1973

GEODÆTISK INSTITUTS INTERNE RAPPORT NR. 8

THE DANISH GEODETIC INSTITUTE

INTERNAL REPORT NO. 8

CHOLESKY's METHOD ON A COMPUTER

by

K. Poder and C.C. Tscherning

KØBENHAVN 1973

GEODÆTISK INSTITUTS INTERNE RAPPORT NR. 8

THE DANISH GEODETIC INSTITUTE

INTERNAL REPORT NO. 8

CHOLESKY's METHOD ON A COMPUTER

by

K. Poder and C. C. Tscherning

KØBENHAVN 1973

Contents

Abstract

Cholesky's method is an algorithm for the solution of a system of linear equations $Ax = y$, where A is a symmetric positive definite n x n matrix, x and y are n-vectors.

The report deals with the inplementation of the method on a digital computer. The numerical and the storing problems are especially treated. In the treatment of the numerical problems the ideas of Wilkinson (Rounding Errors in Algebraic Processes) are followed.

A strategy proposed by Andersen and Krarup (Linear Equations) for the storing of the matrix A is used, being well suited for sparse matrices. An implementation of the method as a subroutine written in FORTRAN IV of the IBM 360/70 system is described. The implementation facilitates the computation of the quadratic form $yA^{-1}y$ or of the columns of A^{-1} without additional space requirements.

I. Introduction

We consider a n x n matrix A, which is symmetric and positive definite. The matrix can be the coefficient matrix of a system of "normal-equations"

$$Ax = y \tag{1}$$

or can be the inverse of a matrix of a quadratic form

$$v = y^T A^{-1} y \ . \tag{2}$$

We shall treat some of the problems which arize, when we use a computer for the solution of equation (1), the computation of (2) and on related problems. We take as our point of departure Cholesky's method for the solution of the equations (1), as this method includes the computation of quantities like (2).

The implementation of the algorithm itself in FORTRAN or ALGOL is no problem. But as known computers have limited core storage capacity in the central processing units (CPU) and use a finite number of bits in the representation of reals and integers.

The limited capacity of the core storage puts a limit on the dimension on A, in case we want to have A stored in core only. We may also face the problem, that the use of big parts of the CPU for storage, may give a job a low priority and hence a long turn-around time.

In spite of the assumption of A being positive definite, some of the eigenvalues of A may be so different, that the actual computer will not have a sufficient number of digits in the computation to deal properly with the rounding errors. This problem has been treated by Wilkinson. (See [5], pp. 91-93, and [1] p. 14).

A "numerical singularity" may occur in practice, e.g. in geometrical geodesy due to lack of observations or in physical geodesy (collocation) due to multiply introduced observations. A method of solution, therefore should preferably be able to indicate the existence of an error and continue the computation (there may be more than one error) in order to give the user a full

list of corrections in one single run. This kind of numerical singularity is easily detected and taken care of during the procedure, because the Cholesky-method will at certain points give us some relevant subdeterminants in hand, namely those corresponding to the k first coefficients of the k first columns (the k'th main-subdeterminant of A, D_k). The subdeterminant D_{k+1} is computed as the product of D_k and a quantity $l^2_{k+1,k+1}$,

$$l^2_{k+1,k+1} = \frac{D_{k+1}}{D_k} , \qquad \text{(see (6b))}.$$

This quantity is essential within numerical investigation of our problem. When it becomes distorted or approaches zero, the result of the computation becomes invalid (at least for the pertaining row and column).

II. Cholesky's Algorithm

We have taken the main principles of the following from [1] and [5] . As known we may write a symmetric, positive matrix as the product of a lower triangular matrix and its transpose

$$A = L \cdot L^T . \tag{3}$$

The equations (1) are formally solved by forming

$$L^{-1} \cdot A \cdot x = L^{-1} y, \tag{4}$$

which gives

$$L^T x = L^{-1} y. \tag{5}$$

However, the numerical task we are facing, actually is to determine L, so that

$$A = LL^T + E \tag{3a}$$

where E is a n x n matrix as close to a zero matrix as possible.

The computation of the upper triangular $L^T = L^{-1}A$ and of the column $L^{-1}y$ is denoted the reduction and the solution of (5) the back-solution.

The elements of the reduced matrix L^T are computed by the algorithm

$$l_{jk} = (a_{jk} - \sum_{m=1}^{k-1} l_{jm} \cdot l_{km}) / l_{kk} \qquad j \neq k, \ (k \leq j) \tag{6a}$$

$$l_{jj} = \sqrt{a_{jj} - \sum_{m=1}^{j-1} l_{jm}} \qquad k = j \tag{6b}$$

((6b) is in fact included in (6a)).

The element l_{jk} may be computed, when all the elements l_{nm}, $n < j$ and $m < k$ are known. Hence we may proceed by computing the elements l_{jk} either column after column or row after row. We also see, that it is not necessary to store both A and L^T. As soon as l_{jk} has been computed, we can store it in the location of a_{jk}.

