STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM
REKENAFDELING

ALGOL 60 procedures in numerical algebra

by

T.J. Dekker

with assistance of H.N. Glorie

NR 8
februari 1969

PREFACE

In this document a system of procedures is given for solving a linear system with some greater accuracy than can be obtained by the system presented in MC tract 22 (T.J. Dekker, ALGOL 60 procedures in numerical algebra, part 1).
Essentially two devices contribute to the increase of accuracy.
(1) double length arithmetic and (2) iterative improvement of a first approximation.

All the routines have been tested for correct operation. Nevertheless this report has a preliminary status: The system still has to be augmented and, perhaps, the description still has to be polished. Moreover, experience concerning the accuracy obtained by and the execution time required for the calculation have to be added. The final form of the report will be published in the series "ALGOL 60 procedures in numerical algebra" as a MC-tract in the near future.

The reason for the distribution of a few copies presenting the main results of the research involved is the absence of the author.

KA and vdH.

CONTENTS

CHAPTER 25

VECTOROPERATIONS

This chapter contains procedures for double length arithmetic. A double precision number is represented as a pair of real numbers (head, tail), where "head" is a representable single precision number, nearest to the exact sum of head and tail. This "head-tail" condition ensures that the tail is indeed small with respect to the head. In particular, any pair (c,0) satisfies this condition.

The procedures of this chapter work correctly in binary floating arithmetic with 40-bit mantissa, provided only that all operations occurring in the body are optimal, i.e. yield a representable result nearest to the exact value.

The operations occurring in the body are addition, subtraction, multiplication by the special factor 2↑20+1 and multiplication of two short numbers, each representable with a 20-bit mantissa. Apart from overflow and underflow this optimality condition is satisfied in the arithmetic of the E1 X8, for which machine these procedures have been written primarily.

The basic devices used in the double precision calculations are the following:

1) the sum of two single precision numbers a and b can be represented by a double precision number (c, cc), which is obtained by performing the statements:
   " c:= a+b ; cc:= a-c+b" if $|a| \geq |b|$; otherwise the roles of a and b are interchanged,
2) a number a, is splitted into two short numbers by performing the statements:
   " p:= (2↑20+1)×a ; x:= a-p+p ; xx:= a-x".
   The value of a, correctly rounded to 20 bits, equals x, and the remaining part xx of a is representable with a mantissa of at most 19 bits.

Section 250 Scalarproducts in double precision

The procedures of this section calculate the sum of the double precision number (c, cc) and the scalarproduct of two single precision vectors. The result is calculated in double precision and delivered in the pair (d, dd). The head-tail condition mentioned above must hold for the given pair (c, cc) and does hold for the resulting pair (d, dd).
The method is as follows:
the double precision product of two single precision factors is obtained by splitting the factors into short numbers (see above device 2), (x, xx) and (y, yy). The product is performed in four statements:

```
" z:= x * yy + xx * y ;  y:= x * y ;  x:= y + z ;
  y:= y - x + z + xx * yy  ".
```

The double precision number (x, y) is added to (c, cc) in double precision.

```
comment mca 2500;
procedure lngvecvec(l, u, shift, a, b, c, cc, d, dd);
value l, u, shift, c, cc; integer l, u, shift; real c, cc, d, dd;
array a, b;
begin integer k;
    real x, xx, y, yy, z;
    for k:= l step 1 until u do
    begin xx:= a[k]; yy:= b[shift + k];
    mul12: x:= xx × 1048577; x:= xx - x + x; xx:= xx - x;
        y:= yy × 1048577; y:= yy - y + y; yy:= yy - y;
        z:= x × yy + xx × y; y:= x × y; x:= y + z;
        y:= y - x + z + xx × yy;
    add2: z:= x + c;
        cc:= (if abs(x) > abs(c) then x - z + c else c - z + x) + y
        + cc; c:= z + cc; cc:= z - c + cc
    end;
    d:= c; dd:= cc
end lngvecvec;

comment mca 2501;
procedure lngmatvec(l, u, i, a, b, c, cc, d, dd);
value l, u, i, c, cc; integer l, u, i; real c, cc, d, dd; array a, b;
begin integer k;
    real x, xx, y, yy, z;
    for k:= l step 1 until u do
    begin xx:= a[i,k]; yy:= b[k];
    mul12: x:= xx × 1048577; x:= xx - x + x; xx:= xx - x;
        y:= yy × 1048577; y:= yy - y + y; yy:= yy - y;
        z:= x × yy + xx × y; y:= x × y; x:= y + z;
        y:= y - x + z + xx × yy;
    add2: z:= x + c;
        cc:= (if abs(x) > abs(c) then x - z + c else c - z + x) + y
        + cc; c:= z + cc; cc:= z - c + cc
    end;
    d:= c; dd:= cc
end lngmatvec;

comment mca 2502;
procedure lngtamvec(l, u, i, a, b, c, cc, d, dd);
value l, u, i, c, cc; integer l, u, i; real c, cc, d, dd; array a, b;
begin integer k;
    real x, xx, y, yy, z;
    for k:= l step 1 until u do
    begin xx:= a[k,i]; yy:= b[k];
    mul12: x:= xx × 1048577; x:= xx - x + x; xx:= xx - x;
        y:= yy × 1048577; y:= yy - y + y; yy:= yy - y;
        z:= x × yy + xx × y; y:= x × y; x:= y + z;
        y:= y - x + z + xx × yy;
    add2: z:= x + c;
        cc:= (if abs(x) > abs(c) then x - z + c else c - z + x) + y
        + cc; c:= z + cc; cc:= z - c + cc
    end;
    d:= c; dd:= cc
end lngtamvec;
```

Description mca 2500

lngvecvec calculates the sum of the double precision number (c, cc) and the scalar product of the vectors, given in array a[1:u] and array b[shift + 1: shift + u] in double precision, and delivers the result in the double precision number (d, dd).

Description mca 2501

lngmatvec calculates the sum of the double precision number (c, cc) and the scalar product of the rowvector, given in array a[i:i,1:u] and the vector, given in array b[1:u] in double precision, and delivers the result in the double precision number (d, dd).

