MR 63

Evaluation of determinants, Solution of systems of linear equations and Matrix inversion.

by

T.J. Dekker

June 1963

CONTENTS. pag.

0. Introduction. 1

1. Description of the method for the general case. 2

2. Numerical considerations. 5

3. Test matrices. 8

4. The method for symmetric positive definite matrices. 11

5. Representation of triangular and symmetric matrices in ALGOL 60. 12

6. The parameters defining triangular and symmetric matrices. 14

7. Some features of the ALGOL 60 system for the Electrologica X1 Computer. 15

8. Calculation of sums and inner products. 17

The ALGOL 60 procedures: 19

Indentifier	Number in AP-series	
SUM	--	19
INPROD	--	21
DET	204	22
SOL	205	25
INV	206	26
DETSOL	207	29
DETINV	208	29
SYMDET1	224	30
SYMSOL1	225	33
SYMINV1	226	35
syminv1	227	37
SYMDET2	228	38
SYMSOL2	229	41

Literature 42

0. INTRODUCTION.

The purpose of this paper is to give a coherent set of ALGOL 60 procedures for the evaluation of determinants, the solution of systems of linear equations and the inversion of matrices. The method used is that of triangular decomposition with row interchanges for the general case, and the square root method for symmetric positive definite matrices. The procedures have been tested by means of the ALGOL 60 system for the Electrologica X1 Computer, written by E.W. Dijkstra and J.A. Zonneveld of the Mathematical Centre.

Chapter 3 deals with a set of asymmetric testmatrices, derived from segments of the Hilbert matrix. The matrices and their inverses are integral matrices and seem to be very useful for testing matrix inversion programs.

In Chapter 7 (and 8) some features of the ALGOL 60 system for the Electrologica X1 Computer are mentioned in order to give some idea about precision and organisation. A knowledge of these features is not necessary for an understanding of the text of the procedures, except SUM and INPROD.

Acknowledgements.

I express my gratutude to the Staff Members of the Computation Department of the Mathematical Centre for their valuable suggestions. Especially I am indebted to J.A. Zonneveld for reading the manuscript and suggesting various improvements and to Dr. P. Wynn for his help to bring the text into accord with the English idiom.

1. DESCRIPTION OF THE METHOD FOR THE GENERAL CASE.

The method used for general square non-singular matrices is that of triangular decomposition (Crout's method) with row interchanges (litt. [3], [9], [17]).

1. Triangular decomposition. A triangular decomposition of a square matrix is a decomposition of the matrix into a product L × U, where L is a lower and U an upper triangular matrix. In Crout's method the arrengement is such that U is a unit upper triangle, i.e. the diagonal elements of U are equal to 1.

2. Row interchanges. Triangular decomposition with row interchanges (so called "partial" pivoting for size) means that the rows of the given matrix are suitably interchanged (viz. in such a way that a reasonable accuracy is attained) and the resulting matrix is triangularly decomposed. Let A be the given matrix of order n (say), and let P denote the permutation matrix defining the row interchanges, so that P × A is the matrix with interchanged rows. Then we have the relation P × A = L × U and the problem is for a given A to find a suitable P and to calculate the triangular matrices L and U.

The columns of L and the rows of U are calculated successively in n steps. Each step is concerned with the calculation of one column of L and the corresponding row of U. After the calculation of a column of L, an element of this column satisfying some maximality condition (see 3 below) is selected as pivot. The pivotal and the diagonal element of the column considered are interchanged together with the corresponding rows of A and of that part of L, which has

already been computed.(Thus, in fact, at each stage the elements of the columns of L are known but for their order of succession, which depends on the subsequent row interchanges.) Thereafter the next row of U is calculated.

At each stage the calculated elements of L and U are written over the corresponding elements of the given matrix and the index of the pivotal row is recorded in an auxiliary n-vector. This auxiliary vector defines the permutation matrix P uniquely.

3. The maximality condition. As to the maximality condition for the pivot selection, at each stage we select an element of the column considered, with the property that its modulus divided by some norm of the corresponding row is maximal. The row norms used are the Euclidean norms of the rows of the original matrix, which need be calculated in advance only once.

4. Determinant evaluation. It is convenient to invert, together with each row interchange, the sign of one of the rows, in order to leave the determinant unaltered in value. Then the determinant of A will obviously be equal to the product of the diagonal elements of L.

5. Solution of linear systems of equations. After completing the process described above, we can obtain, for any given right-hand side b, a solution vector x of the linear system $A \times x = b$ in two steps, viz. the forward substitution, i.e. the calculation of the solution y of the linear system $L \times y = P \times b$, and the back substitution solving the linear system $U \times x = y$.