All the elements l_{jk} of L are formed from product sums (cf. (6)), which of course have rounding errors. The now classical idea of Wilkinson is to use (have) the elements of A in one precision (i.e. number of significant digits) and the elements of L in half the precision. Such a computation involving numbers of different precision is unfortunately not too well supported by the existing computers and program systems.

As a compromise we may formally extend the precision of the elements of A from a single precision representation to a double precision representation by placing a number of zeroes behind the digits. (Example: 2.352165 is transformed to 2.352165 0000000). This extension only have to take place one time for an element, a_{kj} namely at the time of its reduction. When the reduced element l_{jk} has been computed, it will be rounded to single precision and stored at the place a_{kj}. This means, that A should not be stored in double precision in this case.

(Note, that in many least squares problems leading to "normal equations" (e.g. adjustment problems in geometrical geodesy) the elements a_{kj} will generally be formed by product sums and will hence have to be stored in double precision from the beginning).

The column $L^{-1}y$ is computed using the algorithm (6), regarding the "right hand side" y as an extra column of the matrix A:

$$A_o = \left\{ \begin{array}{c|c} A & y \\ \hline y^T & d \end{array} \right\}_{n+1,n+1}$$

d is a fictitious diagonal element, which we in some situations may give a specific value - e.g. a zero or the square sum of the elements of the column y. Using the algorithm (6) on A_o, we see, that

$$l^2_{n+1,n+1} = d - \sum_{m=1}^{n} l^2_{n+1,m} = d - y^T (L^T)^{-1} L^{-1} y = d - y^T A^{-1} y.$$

Hence for d = 0 we get minus the value of the quadratic form (2).

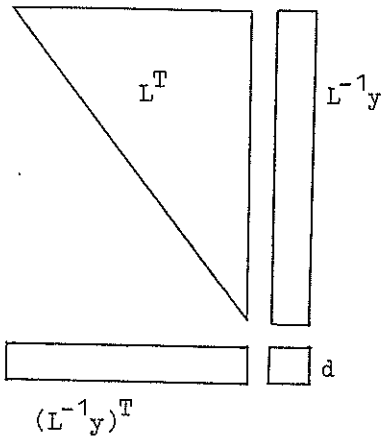

Figure 1

This quantity is of special interest in geodesy. In geometrical geodesy it is the variance, and in a prediction or collocation process of physical geodesy it may be the estimated error of prediction.

With L^T stored permanently, it is possible to solve the linear equations (1) or to compute (2) for different vectors y, by extending the reduction process (6) to an extra column. It may also be possible to extend the matrix A with a number of new columns, and then start the reduction process from the first new column.

In the computation of A^{-1} it is necessary to solve (1) for different right hand sides. As known we get the j'th column of A^{-1}, when we as right hand side in (1) use an y with a 1 in the j'th row and zeroes elsewhere. Note, that in this case, the reduction of the y-vector will not have to start before row j. From (6) we see, that the rows 1 to j-1 will not be changed.

In many cases we may take advantage of these features: a. the back-solution is not always needed (it is certainly not necessary to do the cumbersome matrix

inversion for finding a quadratic form as (2)), b. the reduction process (6) may begin in an arbitrary column and row.

Hence for an ALGOL-procedure or FORTRAN-subroutine implementation of the Cholesky-method, parameters in a call should include (a) a boolean (logical), true in case of back-solution and false otherwise and (b) e.g. the number of the fully reduced row and of the last fully reduced column. (These parameters are called LBS, IIFR, IIFC in the FORTRAN-subroutine shown in the appendix).

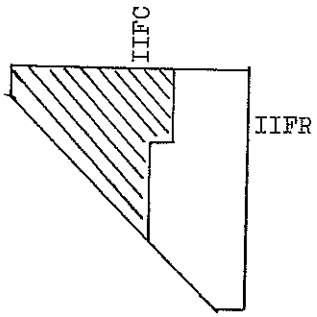

Figure 2a

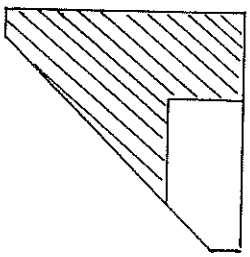

Figure 2b

Shaded part is reduced

Examples of two possible situations at a restart of the reduction process. (We have choosed only to consider the situation described in Figure 2b).

III. Storing the A_o and L^T Matrices

We work either with the symmetric matrix A_o or the upper triangular matrix L^T. Hence, it is only necessary to store the upper triangular part of A_o. As mentioned above, the L^T elements are successively stored in the area reserved for A_o.

Let us suppose, that we have stored the upper triangular part of the matrix A_o or L^T in a one-dimensional array C, column after column with the elements of the first row having the smallest subscript. We store $a_{1,1}$ in C(1), $a_{1,2}$ in C(2), $a_{2,2}$ in C(3) etc.