Description mca 2502

lngtamvec calculates the sum of the double precision number (c, cc) and the scalar product of the columnvector, given in array a[1:u,i:i] and the vector given in array b[1:u] in double precision, and delivers the result in the double precision number (d, dd).

```
comment mca 2503;
procedure lngmatmat(l, u, i, j, a, b, c, cc, d, dd);
value l, u, i, j, c, cc; integer l, u, i, j; real c, cc, d, dd;
array a, b;
begin integer k;
    real x, xx, y, yy, z;
    for k:= l step 1 until u do
    begin xx:= a[i,k]; yy:= b[k,j];
    mul12: x:= xx × 1048577; x:= xx - x + x; xx:= xx - x;
        y:= yy × 1048577; y:= yy - y + y; yy:= yy - y;
        z:= x × yy + xx × y; y:= x × y; x:= y + z;
        y:= y - x + z + xx × yy;
    add2: z:= x + c;
        cc:= (if abs(x) > abs(c) then x - z + c else c - z + x) + y
        + cc; c:= z + cc; cc:= z - c + cc
    end;
    d:= c; dd:= cc
end lngmatmat;

comment mca 2504;
procedure lngtammat(l, u, i, j, a, b, c, cc, d, dd);
value l, u, i, j, c, cc; integer l, u, i, j; real c, cc, d, dd;
array a, b;
begin integer k;
    real x, xx, y, yy, z;
    for k:= l step 1 until u do
    begin xx:= a[k,i]; yy:= b[k,j];
    mul12: x:= xx × 1048577; x:= xx - x + x; xx:= xx - x;
        y:= yy × 1048577; y:= yy - y + y; yy:= yy - y;
        z:= x × yy + xx × y; y:= x × y; x:= y + z;
        y:= y - x + z + xx × yy;
    add2: z:= x + c;
        cc:= (if abs(x) > abs(c) then x - z + c else c - z + x) + y
        + cc; c:= z + cc; cc:= z - c + cc
    end;
    d:= c; dd:= cc
end lngtammat;

comment mca 2505;
procedure lngmattam(l, u, i, j, a, b, c, cc, d, dd);
value l, u, i, j, c, cc; integer l, u, i, j; real c, cc, d, dd;
array a, b;
begin integer k;
    real x, xx, y, yy, z;
    for k:= l step 1 until u do
    begin xx:= a[i,k]; yy:= b[j,k];
    mul12: x:= xx × 1048577; x:= xx - x + x; xx:= xx - x;
        y:= yy × 1048577; y:= yy - y + y; yy:= yy - y;
        z:= x × yy + xx × y; y:= x × y; x:= y + z;
        y:= y - x + z + xx × yy;
    add2: z:= x + c;
        cc:= (if abs(x) > abs(c) then x - z + c else c - z + x) + y
        + cc; c:= z + cc; cc:= z - c + cc
    end;
    d:= c; dd:= cc
end lngmattam;
```

Description mca 2503

lngmatmat calculates the sum of the double precision number (c, cc) and the scalarproduct of the rowvector given in array a[i:i,1:u] and the columnvector given in array b[1:u,j:j] in double precision, and delivers the result in the double precision number (d, dd).

Description mca 2504

lngtammat calculates the sum of the double precision number (c, cc) and the scalarproduct of the columnvectors given in array a[1:u,i:i] and array b[1:u,j:j] in double precision, and delivers the result in the double precision number (d, dd).

Description mca 2505

lngmattam calculates the sum of the double precision number (c, cc) and the scalarproduct of the rowvectors, given in array a[i:i,1:u] and array b[j:j,1:u] in double precision, and delivers the result in the double precision number (d, dd).

CHAPTER 26

LINEAR SYSTEMS

This chapter contains procedures for solving real and complex linear systems of equations, using double precision scalarproduct procedures, followed by iteration. In both cases triangular decomposition with partial pivoting is used.

For large order n the computation time for solving linear systems is proportional to n cubed.
The scalarproducts in double precision require additional time.

Section 260 Triangular decomposition with partial pivoting (real)

This section contains procedures for solving real systems of linear equations.
detaccsol and lngdetsol solve a system of linear equations and calculate the determinant of the system;
lngdet calculates the determinant of a matrix;
lngsol solves a system of linear equations, provided the matrix is given in the triangularly decomposed form as produced by lngdet;
accsol solves a system of linear equations and improves the solution using sol (mca 2101 [6]) resp. lngsol, provided the matrix is given in the triangularly decomposed form as produced by det (mca 2100 [6]) resp. lngdet. One call of det resp. lngdet, followed by several calls of sol resp. lngsol or followed by several calls of accsol using sol resp. lngsol, may be used to solve several linear systems, having the same matrix but different right-hand sides.

The method used in lngdet is triangular decomposition with stabilizing row-interchanges, also called partial pivoting, [6] [7, p. 115], using double-precision scalarproduct procedures. The method yields a lower triangular matrix L and a unit uppertriangular matrix U, such that the product LU equals the given matrix M with permuted rows.
The process is performed in n steps. The k-th step, k = 1,...,n, produces the k-th column of L; subsequently, the "pivot" is selected in this column; the pivotal row and the k-th row of M (and thus also of L) are interchanged; finally, the k-th row of U is produced. That element of the k-th column of L is chosen as pivot, whose absolute value divided by the Euclidean norm of the corresponding row of M is maximal. Thus, matrix M is "equilibrated" in this pivoting strategy, such that the rows effectively obtain unit Euclidean norm. The determinant equals the product of the diagonalelements of L. After the triangular decomposition, lngsol obtains the solution x of the system Mx = b by first permuting the elements of b in the same way as the rows of M, than calculating y, such that Ly equals b with permuted elements (forward substitution) and finally calculating x, such that Ux = y (back substitution). After the triangular decomposition accsol obtains the solution of the system Mx = b using lngsol and improves the solution by iteration.

```
comment mca 2600;
integer procedure lngdet(a, n, aux, p); value n; integer n;
array a, aux; integer array p;
begin integer i, j, k, l, d2;
    real x, y, yy, d1, eps;
    array norm[1:n];
    eps:= aux[0];
    for i:= 1 step 1 until n do
    begin lngmattam(1, n, i, i, a, a, 0, 0, y, yy);
        norm[i]:= 1 / sqrt(y)
    end i;
    d1:= 1; d2:= 0;
    for j:= 1 step 1 until n do
    begin l:= j; x:= 0;
        for i:= j step 1 until n do
        begin lngmatmat(1, j - 1, i, j, a, a, - a[i,j], 0, y, yy);
            a[i,j]:= - y; y:= abs(y × norm[i]); if y > x then
            begin x:= y; l:= i end
        end;
        if l ≠ j then
        begin d1:= - d1; ichrow(1, n, j, l, a); norm[l]:= norm[j]
        end;
        p[j]:= l; d1:= d1 × a[j,j]; if x < eps then
        begin aux[3]:= x; goto end end;
    l1: if abs(d1) ≥ 1 then
        begin d1:= d1 × 0.0625; d2:= d2 + 4; goto l1 end;
    l2: if abs(d1) < 0.0625 then
        begin d1:= d1 × 16; d2:= d2 - 4; goto l2 end;
        x:= - 1 / a[j,j];
        for i:= j + 1 step 1 until n do
        begin lngmatmat(1, j - 1, j, i, a, a, - a[j,i], 0, y, yy);
            a[j,i]:= x × y
        end i
    end j;
    j:= n + 1;
end: lngdet:= j - 1; aux[1]:= d1; aux[2]:= d2
end lngdet;
```

Description mca 2600

lngdet:= n, the order of the matrix M, given in array a[1:n,1:n] and the triangular decomposition of M is performed. The resulting lower-triangular matrix and unit upper-triangular matrix with its unit diagonal omitted are overwritten on a. In array aux[0:5] one must give the relative tolerance in aux[0].

The pivotal indices are delivered in integer array p[1:n] and the determinant of M as aux[1] * 2 ↑ aux[2].