6. Matrix inversion. The inversion of the product matrix $L \times U$ may proceed as follows. The left inverse Q of L is calculated by solving the matrix equation $Q \times L = I$. The matrix Q is again a lower triangular matrix. The inverse matrix of $L \times U$ is obtained by solving the matrix equation $U \times X = Q$.

The rows of Q and X are calculated successively in reverse order in n steps. Each step is concerned with the calculation of one row of Q and the corresponding row of X. At each stage the new row of X can be written over the corresponding row of the given matrix. For this process only one extra n-vector is needed for temporary storage.

7. Column interchanges. Since $L \times U = P \times A$, the inverse of the original matrix A equals $X \times P$. This means that the interchanges, carried out on the rows of A, must be carried out correspondingly in reverse order on the columns of X, in order to obtain the inverse of A.

2. NUMERICAL CONSIDERATIONS.

1. An advantage of triangular decomposition (when compared with Gaussian elimination) is that triangular decomposition gives more accurate results than Gaussian elimination with the same pivoting strategy, provided however the inner products are calculated in extra precision.

2. In order to obtain a reasonably accurate method, some process of pivoting for size is needed, also in floating point arithmetic (cf. [9], p. 50). The partial pivoting sketched above involves interchanges of rows only. The administration of these row interchanges is rather simple and requires a negligeable amount of extra computation. A drawback is that (in exceptional cases) partial pivoting may yield useless results, even on a well-conditioned matrix (cf. [17], p. 327). In order to avoid this, one may use Gaussian elimination with so called "complete" pivoting. This means, that at each stage the pivot is selected in the whole of the remaining square matrix. Complete pivoting involves interchanges of rows and columns and requires a non-negligeable amount of extra computation (which for large order is a nearly constant fraction of the total computation). Unfortunately complete pivoting cannot be car ied out in a triangular decomposition scheme. So the use of triangular decomposition with partial pivoting seems to be a good practical compromise.

3. Pivoting for size (and especially partial pivoting) is a reasonable strategy only if all rows and columns of the original matrix have comparable norms. (Such matrices are called "equilibrated", cf. [17],

p. 284.) Therefore, since in the process described above the pivot is selected from a certain column at each stage, it seems fair to select as pivot an element in this column, not of maximal modulus, but with the property that its modulus devided by some norm of the corresponding row is maximal. If one uses the (e.g. Euclidean) norms, computed beforehand, of the rows of the original matrix, the application of this maximality condition requires only a slight extra effort.

4. No provision is made for detecting singularity. In a system of floating point operations, division by zero should be allowed (i.e. not lead to a stop). In this way we have no need to test whether a pivot vanishes. If some very small pivot occurs, it may be desirable to take special measures. The programmer can do this after completing the triangular decomposition; then the pivots are avaible as the diagonal elements of L.

5. For the calculation of the inverse of $L \times U$ there are various possibilities (cf. [3], p. 29 - 41). One way is to calculate both L and U and then to form the product $U \times L$. Another way is to calculate a left inverse or a right inverse of L and then to compute X by solving the matrix equation $U \times X = L$. In the process described above the scheme entailing the calculation of the left inverse Q of L has been chosen, because it admits an arrangement of the computation in n steps in such a way that in each step a row of Q and a row of X are obtained.

It is also possible to obtain the inverse X without inverting L or U. Indeed the conditions "$X \times L$ is a unit upper triangle" and "$U \times X$ is a lower triangle" together just suffice to determine X.

If inner products are calculated in extra precision, this scheme has the advantage that less intermediate results are rounded to working accuracy. Since, moreover, this scheme also requires only one extra n-vector for temporary storage and gives rise to a rather simple program, it seems to be the most satisfactory scheme for matrix inversion. The reason why I did not incorporate this scheme in the ALGOL procedure printed below, is that I saw this possibility only after writing and testing the ALGOL procedures.

3. TEST MATRICES.

The procedures AP 204-208, which are now used regularly, have been tested on several matrices with satisfactory results. I discuss here only the following test matrices which are derived from segments of the Hilbert matrix.

Consider the n-th order segment H of the Hilbert matrix with the elements

$$H_{ij} = 1/(i + j - 1) \qquad (i, j = 1,\ldots, n).$$

The inverse matrix H^{-1} has the elements (cf. [12]):

$$H^{-1}_{ij} = (-1)^{i+j} f_i f_j /(i + j - 1),$$

where $f_i = (n + i - 1)\,! /(((i - 1)\,!)^2 (n - i)\,!)$.

Let $F = \mathrm{diag}(f_i)$ and $E = \mathrm{diag}((-1)^i)$.

Then we have obviously

$$H^{-1} = F E H E F ,$$

where E H E is the "chess-board" matrix corresponding to H, i.e. the matrix whose elements have the same modulus but are alternating in sign.