For matrices A_o with many coefficients equal to zero, storing may be even more compact. As noted above concerning the computation of the j'th column of A^{-1}, the reduced right hand side $L^{-1}y$ will have the first j-1 rows unchanged, because the j-1 first elements of y are equal to zero. The same is true for columns of A_o, for which the coefficients in a number of the first rows are equal to zero. - The corresponding elements of L^{-1} will be zero too. It is hence not necessary to store these coefficients.

1	-1	0	0	0	0	0	3
	2	1	-1	0	0	2	0	6
		3	0	0	0	-1	0	-1
			4	-2	0	3	0	2
				5	3	2	2	5
					6	-1	3	0
						7	-4	-2
							8	-3
								0

Upper triangular part of A_o

Array C

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	-1	2	1	3	-1	0	4	-2	5	3	6	2	-1	3	2	-1

18	19	20	21	22	23	24	25	26	27	28	29	30	31
7	2	3	-4	8	3	6	-1	2	5	0	-2	-3	0

Catalogue of subscripts of diagonal elements NCAT:

0	1	3	5	8	10	12	18	22	31

Catalogue of saved (ignored) zeroes ISZE:

0	0	1	1	3	4	1	3	0

Figure 3

However, it is necessary to establish a catalogue of the content of array C. A very practical method is to "flag" the diagonal elements. This may be done on some computers, but generally it is not possible to operate on the single bits of the memory cells in a high level language. Another possibility is to store the subscript of the diagonal element of column k+1. Such a method will require the storage of integers and reals together and will give a more complicated procedure for the computation of the address of elements of A_o stored in C.

We have hence chosen to establish a catalogue of the subscripts of the diagonal elements. The catalogue is stored in an integer array NCAT, and NCAT(k) will be the subscript of the diagonal element of column k-1. The element a_{kj} will then be

$$a_{kj} = C(NCAT(j+1)-j+k)$$

cf. Figure 3. The value NCAT(j+1)-j is a constant for a column and should be kept in a simple variable during the reduction of the column.

In the reduction (6) we see, that the sum of products

$$\sum_{m=1}^{j-1} l_{km} \cdot l_{mj}$$

only need to be formed from the summation subscript n, where both l_{km} and l_{jm} are different from zero. The number of elements different from zero in columns k and j are k-NCAT(k+1)-NCAT(k) and j-NCAT(j-1)-NCAT(j), respectively. These numbers are constants for the columns, and are stored in the integer array ISZE (= saved-zeroes), cf. Figure 3.

We shall now consider the situation, where A_o becomes very big in comparison with the available core storage. It is then necessary to divide A_o in records (blocks), which should be transferred to and from a peripheral (backing) store: drum, disc, magnetic tape, paper tape, or even punched cards. A record will generally contain as many columns as possible, but no column is divided over two records. This is of course very useful for the computation of the product sums.

Generally it will facilitate the transport to and from a backing store,

when the records are of equal length. This will give some unused space, in case the columns can't fill a whole record. Furthermore the catalogues NCAT and ISZE are given a fixed length, thus setting an upper limit of the number of columns contained in one record. (Cf. the above remarks on other strategies for mapping the matrix A_o and the catalogues on a simple array).

It is necessary to have access to two different columns - and hence in most cases to two different records - at the same time. Two one-dimensional arrays of each kind will have to be declared: C, NCAT, ISZE and CR, NRCAT, IRSZE. The R indicating, that the matrix contains elements of the reduced matrix L^T.

It will be practical to establish a directory catalogue containing information about the distribution of the columns within the file formed by the records. The catalogue (called NBL) may for example have NBL(I) equal to the number of the last column in record I-1.

Let us, for the matrix A_o of Figure 3, suppose that we can use a file divided in 4 records. As an example in each record the first 10 elements are used for A_o, the next five for NCAT and the last five for ISZE. The used magnitude of NCAT restricts the maximal number of columns to be stored within a record to 4. We then end up with the situation described in Figure 4.

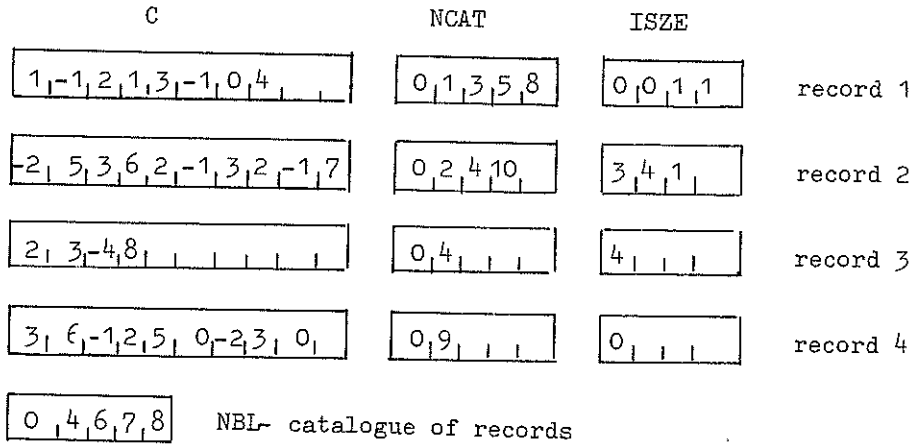

Figure 4. Storing of the matrix of Fig. 3 in 4 records

Naturally, the records will be much larger in practical computations. This will reduce the number of transports to and from the backing store. For an implementation of the algorithm on the GIER-computer (with a CPU of 1024 x 40 bit words) the records had a length of 440 "words". For a disc unit (IBM 3330) a record could e.g. be of a length of 1600 x 64 bit words.