If, however, during the decomposition, a pivot is smaller than aux[0], the process is discontinued, lngdet:= the last stage number minus one, the determinant of the decomposed part is delivered as aux[1] * 2 ↑ aux[2] and the pivot in aux[3].

lngdet uses ichrow ([6], chapter 20), lngmatmat and lngmattam (chapter 25). (see also [3]).

```
comment mca 2601;
procedure lngsol(a, n, p, b); value n; integer n; array a, b;
integer array p;
begin integer i, j, k;
    real x, xx, pi;
    for i:= 1 step 1 until n do
    begin pi:= p[i]; if pi ≠ i then
        begin x:= b[i]; b[i]:= b[pi]; b[pi]:= x end
    end;
    for i:= 1 step 1 until n do
    begin lngmatvec(1, i - 1, i, a, b, b[i], 0, x, xx);
        b[i]:= - x / a[i,i]
    end;
    for i:= n step - 1 until 1 do
    begin lngmatvec(i + 1, n, i, a, b, b[i], 0, x, xx); b[i]:= - x
    end
end lngsol;
```

Description mca 2601

lngsol should be called after lngdet and solves the linear system Mx = b, where M is the n-th order matrix, whose triangularly decomposed form and pivotal indices, as produced by lngdet, must be given in array a[1:n,1:n] and integer array p[1:n], and where b is given as array b[1:n].

The solutionvector x is overwritten on b.

lngsol leaves a and p intact, so that, after one call of lngdet, several calls of lngsol may follow, for solving several systems, having the same matrix, but different right-hand sides.

lngsol uses lngmatvec (chapter 25).

(see also [3]).

```
comment mca 2602;
procedure accsol(a, aa, n, p, b, aux, bb, x, sol); value n; integer n;
array a, aa, aux, x, b, bb; integer array p; procedure sol;
begin integer i, j, k, l;
    real d0, d1, c, cc, xmax, dxmax, eps;
    eps:= aux[0];
    for i:= 1 step 1 until n do
    begin x[i]:= 0; bb[i]:= b[i] end;
    l:= 0;
loop: sol(aa, n, p, bb);
    for i:= 1 step 1 until n do x[i]:= x[i] + bb[i]; l:= l + 1;
    xmax:= dxmax:= 0;
    for i:= 1 step 1 until n do
    begin c:= abs(x[i]); if c > xmax then xmax:= c; c:= abs(bb[i]);
        if c > dxmax then dxmax:= c;
        lngmatvec(1, n, i, a, x, - b[i], 0, c, cc); bb[i]:= - c
    end;
    d1:= dxmax / xmax; if d1 > 2 × eps then
    begin if l > 1 ⌝ d1 × 2 < d0 then
        begin d0:= d1; goto loop end;
        l:= - 1
    end;
    aux[4]:= d1; aux[5]:= l
end acc sol;
```

Description mca 2602

accsol should be called after det resp. lngdet, and solves the linear system Mx = b, where M is the n-th order matrix, given in array a[1:n,1:n], whose triangularly decomposed form and pivotal indices, as produced by det resp. lngdet, must be given in array aa[1:n,1:n] and integer array p[1:n], and where b is given in array b[1:n]. In array aux[0:5] one must give the machine precision in aux[0].

The solution is calculated iteratively, starting from the trial solution x = 0. In each step the residual bb is calculated, and then the system Md = bb is solved, using sol resp. lngsol, and subsequently the correction d is added to x. The refinement is repeated as long as the maximum correction is at most half the previous one, until it is less than twice aux[0] times the maximum norm of x. If the solution fails to improve, the iterationprocess is discontinued, and the number of iterations is delivered with a negative sign. The solution x and the residual bb = b - Mx are delivered in array x[1:n] and bb[1:n].

aux[4] := a measure for the maximal relative error in the solution

aux[5] := the number of iterations.

accsol leaves a, aa, p and b intact, so that a and b can be used to refine the solution x, and so that after one call of det resp. lngdet several calls of accsol with sol resp. lngsol may follow, for solving several systems having the same matrix but different right-hand sides.

accsol uses lngmatvec (chapter 25) and sol resp. lngsol, depending on accsol being preceded by det resp. lngdet, and indirectly matvec ([6], chapter 20). (see also [3]).

```
comment mca 2603;
real procedure detaccsol(a, n, b, aux); value n; integer n;
array a, b, aux;
begin integer i, j;
    array aa[1:n,1:n], bb, x[1:n];
    integer array p[1:n];
    for i:= 1 step 1 until n do
    for j:= 1 step 1 until n do aa[i,j]:= a[i,j];
    detaccsol:= det(aa, n, p);
    accsol(a, aa, n, p, b, aux, bb, x, sol);
    for i:= 1 step 1 until n do b[i]:= x[i]
end detaccsol;

comment mca 2604;
integer procedure lngdetsol(a, n, b, aux); value n; integer n;
array a, b, aux;
begin integer i, j, r;
    array aa[1:n,1:n], bb, x[1:n];
    integer array p[1:n];
    for i:= 1 step 1 until n do
    for j:= 1 step 1 until n do aa[i,j]:= a[i,j];
    r:= lngdetsol:= lngdet(aa, n, aux, p); if r = n then
    begin accsol(a, aa, n, p, b, aux, bb, x, lngsol);
        for i:= 1 step 1 until n do b[i]:= x[i]
    end
end lngdetsol;
```

Description mca 2603

detaccsol:= the determinant of the n-th order matrix M, given in array a[1:n,1:n] and the linear system Mx = b, where vector b is given in array b[1:n], is solved and the solution is improved by iteration. In array aux[0:5] one must give the machine precision in aux[0].
The solution x is overwritten on b.
aux[4]:= a measure for the maximal relative error in the solution.
aux[5]:= the number of iterations, with a negative sign,if the solution failed to improve.
detaccsol leaves a intact.
detaccsol uses det, sol ([6], chapter 21) and accsol and indirectly matvec, matmat, mattam, ichrow ([6], chapter 20) and lngmatvec (chapter 25).

Description mca 2604

lngdetsol:= n , the order of the matrix M, given in array a[1:n,1:n] and the linear system Mx = b, where vector b is given in array b[1:n] is solved and the solution is improved by iteration. In array aux[0:5] one must give the machine precision in aux[0].
The solution x is overwritten on b.
The determinant of M is delivered as aux[1] * 2 ↑ aux[2].
aux[4]:= a measure for the maximal relative error in the solution
aux[5]:= the number of iterations, with a negative sign, if the solution failed to improve.
If, however, during the decomposition by lngdet a pivot is smaller than aux[0], the process is discontinued, lngdetsol:= the last stage number minus one, the determinant of the decomposed part is delivered as aux[1] * 2 ↑ aux[2], the pivot in aux[3] and the procedure is left, without solving the system Mx = b.
lngdetsol leaves a intact.
lngdetsol uses lngdet, lngsol and accsol and indirectly ichrow ([6], chapter 20), lngmatvec, lngmatmat and lngmattam (chapter 25).

Section 265 Triangular decomposition with partial pivoting (complex).