Let f_i, separated into prime factors, have the form

$$f_i = p_1^{m_1} \cdot \ldots \cdot p_k^{m_k}$$

and let $g_i = p_1^{m_1 : 2} \cdot \ldots \cdot p_k^{m_k : 2}$,

where : denotes integer division, as defined in ALGOL 60, thus (for non-negative m) $m : 2 = \mathrm{entier}(m/2)$ (cf. [1] section 3. 3. 4. 2.).

Furthermore let $G = \mathrm{diag}(g_i)$ and $M = F G^{-1} H G$.

Then we have $M^{-1} = G^{-1} F E H E G = E M E$,

in other words: M^{-1} equals the chess-board matrix corresponding to M.

Moreover we have the following

THEOREM The elements of matrix M are integers.

Proof Let $y = i + j - 1$. Then $f_i \; f_j$ contains a factor y^2.

Indeed $f_i \; f_j =$

$$\frac{(n+i-1)!}{(n-j)!y!} \cdot \frac{(n+j-1)!}{(n-i)!y!} \left(\frac{y!}{(i-1)!\;(j-1)!} \right)^2 = \binom{n+i-1}{y} \binom{n+j-1}{y} \binom{y-1}{i-1}^2 y^2 .$$

Consider now an element $M_{ij} = f_i \, g_i^{-1} \, g_j \, y^{-1}$ and any prime number p. Let f_i, f_j and y in this order contain exactly q_1, q_2 and q prime factors p.

Then $q_1 + q_2 \geq 2q$, since $f_i \, f_j$ contains a factor y^2. The number N of factors p in $f_i \; g_i^{-1} \; g_j$ satisfies

$$N = q_1 - q_1 \div 2 + q_2 \div 2 \quad \geq (q_1 + q_2) \div 2 \quad \geq q \quad .$$

In other words $f_i \, g_i^{-1} \, g_j$ contains a factor p^q.

Hence $f_i \, g_i^{-1} \, g_j$ contains a factor y, which proves the theorem.

We have, then, a set of integer matrices M (for n = 1, 2,...) whose inverses are again integer matrices. For large n these matrices are very ill-conditioned, as are the segments of the Hilbert matrix. Thus these matrices are very useful test matrices for matrix inversion programs. If the integers have an exact floating point representation, the errors in the calculated inverses are due solely to the matrix inversion program. This is important, as inaxact input of an ill-conditioned matrix may yield a matrix whose inverse differs considerably from the inverse of the original matrix (cf. [15] and [17], p. 319 sqq).

The inverses of M of order up to 7 have been calculated using DET and INV printed below. The machine representation of floating point

numbers has a relative precision of 40 binary digits, but the inner products are calculated to 52 binary digits. (For further details see Chapter 7 and 8.) Moreover the calculations have been carried out with two modified versions, viz. a version in which the pivoting is suppressed and a version in which the inner products are calculated without extra precision for the partial sums. The maximal absolute errors are tabulated below. The calculation of inner products without extra precision gives indeed a slightly worse result in most cases. The suppression of the pivoting gives better results, which is not surprising in view of the close relation of the matrices M to the positive definite segments of the Hilbert matrix. If the matrix is positive definite, and apparently also in this case, the pivots are preferably chosen on the main diagonal.

Maximal absolute error in the calculated inverse of $M = F\ G^{-1} H\ G$

Order of M	Standard DET and INV	pivoting suppressed	INPROD without extra precision
4	$5.1_{10} - 9$	$6.5_{10} - 9$	$1.5_{10} - 8$
5	$1.2_{10} - 7$	$1.6_{10} - 7$	$1.7_{10} - 6$
6	$2.9_{10} - 4$	$4.9_{10} - 6$	$9.7_{10} - 5$
7	$3.7_{10} - 2$	$9.8_{10} - 5$	$1.1_{10} - 1$

4. THE METHOD FOR SYMMETRIC POSITIVE DEFINITE MATRICES.

The method used is the square root method of Cholesky (litt. [3], [9], [17]).

In this case no pivoting for size is needed. (cf. [17], p. 305 sqq.)

The given matrix A is decomposed into a product $U^T \times U$, where U is upper triangular.

The solution of the linear system $A \times x = b$ proceeds in two steps, viz. the forward substitution, i.e. the calculation of the solution y of the linear system $U^T \times y = b$, and the backsubstitution solving the system $U \times x = y$.

The inversion of $U^T \times U$ proceeds as follows: The left inverse Q of U is calculated by solving the matrix equation $Q \times U = I$ and the inverse X of A is calculated by the matrix multiplication $Q \times Q^T = X$.

No provision is made for detecting singularity or non-definiteness. The matrix may be given in triangular form. Furthermore for all these processes only one extra n-vector is needed for temporary storage.