IV. Description of the Subroutine NES

We have decided (to try) to implement the Cholesky method using the principles described above. The implementation has been done on an IBM 360/70 computer system using the FORTRAN IV language and the so-called H-extended compiler. This compiler allows the use of quadruple precision (128 bit = 16 byte real number representation).

The method has been implemented on the GIER-computer of the Danish Geodetic Institute (since 1963) written in assembler language (see [2]), Algol-procedures, thus not containing all the facilities described above, may also be found in the program library of the institute.

Let us suppose, that a program "MAIN" is responsible for the storing of A_o the mentioned catalogues etc. A direct access file (named file 8) is created on a disc unit (IBM 3330). The file is divided in e.g. 25 records of length 12800 bytes, corresponding to a REAL * 8 (byte) array C of dimension 1500 and INTEGER * 4 (byte) arrays NCAT and ISZE of dimension 100.

(The necessary commands used in the operating system OS at the NEUCC installation are

```
//FT08F001  DD  DSN=DASET,UNIT=3330,VOL=SER=MVTWK1,
//DCB=DSORG=DA,SPACE=(12800,25),DISP=(,DELETE)
```

and the statement

```
DEFINE FILE 8(25,3200,U,ID)
```

in MAIN. (25 = number of records, 3200 x 4 = record length, U = unformatted and ID an integer which points at the next record to be read or written)).

The following information will furhtermore have to be transferred to the subroutine: a) The array NBL, b) the number of records used for storing A_o (called MAXBL), c) the pointer ID, d) the arrays C, NCAT and ISZE used to transfer the solution (stored at the position of the right hand side of A) to MAIN, e) the dimension of A_o (called N), f) the number of the row and column of the last reduced element (IIFC, IIFR), g) the boolean (logical) LBS indicating

back-solution and finally h) the value of the quadratic form (2) (called PW). We have decided to use the variables N, IIFC, IIFR, LBS and PW as parameters in the call of the subroutine:

NES(N,IIFR,IIFC,LBS,PW).

The other variables are transferred to NES through a COMMON block named NESOL.

We note, that the catalogue NBL is not absolutely necessary. But it is very useful, for finding the first column containing unreduced elements. The subprogram must begin with a search through NBL with the purpose of finding the record containing this column (record number JBF). The content of this record will be stored in the arrays C, NCAT and ISZE. When the part of A_o stored in C has been reduced it is transferred to the record, the content of the next record is transferred and reduced, and so on ... . The integer JBL is the number of the record which is being reduced. The reduced elements and their related catalogues are stored in CR, NRCAT and IRSZE. The content of the records JBF to JBL will then successively be stored in these arrays.

We number the columns in C by J, running from 1 to NC and the columns in CR by K, running from 1 to NR. Generally not all the elements of column J can be reduced at the same time (in the same loop). Only the rows corresponding to the columns in CR can be treated. Hence, it is useful to reduce all the elements of the columns in C corresponding to the columns in CR and not to reduce a column from the top to the bottom. (Cf. the flow-chart of Figure 5).

The columns of C and CR are paired according to the algorithm (6). Let us regard one element C(I), which is an element of the J'th column of the record JBL. It will hence be an element of column JD=NBL(JBL)+J of A_o. The row of A_o is KD=NCAT(J+1)-I. Let us suppose, that the column KD is stored in record KBL, column K. A sum of products thus is formed by a summation running at most from row 1 to KD-1, but as mentioned above the summation will only have to run from

MAX(IRSZE(K), IRSZE(J), IIFR)+1 . (7)