This section contains procedures for solving complex systems of linear equations.

lngcomdetsol solves a complex system of linear equations and calculates the determinant of the system,

lngcomdet calculates the determinant of a matrix,

lngcomsol solves a complex system of linear equations,

provided the matrix is given in the triangularly decomposed form as produced by lngcomdet.

comaccsol solves a complex system of linear equations and improves the solution using lngcomsol, provided the matrix is given in the triangularly decomposed form as produced by lngcomdet. One call of lngcomdet, followed by several calls of lngcomsol or comaccsol, using lngcomsol, may be used to solve several complex linear systems, having the same matrix, but different right-hand sides.

The method, used in this section, is analogous to the method described in section 260 for the real case.

The complex scalarproducts are written as four real scalarproducts.

```
comment mca 2650;
integer procedure lngcomdet(ar, ai, n, aux, p); value n; integer n;
array ar, ai, aux; integer array p;
begin integer i, j, j1, k, l, dete;
    real u, v, w, x, y, z, detr, deti;
    array norm[1:n];
    for i:= 1 step 1 until n do
    begin lngmattam(1, n, i, i, ar, ar, 0, 0, x, w);
        lngmattam(1, n, i, i, ai, ai, x, w, norm[i], w)
    end;
    detr:= 1; deti:= 0; dete:= 0;
    for j:= 1 step 1 until n do
    begin l:= j; z:= 0; j1:= j - 1;
        for i:= j step 1 until n do
        begin lngmatmat(1, j1, i, j, ar, ar, - ar[i,j], 0, x, y);
            lngmatmat(1, j1, i, j, ai, ai, - x, - y, x, y);
            lngmatmat(1, j1, i, j, ai, ar, - ai[i,j], 0, y, w);
            lngmatmat(1, j1, i, j, ar, ai, y, w, y, w); ar[i,j]:= x;
            y:= ai[i,j]:= - y; x:= (x × x + y × y) / norm[i];
            if x > z then
            begin z:= x; l:= i end
        end;
        if l ≠ j then
        begin detr:= - detr; deti:= - deti; ichrow(1, n, j, l, ar);
            ichrow(1, n, j, l, ai); p[l]:= p[j]
        end;
        p[j]:= l; x:= ar[j,j]; y:= ai[j,j]; z:= x × x + y × y;
        w:= x × detr - y × deti; deti:= x × deti + y × detr;
        detr:= w;
        if abs(detr) > abs(deti) then w:= detr else w:= deti;
        if w = 0 then
        begin dete:= 0; goto endd end;
    l1: if abs(w) ≥ 1 then
        begin w:= w × 0.0625; detr:= detr × 0.0625;
            deti:= deti × 0.0625; dete:= dete + 4; goto l1
        end;
    l2: if abs(w) < 0.0625 then
        begin w:= w × 16; detr:= detr × 16; deti:= deti × 16;
            dete:= dete - 4; goto l2
        end;
        j1:= j - 1;
        for i:= j + 1 step 1 until n do
        begin lngmatmat(1, j1, j, i, ar, ar, - ar[j,i], 0, w, v);
            lngmatmat(1, j1, j, i, ai, ai, - w, - v, w, v);
            lngmatmat(1, j1, j, i, ai, ar, - ai[j,i], 0, v, u);
            lngmatmat(1, j1, j, i, ar, ai, v, u, v, u); v:= - v;
            ar[j,i]:= (w × x + v × y) / z;
            ai[j,i]:= (v × x - w × y) / z
        end
    end;
    j:= n + 1;
endd: lngcomdet:= j - 1; aux[1]:= detr; aux[2]:= deti;
    aux[3]:= dete
end lngcomdet;
```

Description mca 2650

lngcomdet:= n , the order of the complex matrix M, whose real part is given in array ar[1:n,1:n] and imaginary part in array ai[1:n,1:n] and the triangular decomposition of M is performed. The resulting lowertriangular matrix and unit uppertriangular matrix with its unit diagonal omitted, are overwritten on ar and ai.

The pivotal indices are delivered in integer array p[1:n] and the determinant of M as (aux[1] + i * aux[2]) * 2 ↑ aux[3].

If, however, during the decomposition, a diagonalelement of the lower-triangular matrix becomes zero, the process is discontinued, lngcomdet:= the last stage number minus one, and the determinant of the decomposed part is delivered as (aux[1] + i * aux[2]) * 2 ↑ aux[3].

lngcomdet uses ichrow ([6], chapter 20), lngmatmat and lngmattam (chapter 25). (see also [3]).

```
comment mca 2651;
procedure lngcomsol(ar, ai, n, p, br, bi); value n; integer n;
array ar, ai, br, bi; integer array p;
begin integer i, i1, j, k;
    real x, y, z, a1, a2;
    for i:= 1 step 1 until n do
    begin k:= p[i]; if k ≠ i then
        begin x:= br[i]; br[i]:= br[k]; br[k]:= x; x:= bi[i];
            bi[i]:= bi[k]; bi[k]:= x
        end
    end;
    for i:= 1 step 1 until n do
    begin i1:= i - 1; lngmatvec(1, i1, i, ar, br, - br[i], 0, x, y);
        lngmatvec(1, i1, i, ai, bi, - x, - y, x, y);
        lngmatvec(1, i1, i, ar, bi, - bi[i], 0, y, z);
        lngmatvec(1, i1, i, ai, br, y, z, y, z); y:= - y;
        a1:= ar[i,i]; a2:= ai[i,i]; z:= a1 × a1 + a2 × a2;
        br[i]:= (x × a1 + y × a2) / z; bi[i]:= (y × a1 - x × a2) / z
    end;
    for i:= n step - 1 until 1 do
    begin i1:= i + 1; lngmatvec(i1, n, i, ar, br, - br[i], 0, x, y);
        lngmatvec(i1, n, i, ai, bi, - x, - y, br[i], y);
        lngmatvec(i1, n, i, ar, bi, - bi[i], 0, y, z);
        lngmatvec(i1, n, i, ai, br, y, z, y, z); bi[i]:= - y
    end
end lngcomsol;
```

Description mca 2651

lngcomsol should be called after lngcomdet and solves the complex linear system Mx = b, where M is the n-th order matrix whose triangularly decomposed form and pivotal indices, as produced by lngcomdet, must be given in array ar[1:n,1:n] (real part of M), ai[1:n,1:n] (imaginary part of M) and integer array p[1:n], and where b is given in array br[1:n] (real part of b) and bi[1:n] (imaginary part of b).

The solutionvector x is overwritten on b.

lngcomsol leaves ar, ai and p intact, so that after one call of lngcomdet, several calls of lngcomsol may follow, for solving several complex systems, having the same matrix, but different right-hand sides.

lngcomsol uses lngmatvec (chapter 25).