5. REPRESENTATION OF TRIANGULAR AND SYMMETRIC MATRICES IN ALGOL 60.

In ALGOL 60 a matrix is most obviously represented by a two-dimensional array. If the matrix is triangular or symmetric, this representation has the disavantage that almost half of the space is wasted. This can be avoided by putting the elements of an n-th order triangle in a one-dimensional array of lenght $(n + 1) \times n : 2$. Consider a triangle A of order n with elements a_{ij}, where $1 \leq i \leq j \leq n$. Then a convenient arrangement of these elements in a one-dimensional array is the following:

$a_{11}, a_{12}, a_{22}, a_{13}, a_{23}, a_{33}, \ldots, a_{1n}, a_{2n}, \ldots, a_{nn}$.

In other words: the elements a_{ij} are placed in a one-dimensional array C [1 : (n + 1) × n : 2] according to the formula:

$a_{ij} = C[(j - 1) \times j : 2 + i]$.

In order to avoid repeated evaluation of the index-expression each time an element a_{ij} is needed, it is advisable to introduce an auxiliary integer array J[1 : n] and to carry out once for all the statement "for j: = 1 step 1 until n do J[j]: = (j - 1) × j : 2".

Then we have the rather surveyable formula:

(1) $a_{ij} = C[i + J[j]]$.

In this way we can refer, in an ALGOL program, to an element of a triangle without appreciable loss of time.

Formula (1) is independent of the range of the indices. Indeed, let h be the lower, and k = h + n - 1 the upper bound of the indices (thus $h \leq i \leq j \leq k$). If in this case we introduce an integer array J[h : k] and carry out the prepatory assignments:

"for j: = h step 1 until k do J[j]: = (j-h) × (j-h+1) : 2-h+1", then formula (1) remains valid. So we may consider formula (1) as a suitable standard representation of triangular and symmetric matrices. Using this representation one must of course take care that in each reference to an element of C the relation $i \leq j$ holds.

6. THE PARAMETERS DEFINING TRIANGULAR AND SYMMETRIC MATRICES.

In order to obtain flexible procedures for dealing with triangular or symmetric matrices, these matrices may be defined in the following way by means of 4 actual parameters, corresponding to the formal parameters A, i, j, n (say). The parameters i, j and n are specified integer and A is specified real (or integer). The value parameter n is the order of the triangle, i denotes the smallest index, j the largest index and A the (i, j)-th element of the triangle. In other words:

the actual parameter for A must be a subscripted variable, whose index (or indices) depend(s) on the actual parameters for i and j in such a way, that for each i and j satisfying $1 \leq i \leq j \leq n$ the actual parameter for A is the (i, j)-th element of the triangle.

Using a procedure in which a triangle is defined in this way, one can freely choose dimension and range of the array representing the triangle.

7. SOME FEATURES OF THE ALGOL 60 SYSTEM FOR THE ELECTROLOGICA X1 COMPUTER.

In the ALGOL 60 system for the Electrologica X1 Computer the real arithmetic has a relative precision of 52 binary digits (i.e. about 15 decimals), but the assignment operation rounds off to 40 binary digits (i.e. 12 decimals). This rounding-off takes place at each assignment to a real (simple or subscripted) variable and at the evaluation of value parameters of type real, but not at the assignment to a procedure identifier.

Division by zero is allowed and yields a result whose modulus is large with respect to the numerator.

The absolute value of integer variables must remain less then 2 ↑ 26. As soon as an anonymous intermediate result which according to ALGOL 60 should be of type integer, exceeds the integer capacity, automatically transition to the real representation takes place.

The primaries in an expression are evaluated from left to right. The value parameters of a procedure are evaluated from left to right in order of specification.

The specifications real and integer of non-value scalars and arrays are equivalent. The actual type depends only on the type of the corresponding actual parameter. Likewise the declarators integer procedure and real procedure are equivalent. As each call the type of the value of a function designator is determined only by the arithmetic actually executed in the body.

The type of "abs (E)" is the same as the type of the expression E.

The functions "sqrt" and "ln" operate on the modulus of the argument. In the ALGOL 60 system for the Electrologica X1 Computer some procedures and type procedures, written in machine code, are avaible without declaration. This set contains the standard functions mentioned in the ALGOL 60 report [1] section 3.2.4. and 3.2.5, some fundamental operations, e.g. SUM and INPROD, and procedures for input and output.

8. CALCULATIONS OF SUMS AND INNER PRODUCTS.

It is important to calculate sums, and especially inner products, in extra precision, in order to avoid accumulation of rounding errors. By so doing one obtains appreciably smaller error bounds for many matrix operations (see various papers of J.H. Wilkinson, e.g. [17], p. 329).