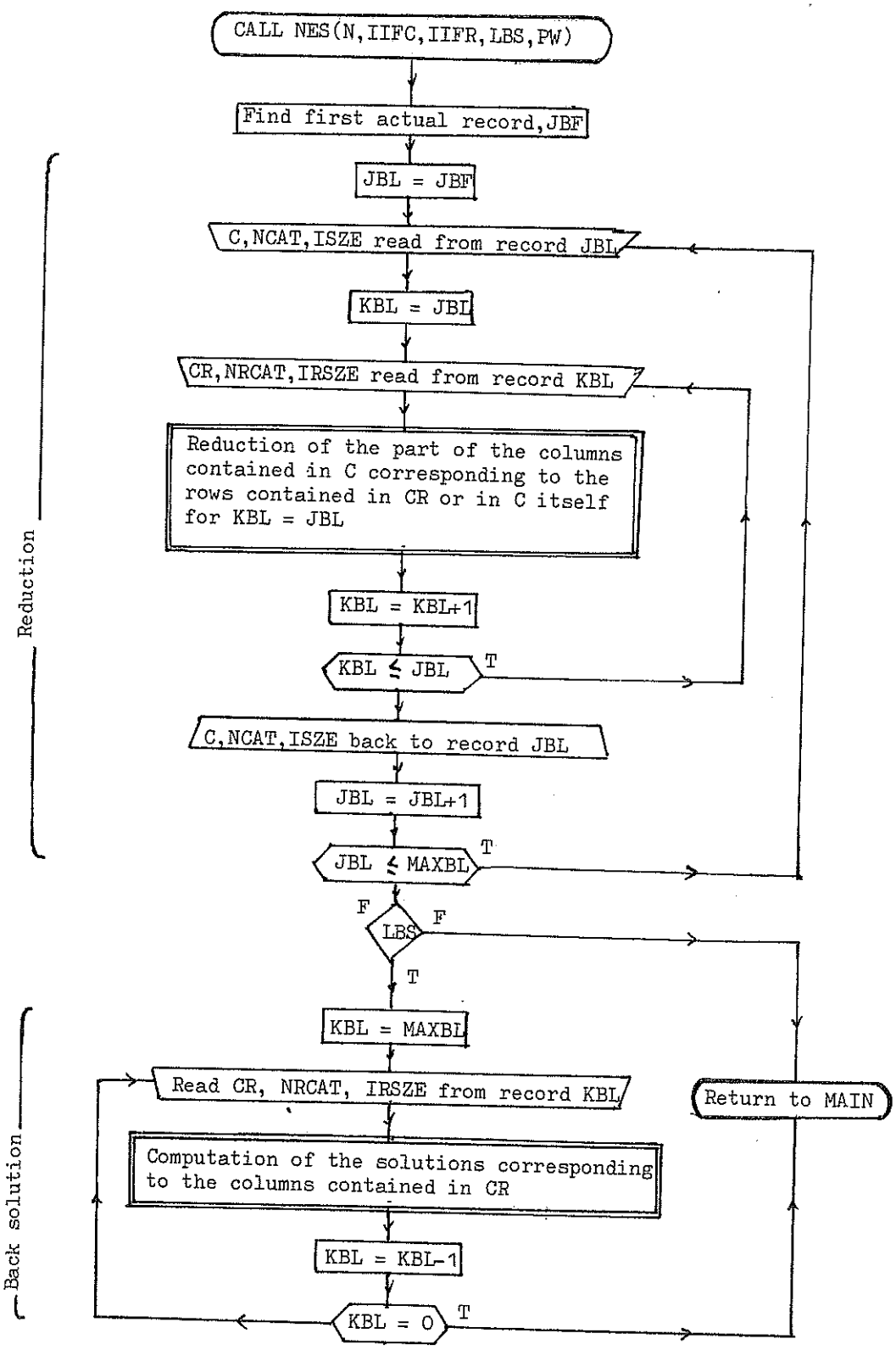

Figure 5. Flow-chart of NES.

T = TRUE, F = FALSE

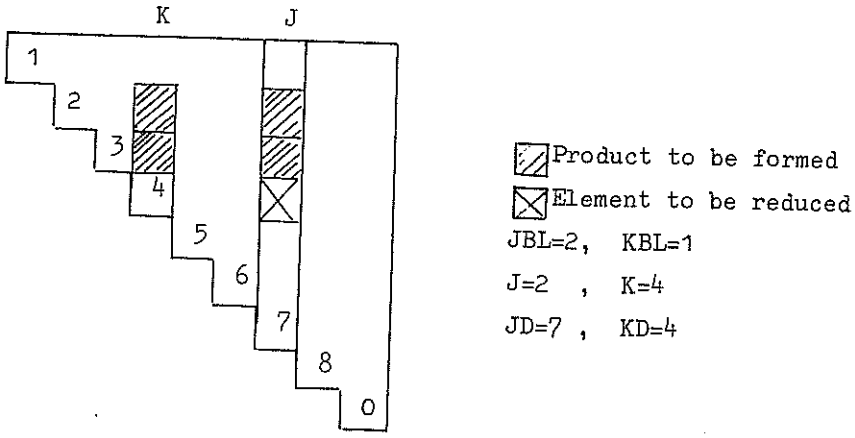

Figure 6. Reduction of $a_{4,7}$. The product sum contains only two products.

The reduction of the diagonal element of a column includes the computations of a square-root. Before doing this, we must be sure, that the difference (6b) is positive. When the difference is very "small" or negative, an error has been detected.

The equation (3a) could also be written as

$$A = (L+E_L)\cdot(L^T+E_L^T) ,$$

where E_L must be as close as possible to a zero-matrix in single precision. Obviously l_{nm} is too small, when it has lost nearly all significant digits corresponding to single precision.