(see also [3]).

```
comment mca 2652;
procedure comaccsol(ar, ai, aar, aai, n, p, br, bi, aux, bbr, bbi,
xr, xi); value n; integer n;
array ar, ai, aar, aai, br, bi, bbr, bbi, xr, xi; integer array p;
begin integer i, j, k, l;
    real e, eps, x, y, d0, d1, xmax, dxmax;
    for i:= 1 step 1 until n do
    begin xr[i]:= xi[i]:= 0; bbr[i]:= br[i]; bbi[i]:= bi[i] end;
    l:= 0; eps:= aux[0]; eps:= eps × eps;
loop: t:= time; lngcomsol(aar, aai, n, p, bbr, bbi);
    t:= time - t; t1:= t1 + t; l:= l + 1;
    for i:= 1 step 1 until n do
    begin xr[i]:= xr[i] + bbr[i]; xi[i]:= xi[i] + bbi[i] end;
    xmax:= dxmax:= 0;
    for i:= 1 step 1 until n do
    begin e:= xr[i] ↑ 2 + xi[i] ↑ 2; if e > xmax then xmax:= e;
        e:= bbr[i] ↑ 2 + bbi[i] ↑ 2; if e > dxmax then dxmax:= e;
        lngmatvec(1, n, i, ar, xr, - br[i], 0, x, y);
        lngmatvec(1, n, i, ai, xi, - x, - y, bbr[i], y);
        lngmatvec(1, n, i, ar, xi, - bi[i], 0, x, y);
        lngmatvec(1, n, i, ai, xr, x, y, x, y); bbi[i]:= - x
    end;
    d1:= dxmax / xmax; if d1 > 4 × eps then
    begin if l > 1 ⌝ d1 × 4 < d0 then
        begin d0:= d1; goto loop end;
        l:= - 1
    end;
    aux[4]:= d1; aux[5]:= l
end comaccsol;
```

Description mca 2652

comaccsol should be called after lngcomdet, and solves the complex linear system Mx = b, where M is the n-th order matrix, whose real part must be given in array ar[1:n,1:n] and imaginary part in array ai[1:n,1:n] and whose triangularly decomposed form and pivotal indices, as produced by lngcomdet, must be given in array aar, aai|1:n,1:n| and integer array p|1:n| and where b is given in array br, bi[1:n], where br contains the real part of b and bi the imaginary part.

In array aux[0:5] one must give the machine precision in aux[0].

The solution is calculated iteratively, starting from the trial solution x = 0. In each step the residual bb is calculated and then the system Md = bb is solved, using lngcomsol, and subsequently the correction d is added to x.

The refinement is repeated as long as the maximum correction is at most half the previous one, until it is less than twice aux[0] times the maximum norm of x. If the solution fails to improve, the iterationprocess is discontinued, and the number of iterations is delivered with a negative sign.

The solution x and the residual bb = b - Mx are delivered in array xr, xi, bbr, bbi[1:n], where xr and bbr contain the real parts of x and bb, and xi and bbi the imaginary parts.

aux[4]:= a measure for the maximal relative error in the solution.

aux[5]:= the number of iterations.

comaccsol leaves ar, ai, aar, aai, p, br and bi intact, so that ar, ai, br and bi can be used to refine the solution x,and so that after one call of lngcomdet several calls of comaccsol with lngcomsol may follow, for solving several complex systems having the same matrix but different right-hand sides.

comaccsol uses lngmatvec (chapter 25) and lngcomsol.

(see also [3]).

```
comment mca 2654;
integer procedure lngcomdetsol(ar, ai, n, br, bi, aux); value n;
integer n; array ar, ai, br, bi, aux;
begin integer i, j, r;
    array aar, aai[1:n,1:n], bbr, bbi, xr, xi[1:n];
    integer array p[1:n];
    for i:= 1 step 1 until n do
    for j:= 1 step 1 until n do
    begin aar[i,j]:= ar[i,j]; aai[i,j]:= ai[i,j] end;
    r:= lngcomdetsol:= lngcomdet(aar, aai, n, aux, p); if r = n then
    begin comaccsol(ar, ai, aar, aai, n, p, br, bi, aux, bbr, bbi,
        xr, xi);
        for i:= 1 step 1 until n do
        begin br[i]:= xr[i]; bi[i]:= xi[i] end
    end
end lngcomdetsol;
```

Description mca 2654

lngcomdetsol:= n , the order of the matrix M, whose real part is given in array ar[1:n,1:n] and imaginary part in array ai[1:n,1:n] and the linear system Mx = b is solved, where vector b is given in array br, bi[1:n,1:n], where br contains the real part of b and bi the imaginary part, and the solution is improved by iteration. In array aux[0:5] one must give the machine precision in aux[0].

The solution x is overwritten on b.

The determinant of M is delivered as (aux[1] + i * aux[2]) * 2 ↑ aux[3].

aux[4]:= a measure for the maximal relative error in the solution.

aux[5]:= the number of iterations, with a negative sign, if the solution failed to improve.

If, however, during the decomposition by lngcomdet, a diagonalelement of the lowertriangular matrix becomes zero the process is discontinued, lngcomdetsol:= the last stage number minus one, the determinant of the decomposed part is delivered as (aux[1] + i * aux[2]) * 2 ↑ aux[3] and the procedure is left without solving the system Mx = b.

lngcomdetsol leaves ar and ai intact.

lngcomdetsol uses lngcomdet, lngcomsol and comaccsol and indirectly ichrow ([6], chapter 20), lngmatvec, lngmatmat and lngmattam (chapter 25).

CHAPTER 27

POSITIVE DEFINITE SYMMETRIC LINEAR SYSTEMS

This chapter contains procedures for solving systems of linear equations, provided the matrices are positive definite symmetric.
In section 270 procedures are given analogous to mca 2200 and mca 2201 (chapter 22, [6]), but using four formal parameters and using single precision scalarproduct procedures.
Section 271 contains the same procedures as section 270, but using double precision scalarproduct procedures.
Moreover section 274 contains procedures for solving linear least-squares problems, using double precision scalar product procedures.

In section 270 and 271 the ordinary Cholesky method is used and in section 274 Householder transformations with pivoting [4].

For large order n the computation time for solving linear systems is proportional to n cubed and about one half of the time required for the general case.

Section 270 Cholesky decomposition without pivoting, using single precision scalar product procedures

This section contains procedures for solving linear systems, provided the matrices are positive definite symmetric:
detaccsolsym solves a system of linear equations and improves the solution by iteration and calculates the determinant of the system,
detsym calculates the determinant of a matrix,
solsym solves a linear system of equations,
accsolsym solves a linear system of equations and improves the solution by iteration.