In the ALGOL 60 system for the Electrologica X1 Computer the machine code procedures SUM and INPROD, calculating sums and inner products respectively, are available without declaration. These procedures do not assign the partial sums to a local variable, but build up their results in the full precision of the arithmetic. So, in fact, sums and inner products are calculated with 12 guarding binary digits. The definitions given below of SUM and INPROD are nearly equivalent to the corresponding machine code procedures. The definitions are in recursive form in order to indicate that no local variable is used for the partial sums and that the results are calculated in the full precision of the arithmetic. Note that in these definitions it is assumed (for the sake of presentation only) that the primaries in an expression are evaluated from left to right.

Of course inner products may be calculated by means of SUM. The machine code procedure INPROD has been written especially for the case that the actual parameters for xk and yk are subscripted variables (of type real or integer), whose indices are linear functions of the controlled variable. Under this restriction the machine code procedure INPROD is nearly equivalent to the declaration given below.

Because of this restriction INPROD cannot always be used for the calculation of inner products of rows or columns of triangles, since the indices of the array elements are not always linear functions of the controlled variable. Therefore in some procedures operating on triangles, SUM is used for the calculation of inner products.

The machine code procedure INPROD carries out the summation in reverse order. Though the numerical result depends on the order of summation, in general it is not important, which order of summation is chosen.

comment SUM: = the sum over k from a until b of tk. The actual parameter for k is the summation variable and the actual parameter for tk is an expression depending on the summation variable. Note that after a call of SUM the summation variable has the (so called rejected) value: if a ≤ b then b + 1 else a;

```
real procedure SUM (k, a, b, tk);
value b, a; integer b, a, k; real tk;
begin k: = a;
      SUM: = if a > b then 0 else tk + SUM (k, a+1, b, tk)
end   SUM;
```

comment INPROD: = the sum over k decreasing from b until a of xk × yk. In other words, the summation is carried out in reverse order.
The actual parameter for k is the summation variable and the actual parameters for xk and yk are expressions depending on the summation variable. For matrix work it is of great value to have a real procedure INPROD in machine code which calculates the inner product in extra precision. In order to obtain a rather fast and yet sufficiently useful process, this machine code INPROD may be written for the special case in which the actual parameters for xk and yk are subscripted variables with indices linearly dependent on the summation variable;

```
real procedure INPROD (k, a, b, xk, yk);
value a, b; integer k, a, b; real xk, yk;
begin  k: = b;
       INPROD:= if a > b then 0 else xk × yk + INPROD (k,a,b-1,xk,yk);
       k: = b + 1
end    INPROD;
```

comment AP 204

DET: = determinant of the n-th order matrix given in array A[1 : n, 1 : n]. The first index of the array elements is the row index, the second one the column index. The method is triangular decomposition according to Crout with row interchanges. This process yields a lower triangle L and a unit upper triangle U such that L × U equals matrix A with interchanged rows. Together with each row interchange the sign of the non-pivotal row is reversed, so that the determinant equals the product of the diagonal elements of L. At each stage the pivot is chosen in a column of L such that its modulus divided by the Euclidean norm of the corresponding row of A is maximal. The integer array p[1 : n] is an output vector in which the pivotal row indices are recorded. The elements of A are replaced by the corresponding calculated elements of L and U. So enough information is retained for subsequent solution of linear systems and for matrix inversion. DET uses the non-local real procedure INPROD;

```
real procedure DET (A,n,p);
value n; integer n; array A; integer array p;
begin   integer i,j,k; real d,r,s; array v[1: n];
        for i:= 1 step 1 until n do
        v[i]:= sqrt (INPROD (j,1,n,A[i,j],A[i,j]));
        d:= 1;
        for k:= 1 step 1 until n do
        begin   r:= - 1;
                for i:= k step 1 until n do
                begin   A[i,k]:=
                        A[i,k] - INPROD (j,1,k - 1,A[i,j],A[j,k]);
                        s:= abs (A[i,k]) / v[i];
                        if s > r then begin r:= s; p[k]:= i end
                end LOWER;
                v[p[k]]:= v[k];
                for j:= 1 step 1 until n do
                begin   r:= A[k,j];
                        A[k,j]:= if j ≤ k then A[p[k],j] else
                        (A[p[k],j] - INPROD (i,1,k - 1,A[k,i],A[i,j]))
                        / A[k,k]; if p[k] ≠ k then A[p[k],j]:= - r
                end UPPER;
                d:= A[k,k] × d
        end LU;
        DET:= d
end     DET;
```

