKED BY
C     W(I)=3.
      DO 220 II=M1,LAST1
      KP=IP(JM,2)
      KL=KP+IW(JM,2)-1
      IS=0
      DO 210 K=KP,KL
      L=IND(K,1)
      IF(W(L).NE.3.)GO TO 210
      IF(IS.NE.0)GO TO 230
      I=L
      KNP=K
      IS=1
210   CONTINUE
      IF(IS.EQ.0)GO TO 580
C MAKE A(I,JM) A PIVOT.
      IND(KNP,1)=IND(KP,1)
      IND(KP,1)=I
      KP=IP(I,1)
      DO 215 K=KP,IA
      IF(IND(K,2).EQ.JM)GO TO 216
215   CONTINUE
216   AM=A(KP)
      A(KP)=A(K)
      A(K)=AM
```

```
      IND(K,2)=IND(KP,2)
      IND(KP,2)=JM
      JM=IND(K ,2)
      IW(II,4)=I
220   W(I)=2.
      II=LAST1
      GO TO 250
230   IN=M1
      DO 240 IJ=II,LAST1
      IW(IJ,4)=IW(IN,3)
240   IN=IN+1
250   LAST2=LAST1-1
      IF(M1.EQ.LAST1)GO TO 485
      DO 260  I=M1,LAST2
260   IW(I ,3)=IW( I,4)
      M1=II
      IF(M1.EQ.LAST1)GO TO 485
C
C CLEAR W
      DO 270 I=1,N
270   W(I)=0.
C
C PERFORM ELIMINATION
      IR=IW(LAST1,3)
      DO 480 II=M1,LAST1
      IPP=IW(II,3)
      KP=IP(IPP,1)
      KR=IP(IR,1)
      JP=IND(KP,2)
      IF(II.EQ.LAST1)JP=JM
C SEARCH NON-PIVOT ROW FOR ELEMENT TO BE ELIMINATED.
C  AND BRING IT TO FRONT OF ITS ROW
      KRL=KR+IW(IR,1)-1
      DO 290 KNP=KR,KRL
      IF(JP.EQ.IND(KNP,2))GO TO 300
290   CONTINUE
      IF(II-LAST1)480,580,480
C BRING ELEMENT TO BE ELIMINATED TO FRONT OF ITS ROW.
300   AM=A(KNP)
      A(KNP)=A(KR)
      A(KR)=AM
```

```
      IND(KNP,2)=IND(KR,2)
      IND(KR,2)=JP
      IF(II.EQ.LAST1)GO TO 310
C     IF(DABS(A(KP)).LT.U*DABS(AM))GO TO 310                              DI/
      IF( ABS(A(KP)).LT.U* ABS(AM))GO TO 310
C     IF(DABS(AM).LT.U*DABS(A(KP)))GO TO 330                              DI/
      IF( ABS(AM).LT.U* ABS(A(KP)))GO TO 330
      IF(IW(IPP,1).LE.IW(IR,1))GO TO 330
C PERFORM INTERCHANGE
310   IW(LAST1,3)=IPP
      IW(II,3)=IR
      IR=IPP
      IPP=IW(II,3)
      K=KR
      KR=KP
      KP=K
      KJ=IP(JP,2)
      DO 320 K=KJ,IA
      IF(IND(K,1).EQ.IPP)GO TO 325
320   CONTINUE
325   IND(K,1)=IND(KJ,1)
      IND(KJ,1)=IPP
330   IF(A(KP).EQ.0.)GO TO 580
      IF(II.EQ.LAST1)GO TO 480
      AM=-A(KR)/A(KP)
C COMPRESS ROW FILE UNLESS IT IS CERTAIN THAT THERE IS ROOM FOR NEW ROW.
      IF(LROW+IW(IR,1)+IW(IPP,1)+LENL.LE.IA)GO TO 340
      IF(NCP.GE.MCP .OR. LENU+IW(IR,1)+IW(IPP,1)+LENL.GT.IA)GO TO 600
C     CALL LA05ED(A,IND(1,2),IP,N,IW,IA  ,.TRUE.)                         DI/
      CALL LA05E (A,IND(1,2),IP,N,IW,IA  ,.TRUE.)
      KP=IP(IPP,1)
      KR=IP(IR,1)
340   KRL=KR+IW(IR,1)-1
      KQ=KP+1
      KPL=KP+IW(IPP,1)-1
C PLACE PIVOT ROW (EXCLUDING PIVOT ITSELF) IN W.
      IF(KQ.GT.KPL)GO TO 350
      DO 345 K=KQ,KPL
      J=IND(K,2)
345   W(J)=A(K)
350   IP(IR,1)=LROW+1
```