The method used is Cholesky's square root method without pivoting [7, p. 117], [2]. If the given symmetric matrix M is positive definite, then the method yields an upper triangular matrix U, the "Cholesky matrix" of M, such that U'U equals M; moreover the determinant of M is delivered, calculated as the product of the squares of the diagonal elements of U (and, thus, always positive). The process is completed in n stages, each stage producing a row of U.
However, the process is dicontinued, if at some stage, k, the k-th diagonal-element of M minus the sum of the squared elements of the k-th column of U (the square root of this quantity being the k-th diagonal element of U) is not positive, meaning that M, perhaps modified by rounding errors, is not positive definite. In that case, the stage number k-1 is delivered.
After the Cholesky decomposition the solution of the linear system U'Ux = b is obtained by solving U'y = b (forward substitution) and Ux = y (back substitution).

```
comment mca 2700;
integer procedure detsym(a, n, aux, diag); value n; integer n;
array a, aux, diag;
begin integer i, j, d2;
    real x, xx, d1;
    d1:= 1; d2:= 0;
    for i:= 1 step 1 until n do
    for j:= i step 1 until n do
    begin x:= a[i,j] - mattam(1, i - 1, i, j, a, a); if j = i then
        begin if x < 0 then
            begin aux[3]:= x; goto end end;
            d1:= d1 × x;
        l1: if abs(d1) ≥ 1 then
            begin d1:= d1 × 0.0625; d2:= d2 + 4; goto l1 end;
        l2: if abs(d1) < 0.0625 then
            begin d1:= d1 × 16; d2:= d2 - 4; goto l2 end;
            diag[i]:= 1 / sqrt(x)
        end
        else a[j,i]:= x × diag[i]
    end;
    j:= n + 1;
end: detsym:= j; aux[1]:= d1; aux[2]:= d2
end detsym;

comment mca 2701;
procedure solsym(a, n, diag, b); value n; integer n; array a, diag, b;
begin integer i;
    real y, yy;
    for i:= 1 step 1 until n do
    begin b[i]:= - diag[i] × (b[i] + matvec(1, i - 1, i, a, b)) end;
    for i:= n step - 1 until 1 do
    begin b[i]:= - diag[i] × (b[i] + tamvec(i + 1, n, i, a, b)) end
end solsym;
```

Description mca 2700

detsym:= n , the order of the positive definite symmetric matrix M, whose uppertriangle is given in array a[1:n,1:n], and the Cholesky matrix of M is calculated and stored in the remainder of a, except the reciprocals of the diagonal elements which are stored in array diag[1:n]. Moreover, in array aux[0:5] the determinant of M is delivered in aux[1] * 2 ↑ aux[2]. If, however, M is not positive definite, the Cholesky-decomposition is discontinued, detsym:= the last stage number minus one, the determinant of the decomposed part is delivered as aux[1] * 2 ↑ aux[2] and the non-positive element is delivered in aux[3].
detsym leaves the uppertriangle of a, thus also M, intact.
detsym uses mattam ([6], chapter 20).

Description mca 2701

solsym should be called after detsym and solves the linear system Mx = b, where M is the n-th order positive definite symmetric matrix, given in the uppertriangle of array a[1:n,1:n], whose Cholesky matrix, as produced by detsym must be given in the lowertriangle of a, except for the diagonalelements, whose reciprocals must be given in array diag[1:n], and where b is given in array b[1:n].
The solutionvector x is overwritten on b.
solsym leaves the elements of a invariant, so that after one call of detsym several calls of solsym may follow for solving several linear systems having the same matrix, but different right-hand sides.
solsym uses matvec and tamvec ([6], chapter 20).

```
comment mca 2702;
procedure accsolsym(a, n, b, aux, diag, bb, x, solsym); value n;
integer n; array a, b, aux, diag, bb, x; procedure solsym;
begin integer i, l;
    real d0, d1, c, cc, xmax, dxmax, eps;
    eps:= aux[0];
    for i:= 1 step 1 until n do
    begin x[i]:= 0; bb[i]:= b[i] end;
    l:= 0;
loop: solsym(a, n, diag, bb); l:= l + 1;
    for i:= 1 step 1 until n do x[i]:= x[i] + bb[i]; xmax:= dxmax:= 0;
    for i:= 1 step 1 until n do
    begin c:= abs(x[i]); if c > xmax then xmax:= c; c:= abs(bb[i]);
        if c > dxmax then dxmax:= c;
        lngtamvec(1, i - 1, i, a, x, - b[i], 0, c, cc);
        lngmatvec(i, n, i, a, x, c, cc, c, cc); bb[i]:= - c
    end;
    d1:= dxmax / xmax; if d1 > 2 × eps then
    begin if l > 1 ∧ d1 × 2 < d0 then
        begin d0:= d1; goto loop end;
        l:= - l
    end;
    aux[4]:= d1; aux[5]:= l
end accsolsym;
```

Description mca 2702

accsolsym should be called after detsym resp. lngdetsym (section 271) and solves the linear system Mx = b, where M is the n-th order positive definite symmetric matrix, given in the uppertriangle of array a[1:n,1:n], whose Cholesky matrix, as produced by detsym resp. lngdetsym, must be given in the lowertriangle of a, except for the diagonalelements whose reciprocals must be given in array diag[1:n], and where b is given in array b[1:n]. In array aux[0:5] one must give the machine precision in aux[0].

The solution is calculated iteratively, starting from the trial solution x = 0. In each step the residual bb is calculated and then the system Md = bb is solved, using solsym resp. lngsolsym (section 271), and subsequently the correction d is added to x. The refinement is repeated as long as the maximum correction is at most half the previous one, until it is less than twice aux[0] times the maximum norm of x. If the solution fails to improve, the iteration-process is discontinued, and the number of iterations is delivered with a negative sign. The solution x and the residual bb = b - Mx are delivered in array x[1:n] and bb[1:n].

aux[4]:= a measure for the maximal relative error in the solution.

aux[5]:= the number of iterations.

accsolsym leaves a, diag and b intact, so that the uppertriangle of a and b can be used to refine the solution x, and so that after one call of detsym resp. lngdetsym, several calls of accsolsym with solsym resp. lngsolsym may follow, using the lower_triangle of a (minus the maindiagonal) and diag, for solving several systems having the same matrix, but different right-hand sides.

accsolsym uses lngmatvec and lngtamvec (chapter 25) and solsym resp. lngsolsym, depending on accsolsym being preceded by detsym or lngdetsym, and indirectly matvec and tamvec ([6], chapter 20).

(see also [2b]).

```
comment mca 2703;
integer procedure detaccsolsym(a, n, b, aux); value n; integer n;
array a, b, aux;
begin integer i, r;
    array diag, bb, x[1:n];
    detaccsolsym:= r:= detsym(a, n, aux, diag); if r = n then
    begin accsolsym(a, n, b, aux, diag, bb, x, solsym);
        for i:= 1 step 1 until n do b[i]:= x[i]
    end
end detaccsolsym;
```

Description mca 2703

detaccsolsym:= n ., the order of the positive definite symmetric matrix M, whose uppertriangle is given in array a[1:n,1:n], and the linear system Mx = b, where vector b is given in array b[1:n], is solved and the solution is improved by iteration. In array aux[0:5] one must give the machine precision in aux[0].

The solution x is overwritten on b.

The determinant of M is delivered as aux[1] × 2 ↑ aux[2].

aux[4]:= a measure for the maximal relative error in the solution.

aux[5]:= the number of iterations, with a negative sign, if the solution failed to improve.