comment AP 205

SOL replaces the vector given in array b[1 : n], by the solution vector x of the linear system L × U × x = b with interchanged (and possibly sign-reversed) elements. Here L is the lower triangle and U the unit upper triangle which are given in array LU[1:n, 1:n] such that the elements with first index < second index are elements of U and the other elements of LU are elements of L. The integer array p[1 : n] defines the interchanges with sign reversions of the elements of b in correspondence with the pivoting administration in DET. Hence a call DET(A,n,p), followed by a call SOL(A,b,n,p), has the effect that b is replaced by the solution vector x of the linear system: Sigma (A[i,j] × x[j]) = b[i]. The procedure SOL leaves the elements of LU and p unaltered and uses the non-local real procedure INPROD;

```
procedure SOL (LU,b,n,p);
value n; integer n; array LU,b; integer array p;
begin   integer i,k; real r;
        for k:= 1 step 1 until n do
        begin   r:= b[k]; b[k]:=
                (b[p[k]] - INPROD (i,1,k - 1,LU[k,i],b[i])) / LU[k,k];
                if p[k] ≠ k then b[p[k]]:= - r
        end;
        for k:= n step - 1 until 1 do
        b[k]:= b[k] - INPROD (i,k + 1,n,LU[k,i],b[i])
end     SOL;
```

comment AP 206

INV replaces the elements of array LU[1 : n, 1 : n] by the corresponding elements of the inverse of L × U, where L is the lower triangle and U the unit upper triangle which are given in array LU such that the elements with first index < second index are elements of U and the other elements of LU are elements of L. The inverse is calculated by forming the left inverse of L and solving the resulting matrix equation. Subsequently the interchanges with sign reversions, defined by the integer array p[1 : n] in correspondence with the pivoting administration in DET, are carried out in reverse order on the columns of LU. Hence a call DET(A,n,p), followed by a call INV(A,n,p), has the effect that matrix A is replaced by its inverse. INV uses the non-local real procedure INPROD;

```
procedure INV (LU,n,p); value n; integer n; array LU; integer array p;
begin   integer i,j,k; real r; array v[1: n];
        for k:= n step - 1 until 1 do
        begin   for j:= k + 1 step 1 until n do
                begin   v[j]:= LU[k,j]; LU[k,j]:= 0 end;
                LU[k,k]:= 1 / LU[k,k];
                for j:= k - 1 step - 1 until 1 do
                LU[k,j]:= - INPROD(i,j + 1,k,LU[k,i],LU[i,j])/LU[j,j];
                for j:= 1 step 1 until n do
                LU[k,j]:= LU[k,j] - INPROD (i,k + 1,n,v[i],LU[i,j])
        end;
        for k:= n step - 1 until 1 do
        begin   if p[k] ≠ k then for i:= 1 step 1 until n do
                begin   r:= LU[i,k]; LU[i,k]:= - LU[i,p[k]];
                        LU[i,p[k]]:= r
                end
        end
end     INV;
```

```
comment  AP 207
```

DETSOL: = determinant of the n-th order matrix given in array A[1 : n, 1: n]. Moreover the vector, given in array b[1 : n], is replaced by the solution vector x of the linear system: Sigma (A[i,j] × x[j]) = b[i]. For further details see DET and SOL, which are used by DETSOL;

```
real procedure DETSOL (A,b,n); value n; integer n; array A,b;
begin   integer array p[1:n];
        DETSOL:= DET (A,n,p); SOL (A,b,n,p)
end     DETSOL;
```

```
comment  AP 208
```

DETINV: = determinant of the n-th order matrix given in array A[1 : n, 1 : n]. Moreover this matrix is replaced by its inverse. For further details see DET and INV, which are used by DETINV;

```
real procedure DETINV (A,n); value n; integer n; array A;
begin   integer array p[1:n];
        DETINV:= DET (A,n,p); INV (A,n,p)
end     DETINV;
```

comment AP 224

SYMDET1: = determinant of the n-th order symmetric positive definite matrix M which is defined as follows: the actual parameter for A - being a subscripted real variable whose indices (or index) depend(s) on the actual parameters for i and j - is the (i,j)-th element of M for each i and j satisfying $1 \leq i \leq j \leq n$. Thus one needs to give only the upper triangle of M. In order to avoid waste of space, one may give this triangle in a one-demensional array.

E.g. if the upper triangle of M is given in array C[1: n × (n+1): 2] columnwise, i.e. the columns one after the other, and the successive values (j-1) × j : 2 have been recorded in an auxiliary integer array J[1 : n], then the appropriate call of SYMDET1 reads:

SYMDET1 (C[i+J[j]],i,j,n).