In this connection we note a second advantage of using the two catalogues NCAT, ISZE. We may as mentioned want to continue the computations, when a numerical singularity has occured (e.g. in column JD). This can be done putting ISZE(J)=JD, because the summation start value (7) then will be greater than the summation limit. The elements in row JD of the succeeding columns will be put equal to zero.

When this test of "numerical singularity" has been done, we can take the square-root of all $l^2_{n,m}$, except perhaps for the last column in (the extended matrix) A_o, which contains the fictitious diagonal element noted d.

When the right hand side has been reduced, two situations may occur

(depending of the value of the logical LBS). If the value of the quadratic form (2) is wanted, no more computations are needed, and the result is found in the diagonal element corresponding to the last column. Otherwise we must start the back-solution. This will take place in a very simple way. The unknown x_k are computed successively from the last element x_{n-1} to the first x_1: $x_k=l'_{k,n}/l_{k,k}$, $l'_{k,n}=l_{k,n}-\sum_{m=k+1}^{n-1} l_{m,k}\cdot x_m$. The value of x_k is multiplied with the reduced coefficients of column k, and the value are subtracted from the reduced elements on the right hand side in rows 1 to k-1. In this way a new upper-triangular equation is formed of dimension k-1. The solution x_k can be stored at the place of $l_{k,n}$ and are finally transferred to MAIN using the array C of the COMMON-block NESOL.

The subroutine can e.g. be called using the following parameters:

Solution of the equations (1):

```
CALL NES(N,0,0,.TRUE.,DUMMY).
```

The solution x_k are found in C(I), where

```
I=NCAT(N-NBL(MAXBL))+k.
```

When a first call of NES has taken place, the reduced matrix L^T is stored in the file.

The computation of the quadratic form (2) is in this case very simple:

```
CALL NES(N,N-1,0,.FALSE.,PW)
```

and

PW will contain the value of (2). The y-vector should be stored at the place of the original right hand sides.

The Q'th diagonal element or the Q'th column of A^{-1} is computed under the same conditions, having an y-vector with only zeroes except in row Q, where a real 1.0 must be placed:

```
CALL NES(N,N-1,Q,.FALSE.,PW).
```

PW will be equal to the diagonal element, and the Q'th column of A^{-1} can be found in the array C.

V. Final Remarks

The subroutine NES found in the appendix has been worked out as an illustration of the ideas outlined. It has not been tested in all detaills. In fact we think that the algorithm should be written in assembler language.

The ideas outlined should make it possible to solve very big systems of equations. - Thus the rounding errors setting an upper limit on the number of equations, rather than the space available in a computer.

Literature

[1] Andersen, Chr. and Torben Krarup: Linear Equations. (In Chr. Gram (ed.), Selected Numerical Methods, Copenhagen 1962.)

[2] Andersen, O.B., E. Kejlsø, K. Poder and P.M. Thomsen: EDB-Rapport, Geodætisk Afdeling I, DEC 1971, København 1972 (The Danish Geodetic Institute Internal Report No. 1).

[3] IBM System 360 and System 370 FORTRAN IV Language, 10. Ed., 1972.

[4] Krarup, T.: The Theory of Rounding Errors in the Adjustment by Elements of Geodetic Networks. Bulletin Géodésique, No. 84, 1967.

[5] Wilkinson, J.H.: Rounding Errors in Algebraic Processes, 1963.