If, however, M is not positive definite, the Cholesky decomposition is discontinued, detaccsolsym:= the last stage number minus one, the determinant of the decomposed part is delivered as aux[1] × 2 ↑ aux[2], the non-positive element in aux[3] and the procedure is left without solving the system Mx = b. detaccsolsym leaves a intact.

detaccsolsym uses detsym, solsym and accsolsym, and indirectly matvec, tamvec and mattam ([6], chapter 20) and lngmatvec (chapter 25).

```
comment mca 2710;
integer procedure lngdetsym(a, n, aux, diag); value n; integer n;
array a, aux, diag;
begin integer i, j, d2;
    real x, xx, d1;
    d1:= 1; d2:= 0;
    for i:= 1 step 1 until n do
    for j:= i step 1 until n do
    begin lngmattam(1, i - 1, i, j, a, a, - a[i,j], 0, x, xx);
        x:= - x; if j = i then
        begin if x ≤ 0 then
            begin aux[3]:= x; goto end end;
            d1:= d1 × x;
        l1: if abs(d1) ≥ 1 then
            begin d1:= d1 × 0.0625; d2:= d2 + 4; goto l1 end;
        l2: if abs(d1) < 0.0625 then
            begin d1:= d1 × 16; d2:= d2 - 4; goto l2 end;
            diag[i]:= 1 / sqrt(x)
        end
        else a[j,i]:= x × diag[i]
    end;
    j:= n + 1;
end: lngdetsym:= j; aux[1]:= d1; aux[2]:= d2
end lngdetsym;
```

Section 271 Cholesky decomposition without pivoting, using double precision scalar product procedures.

This section contains procedures for solving linear systems, provided the matrices are positive definite symmetric:
lngdetsolsym solves a system of linear equations and improves the solution by iteration and calculates the determinant of the system,
lngdetsym calculates the determinant of a matrix,
lngsolsym solves a linear system of equations.
The method used is Cholesky's square root method without pivoting [7, p. 117], [2]. (see section 270).

Description mca 2710

lngdetsym:= n, the order of the positive definite symmetric matrix M, whose uppertriangle is given in array a[1:n,1:n] and the Cholesky matrix of M is calculated and stored in the remainder of a, except the reciprocals of the diagonal elements, which are stored in array diag[1:n]. Moreover, in array aux[0:5] the determinant of M is delivered as aux[1] * 2 ↑ aux[2].
If, however, M is not positive definite, the Cholesky decomposition is discontinued, lngdetsym:= the last stage number minus one, the determinant of the decomposed part is delivered as aux[1] * 2 ↑ aux[2] and the non-positive element is delivered in aux[3].
lngdetsym leaves the uppertriangle of a, thus also M, intact.
lngdetsym uses lngmattam (chapter 20).

```
comment mca 2711;
procedure lngsolsym(a, n, diag, b); value n; integer n;
array a, diag, b;
begin integer i;
    real y, yy;
    for i:= 1 step 1 until n do
    begin lngmatvec(1, i - 1, i, a, b, b[i], 0, y, yy);
        b[i]:= - diag[i] × y
    end;
    for i:= n step - 1 until 1 do
    begin lngtamvec(i + 1, n, i, a, b, b[i], 0, y, yy);
        b[i]:= - diag[i] × y
    end
end lngsolsym;

comment mca 2713;
integer procedure lngdetsolsym(a, n, b, aux); value n; integer n;
array a, b, aux;
begin integer i, r;
    array diag, bb, x[1:n];
    lngdetsolsym:= r:= lngdetsym(a, n, aux, diag); if r = n then
    begin accsolsym(a, n, b, aux, diag, bb, x, lngsolsym);
        for i:= 1 step 1 until n do b[i]:= x[i]
    end
end lngdetsolsym;
```

Description mca 2711

lngsolsym should be called after lngdetsym and solves the linear system Mx = b, where M is the n-th order positive definite symmetric matrix, given in the upper triangle of array a[1:n,1:n], whose Cholesky matrix, as produced by lngdetsym must be given in the lower triangle of a, except for the diagonal-elements, whose reciprocals must be given in array diag[1:n], and where b is given in array b[1:n].

The solutionvector x is overwritten on b.

lngsolsym leaves the elements of a invariant, so that after one call of lngdetsym several calls of lngsolsym may follow for solving several linear systems having the same matrix, but different right-hand sides.

lngsolsym uses lngmatvec and lngtamvec (chapter 25).

Description mca 2713

lngdetsolsym:= n ., the order of the positive definite symmetric matrix M, whose upper triangle is given in array a[1:n,1:n], and the linear system Mx = b, where vector b is given in array b[1:n], is solved and the solution is improved by iteration. In array aux[0:5] one must give the machine precision in aux[0].

The solution x is overwritten on b.

The determinant of M is delivered as aux[1] × 2 ↑ aux[2].

aux[4]:= a measure for the maximal relative error in the solution.

aux[5]:= the number of iterations, with a negative sign, if the solution failed to improve.

If, however, M is not positive definite, the Cholesky decomposition is discontinued, lngdetsolsym:= the last stage number minus one, the determinant of the decomposed part is delivered as aux[1] × 2 ↑ aux[2], the non-positive element in aux[3] and the procedure is left without solving the system Mx = b.

lngdetsolsym leaves a intact.

lngdetsolsym uses lngdetsym, lngsolsym and accsolsym (section 270) and indirectly lngmatvec, lngtamvec and lngmattam (chapter 25).

Section 274 Least-squares problems

This section contains procedures for solving linear least-squares problems, using double precision scalarproducts.
lsqwgtmat ([6], section 224) multiplicates a matrix and a weightvector.

lsqlngdec performs the Householder triangularisation of M and calculates its rank.
lsqlngsol and lsqlngdglinv are to be used in combination with lsqlngdec, for solving a linear least-squares problem (or several problems having the same matrix M, but different right-hand sides), and for calculating the maindiagonal of the inverse of M'M.
Apart from some changes and adaptations to our vector procedures, lsqlngdec and lsqlngsol have been derived from [4].
The method is Householder triangularisation with columninterchanges. Let M have n rows and m columns; lsqlngdec produces an n-th order orthogonal matrix Q and an n * m uppertriangular matrix R, such that R equals QM with permuted columns. Matrix Q is the product of at most m orthogonal symmetric n-th order "Household matrices", which are of the form I - sww', where I is the identity matrix, w a columnvector and s a scalar. Matrix M is reduced to R in (at most) m stages. In the k-th stage the desired zeroes are introduced in the k-th column, i.e. the column having maximum Euclidean norm, is selected from the remaining (n - k + 1) * (m - k + 1) submatrix, and the pivotal and the k-th columns are interchanged; then the k-th Householder matrix is calculated and postmultiplied by the remaining submatrix. The k-th Householder matrix is chosen such that this postmultiplication introduces the desired zeroes in the k-th column and the first k - 1 elements of w are zero. If at some stage k the Euclidean norm of the pivotal column is smaller than some tolerance, viz. a given relative tolerance times the maximum of the Euclidean norms of the columns of M, then the process is discontinued, and k - 1 is delivered as the rank of M; otherwhise the rank equals m.

In lsqlngsol, the least-squares solution x of the problem Mx - b is obtained by first calculating y = Qb, then solving the triangular system consisting of the first equations of Rx = y (backsubstitution), and finally interchanging the elements of x in "reverse correspondence" with the interchanges of the columns of M, i.e. the same interchanges are carried out in reverse order. As by-product the last n-m elements of y are delivered; the sum of the squares of these elements is approximately equal to the square of the Euclidean norm of the residuevector Mx - b.