```
C
C TRANSFER MODIFIED ELEMENTS.
      IND(KR,2)=0
      KR=KR+1
      IF(KR.GT.KRL)GO TO 380
      DO 370 KS=KR,KRL
      J =IND(KS,2)
      AU=A(KS)+AM*W(J)
      IND(KS,2)=0
C IF ELEMENT IS VERY SMALL REMOVE IT FROM U.
C     IF(DABS(AU).LE.SMALL)GO TO 365                                    DI/
      IF( ABS(AU).LE.SMALL)GO TO 365
C     G=DMAX1(G,DABS(AU))                                               DI/
      G=AMAX1(G, ABS(AU))
      LROW=LROW+1
      A(LROW)=AU
      IND(LROW,2)=J
      GO TO 370
365   LENU=LENU-1
C REMOVE ELEMENT FROM COL FILE.
      K=IP(J,2)
      KL=K+IW(J,2)-1
      IW(J,2)=KL-K
      DO 366 KK=K,KL
      IF(IND(KK,1).EQ.IR)GO TO 367
366   CONTINUE
367   IND(KK,1)=IND(KL,1)
      IND(KL,1)=0
370   W(J)=0.
C
C SCAN PIVOT ROW FOR FILLS.
380   IF(KQ.GT.KPL)GO TO 435
      DO 430 KS=KQ,KPL
      J=IND(KS,2)
      AU=AM*W(J)
C     IF(DABS(AU).LE.SMALL)GO TO 430                                    DI/
      IF( ABS(AU).LE.SMALL)GO TO 430
      LROW=LROW+1
      A(LROW)=AU
      IND(LROW,2)=J
      LENU=LENU+1
```

```
C
C CREATE FILL IN COLUMN FILE.
      NZ=IW(J,2)
      K=IP(J,2)
      KL=K+NZ-1
C IF POSSIBLE PLACE NEW ELEMENT AT END OF PRESENT ENTRY.
      IF(KL.NE.LCOL)GO TO 390
      IF(LCOL+LENL.GE.IA)GO TO 400
      LCOL=LCOL+1
      GO TO 395
390   IF(IND(KL+1,1).NE.0)GO TO 400
395   IND(KL+1,1)=IR
      GO TO 425
C NEW ENTRY HAS TO BE CREATED.
400   IF(LCOL+LENL+NZ+1.LT.IA)GO TO 410
C COMPRESS COLUMN FILE IF THERE IS NOT ROOM FOR NEW ENTRY.
      IF(NCP.GE.MCP .OR. LENU+LENL+NZ+1.GE.IA)GO TO 600
C     CALL LA05ED(A,IND,IP(1,2),N,IW(1,2),IA  ,.FALSE.)                DI/
      CALL LA05E (A,IND,IP(1,2),N,IW(1,2),IA  ,.FALSE.)
      K=IP(J,2)
      KL=K+NZ-1
C TRANSFER OLD ENTRY INTO NEW.
410   IP(J,2)=LCOL+1
      DO 420 KK=K,KL
      LCOL=LCOL+1
      IND(LCOL,1)=IND(KK,1)
420   IND(KK,1)=0
C ADD NEW ELEMENT.
      LCOL=LCOL+1
      IND(LCOL,1)=IR
C425  G=DMAX1(G,DABS(AU))                                              DI/
 425  G=AMAX1(G, ABS(AU))
      IW(J,2)=NZ+1
430   W(J)=0.
435   IW(IR,1)=LROW+1-IP(IR,1)
C
C STORE MULTIPLIER
      IF(LENL+LCOL+1.LE.IA)GO TO 450
C COMPRESS COL FILE IF NECESSARY.
      IF(NCP.GE.MCP)GO TO 600
C     CALL LA05ED(A,IND,IP(1,2),N,IW(1,2),IA  ,.FALSE.)                DI/
```

```
      CALL LA05E (A,IND,IP(1,2),N,IW(1,2),IA  ,.FALSE.)
450   K=IA-LENL
      LENL=LENL+1
      A(K)=AM
      IND(K,1)=IPP
      IND(K,2)=IR
C CREATE BLANK IN PIVOTAL COLUMN.
      KP=IP(JP,2)
      NZ=IW(JP,2)-1
      KL=KP+NZ
      DO 460 K=KP,KL
      IF(IND(K,1).EQ.IR)GO TO 465
460   CONTINUE
465   IND(K,1)=IND(KL,1)
      IW(JP,2)=NZ
      IND(KL,1)=0
      LENU=LENU-1
480   CONTINUE
C
C CONSTRUCT COLUMN PERMUTATION AND STORE IT IN IW(,,4)
485   DO 490 II=M,LAST
      I=IW(II,3)
      K=IP(I,1)
      J=IND(K,2)
490   IW(II,4)=J
      GO TO 720
C
C     THE FOLLOWING INSTRUCTIONS IMPLEMENT THE FAILURE EXITS.
580   IF(LP.NE.0)WRITE(LP,590)MM
590   FORMAT(//34X,45HSINGULAR MATRIX CREATED BY REPLACEMENT OF COL,I5)
      G=-6.
      GO TO 700
600   IF(LP.NE.0)WRITE(LP,610)
610   FORMAT(//34X,15HIA IS TOO SMALL)
      G=-7.
      GO TO 700
640   IF(LP.NE.0)WRITE(LP,650)
650   FORMAT(//34X,31HEARLIER ENTRY GAVE ERROR RETURN)
700   IF(LP.NE.0)WRITE(LP,710)
C710  FORMAT(33H+ERROR RETURN FROM LA05CD BECAUSE )                    DI/
 710  FORMAT(33H+ERROR RETURN FROM LA05C  BECAUSE )
```