Appendix

```
      SUBROUTINE NES(NN,IIFC,IIFR,LBS,PW)
      IMPLICIT INTEGER(I,J,K,M,N),REAL *8(A-H,O,P,R-Z),LOGICAL(L),
     *REAL *16(Q)
      DIMENSION CR(1500),NRCAT(100),IRSZE(100)
      COMMON/NESOL/C(1500),NCAT(100),ISZE(100),NBL(25),MAXBL,ID
C THE SUBROUTINE WILL, USING THE CHOLESKYS METHOD
C (1) COMPUTE THE REDUCED MATRIX L CORRESPONDING TO A SYMMETRIC POSITIVE
C     DEFINITE (NN-1)*(NN-1) MATRIX A, WHEN THE IIFC COLUMNS AND IIFR
C     ROWS OF L ARE KNOWN, (L*LT = A, LT THE TRANSPOSED OF L).
C (2) COMPUTE THE REDUCED NN-1 VECTOR (L**-1)*Y.
C (3) COMPUTE THE DIFFERENCE PW = YN-YT*(A**-1)*Y.
C (4) SOLVE THE EQUATIONS LT*X = (L**-1)*Y,(THE SO CALLED BACK-SOLUTION)
C
C THE REDUCED ROWS AND COLUMNS OF L (THERE MAY BE NONE), THE CORRES-
C PONDING UNREDUCED UPPER TRIANGULAR PART OF A, THE NN-VECTOR FORMED
C BY Y AND YN FORMS AN UPPER TRIANGULAR NN*NN-MATRIX.
C THE MATRIX IS STORED COLUMNVIZE IN MAXBL RECORDS OF FILE 8. A RECORD
C (12800 BYTES) CONTAINS AS MANY COLUMNS AS POSSIBLE IN THE FIRST 8*1500
C BYTES (CORRESPONDING TO THE MAGNITUDE OF THE TRANSPORT ARRAYS C AND
C CR) AND THE TWO CATALOGUES NCAT AND ISZE (OR NRCAT AND IRSZE) IN THE
C LAST 4*200 BYTES.
C WHEN THE CONTENT OF A RECORD HAS BEEN TRANSFERRED E.G. TO C, NCAT,
C ISZE, WE HAVE THE FOLLOWING SITUATION. THE COLUMNS ARE STORED IN C
C FROM THE FIRST ELEMENT DIFFERENT FROM ZERO TO THE DIAGONAL ELEMENT.
C NCAT(I) IS THE SUBSCRIPT OF THE DIAGONAL ELEMENT OF COLUMN I-1 AND
C ISZE(I) IS THE NUMBER OF IGNORED (SAVED) ZEROES IN COLUMN I. NCAT(100)
C IS THE NUMBER OF COLUMNS STORED IN THE RECORD.
C NBL(I) IS EQUAL TO THE NUMBER OF THE LAST COLUMN STORED IN RECORD I-1.
C (1) TO (3) ABOWE WILL ALWAYS BE EXECUTED, BUT (4) WILL ONLY BE EXE-
C CUTED WHEN THE LOGICAL LBS IS TRUE. THE EXECUTION OF (1), (2) AND (4)
C IS EQUIVALENT TO THE SOLUTION OF THE EQUATIONS A*X=Y. THE SOLUTIONS
C WILL BE STORED IN THE ARRAY C IN THE POSITIONS ORIGINALLY OCCUPIED BY
C Y AND WILL BE TRANSFERRED TO MAIN THROUGH THE COMMON BLOCK.
C IN CASE A NUMERICAL SINGULARITY OCCURS IN COLUMN NUMBER JD, THE COLUMN
C IS DELETED BY CHANGING THE CATALOGUE ISZE AND THE ELEMENTS IN ROW JD
C AND THE UNKNOWN CORRESPONDING TO ROW JD ARE PUT EQUAL TO ZERO.
C
      N = NN
      IFR=IIFR+1
      IFC=IIFC+1
C
C REDUCTION OF COLUMNS IFC TO N. ELEMENTS, WHICH ARE ALREADY REDUCED,
C ARE STORED IN CR (EXCEPT FOR JBL=KBL). ELEMENTS, WHICH ARE GOING TO BE
C REDUCED, ARE STORED IN C.
C
C FIND FIRST ACTUAL RECORD AND ROW/COLUMN.
      JBF=0
  200 JBF=JBF+1
      IF(NBL(JBF+1).LT.IFC) GO TO 200
      IL = NBL(JBF)
      JF = IFC-IL
      FIND(8'JBF)
      KF = IFR-IL
      KF0=KF

READ RECORD JBL FROM FILE. THE ARRAY C WILL CONTAIN AT LEAST ONE UNRE-
DUCED COLUMN.
      DO 280 JBL=JBF,MAXBL
      READ(8'JBL)C,NCAT,ISZE
```

```
      FIND(8'JBF)
      NC=NCAT(100)
      JO=NBL(JBL)
      KL = IL
C
      DO 270 KBL = JBF,JBL
      LREC=KBL.EQ.JBL
      READ(8'KBL)CR,NRCAT,IRSZE
      NR=NRCAT(100)
      KO=KL
      KL=KL+NR
      MR=MAXO(KO,IIFR)
C
C IN ORDER TO MINIMIZE THE NUMBER OF TRANSPORTS TO AND FROM FILE 8, WE
C DO NOT (GENERALLY) COMPUTE ALL THE REDUCED ELEMENTS OF ONE COLUMN, BUT
C ONLY THE ELEMENTS IN ROW KO+1 TO KL. (KO IS THE NUMBER OF THE LAST
C COLUMN IN THE PREVIOUS RECORD, KL THE NUMBER OF THE LAST IN THE ACTUAL
C RECORD.)
C
      DO 260 J=JF,NC
      ISZ=ISZE(J)