In lsqlngdglinv the maindiagonal of M'M is obtained by calculating the inverse of R, from this the maindiagonal of the inverse of R'R, and then interchanging the calculated diagonal elements in reverse correspondence with the interchanges of the columns of M.

```
comment mca 2245;
procedure lsqwgtmat(w, a, n, m); value n, m; integer n, m; array w, a;
begin integer i, j;
    real wi;
    for i:= 1 step 1 until n do
    begin wi:= w[i];
        for j:= 1 step 1 until m do a[i,j]:= a[i,j] × wi
    end
end lsqwgtmat;

comment mca 2740;
integer procedure lsqlngdec(a, n, m, aux, aid, ci); value n, m;
integer n, m; array a, aux, aid; integer array ci;
begin integer j, k, kpiv;
    real beta, sigma, norm, w, ww, eps, akk, aidk;
    array sum[1:m];
    norm:= 0; lsqlngdec:= m;
    for k:= 1 step 1 until m do
    begin lngtammat(1, n, k, k, a, a, 0, 0, w, ww); sum[k]:= w;
        if w > norm then norm:= w
    end;
    w:= aux[1]:= sqrt(norm); eps:= aux[0] × w;
    for k:= 1 step 1 until m do
    begin sigma:= sum[k]; kpiv:= k;
        for j:= k + 1 step 1 until m do if sum[j] > sigma then
        begin sigma:= sum[j]; kpiv:= j end;
        if kpiv ≠ k then
        begin sum[kpiv]:= sum[k]; ichcol(1, n, k, kpiv, a) end;
        ci[k]:= kpiv; akk:= a[k,k];
        lngtammat(k, n, k, k, a, a, 0, 0, sigma, ww);
        w:= sqrt(sigma); aidk:= aid[k]:= if akk < 0 then w else - w;
        if w < eps then
        begin lsqlngdec:= k - 1; goto enddec end;
        beta:= 1 / (sigma - akk × aidk); a[k,k]:= akk - aidk;
        for j:= k + 1 step 1 until m do
        begin lngtammat(k, n, k, j, a, a, 0, 0, sigma, ww);
            elmcol(k, n, j, k, a, a, - beta × sigma);
            sum[j]:= sum[j] - a[k,j] ↑ 2
        end
    end for k;
enddec: aux[2]:= w
end lsqlngdec;
```

Description mca 2245

lsqwgtmat multiplicates the matrix M, given in array a[1:n,1:n], and the weightvector w, given in array w[1:n], and delivers M * w in a[1:n,1:m].

Description mca 2740

lsqlngdec:= rank, r, of the n * m matrix M, given in array a[1:n,1:m].
In array aux[0:5] one must give a relative tolerance in aux[0]. The pivotal column indices are delivered in integer array ci[1:r], the (r first) diagonal elements of the uppertriangular matrix R in array aid[1:r], and the other elements of the upper triangle of R in array a, together with the vectors w of the Householder matrices.
Moreover,
aux[1]:= the maximum Euclidean norm of the columns of M,
aux[2]:= the absolute value of the r-th diagonal element of R.
lsqlngdec uses elmcol, ichcol ([6], chapter 20) and lngtammat (chapter 25).

```
comment mca 2741;
procedure lsqlngsol(a, n, m, aid, ci, b); value n, m; integer n, m;
array a, aid, b; integer array ci;
begin integer k, cik;
    real w, ww;
    for k:= 1 step 1 until m do
    begin lngtamvec(k, n, k, a, b, 0, 0, w, ww);
        elmveccol(k, n, k, b, a, w / (aid[k] × a[k,k]))
    end;
    for k:= m step - 1 until 1 do
    begin lngmatvec(k + 1, m, k, a, b, - b[k], 0, w, ww);
        b[k]:= - w / aid[k]
    end;
    for k:= m step - 1 until 1 do
    begin cik:= ci[k]; if cik ≠ k then
        begin w:= b[k]; b[k]:= b[cik]; b[cik]:= w end
    end
end lsqlngsol;

comment mca 2742;
procedure lsqlngdglinv(a, m, aid, ci, diag); value m; integer m;
array a, aid, diag; integer array ci;
begin integer j, k, cik;
    real w, ww;
    for k:= 1 step 1 until m do
    begin diag[k]:= 1 / aid[k];
        for j:= k + 1 step 1 until m do
        begin lngtamvec(k, j - 1, j, a, diag, 0, 0, w, ww);
            diag[j]:= - w / aid[j]
        end;
        lngvecvec(k, m, 0, diag, diag, 0, 0, diag[k], ww)
    end;
    for k:= m step - 1 until 1 do
    begin cik:= ci[k]; if cik ≠ k then
        begin w:= diag[k]; diag[k]:= diag[cik]; diag[cik]:= w end
    end
end lsqlngdglinv;
```

Description mca 2741

lsqlngsol should be called after lsqlngdec (but only if the rank equals m), and calculates the least-squares solution x of Mx = b, where b is the vector given in array b|1:n|, and M is the n * m matrix, whose Householder triangularized form R, with the vectors w of the Householder matrices and the pivotal indices as produced by lsqlngdec must be given in array a[1:n,1:n], aid[1:m] and integer array ci[1:m]. The solution vector x is overwritten on the first m elements of b, and the last n - m elements of y are overwritten on the last n - m elements of b.

lsqlngsol leaves the elements of a, aid and ci intact, so that after one call of lsqlngdec, several calls of lsqlngsol may follow for solving several least-squares problems, having the same matrix M, but different right-hand sides.

lsqlngsol uses lngmatvec, lngtamvec (chapter 25) and elmveccol ([6], chapter 20).

Description mca 2742

lsqlngdglinv should be called after lsqlngdec (but only if the rank equals m), and calculates the maindiagonal of the inverse of M'M where M is the matrix, whose Householder triangularized form R with the pivotal indices, as produced by lsqlngdec must be given in array a[1:n,1:n], aid[1:m] and integer array ci[1:m]. The calculated maindiagonal is delivered in array diag[1:m]; the elements of a, aid and ci are left intact.

lsqlngdglinv uses lngvecvec and lngtamvec (chapter 25).

REFERENCES

1) P. Naur (ed.),
Revised report on the algorithmic language ALGOL 60 (1962).

2) R.S. Martin, G. Peters and J.H. Wilkinson,
a) Symmetric decomposition of a positive definite matrix.
Num. Math. 7, 362-282 (1965).
b) Iterative refinement of the solution of a positive definite system of equations.
Num. Math. 8, 203-216 (1966).

3) H.J. Bowdler, R.S. Martin, G. Peters and J.H. Wilkinson,
Solution of real and complex systems of linear equations.
Num. Math. 8, 217-234 (1966).

4) P. Businger and G. Golub,
Linear least-squares solution by Householdertransformations.
Num. Math. 7, 269-276 (1965).

5) G. Golub and J.H. Wilkinson,
Note on the iterative refinement of least-squares solution.
Num. Math. 9, 139-148 (1966).

6) T.J. Dekker,
ALGOL 60 procedures in Numerical Algebra I.
(Mathematical Centre Tracts 22, Amsterdam 1968).

7) J.H. Wilkinson,
Rounding Errors in algebraïc processes.
(London 1963).

8) J.H. Wilkinson,
The algebraïc eigenvalue problem.
(Oxford 1965).