```
  720   RETURN
        END
```



```
        BLOCK DATA
C       COMMON/LA05DD/SMALL,LP,LENL,LENU,NCP,LROW,LCOL                  DI/
        COMMON/LA05D /SMALL,LP,LENL,LENU,NCP,LROW,LCOL
C       DOUBLE PRECISION SMALL                                          DI/
        REAL SMALL
C       DATA SMALL,LP/0.D0,6/                                           DI/
        DATA SMALL,LP/0.0,6/
        END
```

```
C     SUBROUTINE LA05ED(A,IRN,IP,N,IW,IA,REALS)                          DI/
      SUBROUTINE LA05E (A,IRN,IP,N,IW,IA,REALS)
      LOGICAL REALS
C     DOUBLE PRECISION A(IA),SMALL                                       DI/
                REAL A(IA)
C     INTEGER*2 IRN(IA),IW(N)                                            IJ/
      INTEGER   IRN(IA),IW(N)
      INTEGER IP(N)
C     COMMON/LA05DD/SMALL,LP,LENL,LENU,NCP,LROW,LCOL                     DI/
      COMMON/LA05D /SMALL,LP,LENL,LENU,NCP,LROW,LCOL
      NCP=NCP+1
C     COMPRESS FILE OF POSITIVE INTEGERS. ENTRY J STARTS AT IRN(IP(J))
C  AND CONTAINS IW(J) INTEGERS,J=1,N. OTHER COMPONENTS OF IRN ARE ZERO.
C  LENGTH OF COMPRESSED FILE PLACED IN LROW IF REALS IS .TRUE. OR LCOL
C  OTHERWISE.
C  IF REALS IS .TRUE. ARRAY A CONTAINS A REAL FILE ASSOCIATED WITH IRN
C  AND THIS IS COMPRESSED TOO.
C  A,IRN,IP,IW,IA ARE INPUT/OUTPUT VARIABLES.
C  N,REALS ARE INPUT/UNCHANGED VARIABLES.
C
      DO 5 J=1,N
C STORE THE LAST ELEMENT OF ENTRY J IN IW(J) THEN OVERWRITE IT BY -J.
      NZ=IW(J)
      IF(NZ.LE.0)GO TO 5
      K=IP(J)+NZ-1
      IW(J)=IRN(K)
      IRN(K)=-J
5     CONTINUE
C KN IS THE POSITION OF NEXT ENTRY IN COMPRESSED FILE.
      KN=0
      IPI=0
      KL=LCOL
      IF(REALS)KL=LROW
C LOOP THROUGH THE OLD FILE SKIPPING ZERO (DUMMY) ELEMENTS AND
C     MOVING GENUINE ELEMENTS FORWARD. THE ENTRY NUMBER BECOMES
C     KNOWN ONLY WHEN ITS END IS DETECTED BY THE PRESENCE OF A NEGATIVE
C     INTEGER.
      DO 25 K=1,KL
      IF(IRN(K).EQ.0)GO TO 25
      KN=KN+1
      IF(REALS)A(KN)=A(K)
```

```
      IF(IRN(K).GE.0)GO TO 20
C END OF ENTRY. RESTORE IRN(K), SET POINTER TO START OF ENTRY AND
C     STORE CURRENT KN IN IPI READY FOR USE WHEN NEXT LAST ENTRY
C     IS DETECTED.
      J=-IRN(K)
      IRN(K)=IW(J)
      IP(J)=IPI+1
      IW(J)=KN-IPI
      IPI=KN
20    IRN(KN)=IRN(K)
25    CONTINUE
      IF(REALS)LROW=KN
      IF(.NOT.REALS)LCOL=KN
      RETURN
      END
```

References

D. Goldfarb and J.K. Reid (1975). A practicable steepest edge simplex algorithm. Harwell report C.S.S.19.

D. Goldfarb and J.K. Reid (1976). Fortran subroutines for sparse in-core linear programming. Harwell report to appear.

F.G. Gustavson (1972). Some basic techniques for solving sparse systems of linear equations. In "Sparse matrices and their applications", ed. D.J. Rose and R.A. Willoughby. Plenum Press.

H.M. Markovitz (1957). The elimination form of the inverse and its applications to linear programming. Management Sci., 3, 255-269.

J.K. Reid (1975). A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. Harwell report C.S.S. 20.

B.G. Ryder (1973). The FORTRAN verifier: user's guide. Computing Science Technical Report # 12, Bell Telephone Laboratories.