C THE SUBSCRIPT OF THE ELEMENT IN COLUMN J,ROW K IS NCAT(J)+K-ISZE(J).
C THE DIFFERENCE NCAT(J)-ISZE(J) IS STORED IN THE VARIABLE ICO.
C THE SAME DIFFERENCE FOR COLUMN (ROW) K IS STORED IN IRO. THE ELEMENT
C I=ICO+MR, WHERE MR=MAX(KO,IIFR).
      ICO = NCAT(J)-ISZ
      I=ICO+MR
      JD=JO+J
      NRO=MINO(JD,KL)-KO
C
      DO 250 K = KF, NRO
      QSUM = 0.0Q0
      IRSZ=IRSZE(K)
      IRO = NRCAT(K)-IRSZ
      I=I+1
      QCI=C(I)
      KD = KO+K
      K1 = KD-1
      KF1=MAXO(IIFR,ISZ,IRSZ)+1
C IRSZ = KD INDICATES THAT COLUMN KD CONTAINS A NUMERICAL SINGULARITY.
C THE ELEMENTS OF ROW KD IS PUT EQUAL TO ZERO.
      IF (IRSZ .EQ. KD) QCI = QSUM

      IF(KF1.GT.K1) GO TO 245
      IF (LREC) GO TO 235
      DO 230 M = KF1, K1
      QCRM=CR(IRO+M)
      QCM=C(ICO+M)
230   QSUM = QSUM+QCM*QCRM
      GO TO 245
235   DO 240 M = KF1, K1
      QCRM=C(IRO+M)
      QCM = C(ICO+M)
240   QSUM = QSUM+QCM*QCRM
245   QCI = QCI-QSUM

      IF(LREC) GO TO 246
      C(I) = QCI/CR(IRO+KD)
      GO TO 250
```

```
  246 IF (JD.NE.KD) C(I) = QCI/C(IR0+KD)
  250 CONTINUE
C
      KF=1
      IF(.NOT.LREC.OR.(JD.NE.KD).OR.JD.EQ.N) GO TO 260
C
C  TEST OF NUMERICAL STABILITY
      QCI2 = C(I)**2
      IF (QCI/QCI2 .GT. 0.1Q-16) GO TO 251
      WRITE(6,20)JD
   20 FORMAT('NUMERICAL SINGULARITY IN ROW NO.',I4)
      ISZE(J)=JD
C THE COLUMN IS DELETED.
      GO TO 260
C
  251 C(I) = QSQRT(QCI)
  260 CONTINUE
  270 CONTINUE
C
C REDUCED ARRAY C BACK TO FILE. FIRST COLUMN IN NEXT RECORD IS NOW THE
C FIRST STORED, BUT FIRST REDUCED COLUMN IS AGAIN COLUMN KF0 IN REC. JBF
      WRITE(8'JBL)C,NCAT,ISZE
      JF=1
      KF=KF0
C
      IF(JBL.NE.MAXBL) GO TO 280
      PW = QCI
      IF(.NOT.LBS) GO TO 280
C
C BACK-SOLUTION. NOTE THAT THE VARIABLE I AT THIS MOMENT IS THE SUB-
C SCRIPT OF THE DIAGONAL ELEMENT OF THE COLUMN CONTAINING THE RIGHT-HAND
C SIDE.  I WILL SUCCESSIVELY TAKE THE VALUE OF THE SUBSCRIPT OF THE ELE-
C MENT IN ROW M OF THIS COLUMN, AND THE SOLUTION WILL BE STORED IN C(I).
      M = N
      KBL=MAXBL
C
      DO 277 KB= 1,MAXBL
      READ(8'KBL)CR,NRCAT,IRSZE
      KBL=KBL-1
      NR=NRCAT(100)
C THE COLUMN CONTAINING THE RIGHT-HAND SIDE IS SKIPPED.
      IF(KB.EQ.1)NR=NR-1
      K1=NR+1
      IF(NR.EQ.0) GO TO 277
      DO 276 K=1,NR
C WE STEP BACKWARD (M) FROM COLUMN N-1, TAKING THE NR COLUMNS FROM EACH
C RECORD SUCCESSIVELY.
      I=I-1
      IC=I
      M=M-1
      IR=NRCAT(K1)
      K1=K1-1
      IRSZ=IRSZE(K1)
      MR=M-IRSZ
      IF (MR.GT.0) GO TO 273
C IN CASE A COLUMN HAS BEEN DELETED, THE UNKNOWN IS PUT EQUAL TO ZERO.
      C(I) = 0.0D0
      GO TO 276
C THE (UN)KNOWN C(I) IS COMPUTED (ONE EQUATION WITH ONE UNKNOWN).
  273 C(I) = C(I)/CR(IR)
```

```
C
      IF(MR.EQ.1) GO TO 276
      DO 274 MP=2,MR
C IR IS THE SUBSCRIPT OF THE ELEMENTS OF COLUMN M, RUNNING FROM THE
C DIAGONAL-1 UP TO THE SUBSCRIPT OF THE FIRST ELEMENT DIFFERENT FROM
C ZERO. IC IS THE SUBSCRIPT OF THE ELEMENTS ON THE RIGHT-HAND SIDE IN
C THE SAME ROW AS CR(IR).
      IR=IR-1
      IC=IC-1
C THE CONTRIBUTION FROM THE (UN)KNOWN C(I) IS SUBTRACTED FROM THE QUAN-
C TITIES ON THE RIGHT-HAND SIDE.
  274 C(IC)=C(IC)-C(I)*CR(IR)
C
  276 CONTINUE
  277 CONTINUE
C THE SOLUTIONS ARE NOW ALL STORED IN C FROM C(NCAT(N)+1) TO C(NCAT(N+1)
C -1). THEY ARE TRANSFERRED TO 'MAIN' THROUGH THE COMMON BLOCK NESOL.
  280 CONTINUE
      RETURN
      END
```