The method used is the square root method of Cholesky, yielding an upper triangle which premultiplied by its transpose gives the original matrix. SYMDET1 replaces the elements of M by the corresponding elements of this upper triangle and uses the non-local real procedure SUM;

```
real procedure SYMDET1 (A,i,j,n); value n; integer i,j,n; real A;
begin   integer k; real d,r; array v[1:n];
        d:= 1;
        for k:= 1 step 1 until n do
        begin   j:= k; for i:= 1 step 1 until k do v[i]:= A;
                i:= k; A:= r:= sqrt (v[k] - SUM (i,1,k-1,v[i] ↑ 2));
                d:= r × d;
                for j:= k+1 step 1 until n do
                begin i:= k; A:= (A - SUM (i,1,k-1,A × v[i])) / r end
        end     LU;
        SYMDET1 := d ↑ 2
end     SYMDET1 ;
```

comment AP 226

SYMINV1 replaces the matrix elements A by the corresponding upper triangular elements of the inverse of U transpose × U, where U is an upper triangle which is defined by the actual parameters in the same way as the upper triangle of M in SYMDET1.
Consequently a call of SUMDET1, followed by a call of SYMINV1 with the same actual parameters, has the effect that the upper triangle of the symmetric positive definite matrix M is replaced by the upper triangle of the inverse of M. The procedure SYMINV1 uses the non-local real procedure SUM;

```
procedure SYMINV1 (A,i,j,n); value n; integer i,j,n; real A;
begin   integer k; real r; array v[1:n];
        for k:= 1 step 1 until n do
        begin   i:= j:= k; A:= v[k]:= 1 / A;
                for j:= k+1 step 1 until n do
                begin   i:= j; r:= A; i:= k;
                        A:= v[j]:= - SUM (i,k,j-1,A × v[i]) / r
                end;
                for i:= 1 step 1 until k do
                begin   j:= k; A:= SUM (j,k,n,A × v[j]) end
        end
end     SYMINV1 ;
```

comment AP 225

SYMSOL1 replaces the vector given in array b[1 : n], by the solution vector x of the linear system: U transpose × U × x = b, where U is an upper triangle which is defined by the actual parameters for A, i, j and n in the same way as the upper triangle of M in SYMDET1. Consequently a call of SYMDET1, followed by a call of SYMSOL1 with the same actual parameters for A, i, j and n, has the effect that b is replaced by the solution vector x of the linear system M × x = b. The procedure SYMSOL1 leaves the elements A unaltered and uses the non-local real procedure SUM;

```
procedure SYMSOL1(A,i,j,n,b); value n; integer i,j,n; real A; array b;
begin   real r;
        for j:= 1 step 1 until n do
        begin   i:= j; r:= A;
                b[j]:= (b[j] - SUM (i,1,j-1,A × b[i])) / r
        end;
        for i:= n step -1 until 1 do
        begin   j:= i; r:= A;
                b[i]:= (b[i] - SUM (j,i+1,n,A × b[j])) / r
        end
end     SYMSOL1 ;
```

comment AP 227

symminv1 calculates the main diagonal of the inverse of U transpose × U, where U is an upper triangle which is defined by the actual parameters for A, i, j and n in the same way as the upper triangle of M in SYMDET1. The calculated diagonal elements are delivered in array d[1 : n].

Consequently a call of SYMDET1, followed by a call of symminv1 with the same actual parameters for A, i, j and n, has the effect that the diaginal elements of the inverse of the symmetric positive definite matrix M are delivered in array d. The procedure symminv1 leaves the elements A unaltered and uses the non-local real procedure SUM;

```
procedure symminv1(A,i,j,n,d); value n; integer i,j,n; real A; array d;
begin   integer k; real r;
        for k:= 1 step 1 until n do
        begin   i:= j:= k; d[k]:= 1 / A;
                for j:= k+1 step 1 until n do
                begin   i:= j; r:= A;
                        d[j]:= - SUM (i,k,j-1,A × d[i]) / r
                end;
                d[k]:= SUM (j,k,n,d[j] ↑ 2)
        end
end     symminv1 ;
```

comment AP 228

SYMDET2 : = determinant of the n-th order symmetric positive definite matrix, given in integer array A[1 : n × (n+1) : 2] in such a way that, for all i and j satisfying $1 \leq i \leq j \leq n$, the (i,j)-th element is A[i+ (j-1) × j : 2]. The method used is the square root method of Cholesky, yielding an upper triangle U which premultiplied by its transpose gives alfa × matrix A.

The elements of U are written over the corresponding elements of A. The scaling factor alfa must be chosen such that the maximal element of U is just within the integer capacity, in order to obtain a reasonably accurate representation of U. In view of the definiteness of A this means that alfa must be slightly less (but not too critical, on account of the inexactness of the arithmetic) than the square of the integer capacity divided by the maximal element of A. One may use SYMDET2 also with real array A, in which case the most obvious value of alfa is 1.0. If A is negative definite, one may use SYMDET2 with alfa negative. SYMDET2 uses the non-local real procedure INPROD;

```
real procedure SYMDET2 (A,n,alfa);
value n,alfa; integer n; real alfa; integer array A;
begin   integer i,j,k,kk,kj; real d;
        d:= 1; kk:= 0;
        for k:= 1 step 1 until n do
        begin   kk:= kk+k; A[kk]:=
                sqrt(A[kk] × alfa - INPROD(i,1-k,-1,A[kk+i],A[kk+i]));
                d:= A[kk] × d; kj:= kk;
                for j:= k+1 step 1 until n do
                begin   kj:= kj+j-1;
                        A[kj]:= (A[kj] × alfa
                        - INPROD (i,1-k,-1,A[kj+i],A[kk+i])) / A[kk]
                end
        end     LU;
        SYMDET2:= d ↑ 2 / alfa ↑ n
end     SYMDET2;
```

comment AP 229

SYMSOL2 replaces the vector given in real array b[1 : n], by the solution vector x of the linear system: U transpose × U × x = alfa × b, where U is an upper triangle, given in integer (or real) array A[1 : n × (n+1) : 2] in such a way that, for all i and j satisfying 1 ≤ i ≤ j ≤ n, the (i,j)-th element is A[i + (j-1) × j : 2]. The scaling factor alfa is chosen in relation to the scaling of U. Consequently a call SYMDET2 (A, n, alfa), followed by a call SYMSOL2 (A, n, alfa, b), has the effect that b is replaced by the solution vector x of the linear system A × x = b. The procedure SYMSOL2 leaves the elements of A unaltered and uses the non-local real procedure SUM;

```
procedure SYMSOL2 (A,n,alfa,b);
value n,alfa; integer n; real alfa; integer array A; real array b;
begin   integer i,j,j0; integer array J[1:n];
        j0:= 0;
        for j:= 1 step 1 until n do
        begin   b[j]:=
                (b[j] × alfa - SUM (i,1,j-1,A[i+j0] × b[i]))/A[j+j0];
                J[j]:= j0; j0:= j0+j
        end;
        for i:= n step -1 until 1 do
        b[i]:= (b[i] - SUM (j,i+1,n,A[i + J[j]] × b[j]))/A[i + J[i]]
end     SYMSOL2;
```

LITERATURE.

1. J.W. Backus, e.a.; Revised Report on the algorithmic language ALGOL 60, Copenhagen 1962, Comm. ACM 6 (1963) 1 - 17, Num. Math. 4 (1963) 420-453.

2. W.R. Busing and H.A. Levy; A procedure for inverting large symmetric matrices, Comm. ACM 5 (1962) 445-446.

3. L. Fox; Practical solution of linear equations and inversion of matrices, Nat. Bur. Stand. - App. Math. Ser. 39 (1954) 1-54.

4. L. Fox, H.D. Huskey and J.H. Wilkinson; Note on the solution of algebraic linear simultaneous equations, Quart. J. App. Math. 1 (1948) 149-173.

5. H.H. Goldstine and J. von Neumann; Numerical inverting of matrices of high order, Bull. Am. Math. Soc. 53 (1947) 1021-1099.

6. A.S. Householder; A survey of some closed methods for inverting matrices, J. Soc. Ind. App. Math. 5 (1957) 155-169.

7. M. Lotkin; A set of testmatrices, MTAC 9 (1955) 153-161.

8. M. Marcus; Basic theorems in matrix theory, Nat. Bur. Stand. - App. Math. Ser. 57 (1960).

9. Nat. Phys. Lab.; Modern computing methods, London 1961.

10. M. Newman and J. Todd; The evaluation of matrix inversion programs, J. Soc. Ind. App. Math. 6 (1958) 466-476.

11. H. Rutishauser; Zur Matrizeninversion nach Gauss-Jordan, ZAMP 10 (1959) 281-291.

12. I.R. Savage and E. Lukacs; Tables of inverses of finite segments of the Hilbert matrix, Nat. Bur. Stand. - App. Math. Ser. 39 (1954) 105-108.

13. J. Todd; Computional problems concerning the Hilbert matrix, J. Res. NBS-B 65 (1961) 19-22.

14. A.M. Turing; Rounding-off errors in matrix processes, Quart. J. Mech. App. Math. 1 (1948) 287-308.

15. J.H. Wilkinson; Rounding errors in algebraic processes, Proc. Int. Conf. Inf. Proc. Unesco, Paris 1959.

16. J.H. Wilkinson; Error analysis of floating point computation, Num. Math. 2 (1960) 319-340.

17. J.H. Wilkinson; Error analysis of direct methods of matrix inversion, J. Ass. Comp. Mach. 8 (1961) 281-330.

18. H.C. Thacher; Crout with pivoting II, Alg. 43 (Revision of Alg. 16 of G.E. Forsythe),ACM 4 (1961), p. 176.

19. W.M. McKeeman; Crout with equilibration and iteration, Alg. 135, ACM 5 (1962), p. 553.