

Algorithms

ALGORITHM 41 EVALUATION OF DETERMINANT

JOSEF G. SOLOMON

RCA Digital Computation and Simulation Group, Moorestown, New Jersey

```

real procedure Determinant (A,n);
real array A; integer n;
comment This procedure evaluates a determinant by triangularization;
begin real Product, Factor, Temp; array B[1 : n, 1 : n],
    C[1 : n, 1 : n];
integer Count, Sign, i, j, r, y;
    Sign := 1; Product := 1;
for i := 1 step 1 until n do for j := 1 step 1 until
    n do
begin B[i,j] := A[i,j]; C[i,j] := A[i,j] end;
for r := 1 step 1 until n-1 do
begin Count := r-1;
    zerocheck: if B[r,r]  $\neq$  0 then go to resume;
    if Count < n-1 then Count := Count + 1
    else go to zero;
for y := r step 1 until n do
begin Temp := B[Count+1,y]; B[Count+1,y] :=
    B[Count,y]; B[Count,y] := Temp end;
    Sign := - Sign; go to zerocheck;
    zero: Determinant := 0; go to return;
    resume: for i := r+1 step 1 until n do
begin Factor := C[i,r] / C[r,r];
    for j := r+1 step 1 until n do
begin B[i,j] := B[i,j] - Factor  $\times$  C[r,j] end end;
for i := r+1 step 1 until n do
for j := r+1 step 1 until n do C[i,j] := B[i,j]
    end;
for i := 1 step 1 until n do Product := Product
     $\times$  B[i,i]; Determinant := Sign  $\times$  Product;
return: end

```

Contributions to this department must be in the form stated in the Algorithms Department policy statement (*Communications*, February, 1960) except that ALGOL 60 notation should be used (see *Communications*, May, 1960). Contributions should be sent in duplicate to J. H. Wegstein, Computation Laboratory, National Bureau of Standards, Washington 25, D. C. Algorithms should be in the Publication form of ALGOL 60 and written in a style patterned after the most recent algorithms appearing in this department.

Although each algorithm has been tested by its contributor, no warranty, express or implied, is made by the contributor, the editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material and no responsibility is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm.

ALGORITHM 42

INVERT

T. C. Wood

RCA Digital Computation and Simulation Group, Moorestown, New Jersey

```

procedure Invert (A) order: (n) Singular: (s) Inverse: (A1);
    array A, A1; integer n,s,value n;
comment This procedure inverts the square matrix A of order
    n by applying a series of elementary row operation to the matrix
    to reduce it to the identity matrix. These operations when
    applied to the identity matrix yield the inverse A1. The case
    of a singular matrix is indicated by the value s := 1;
begin comment augment matrix A with identity matrix;
    array a[1:n, 1:2  $\times$  n]; integer i,j;
    for i := 1 step 1 until n do
    for j := 1 step 1 until 2  $\times$  n do
    if j  $\leq$  n then a[i,j] := A[i,j] else
    if j = n+1 then a[i, j] := 1.0 else a[i,j] := 0.0;
    comment begin inversion;
    for i := 1 step 1 until n do
begin integer k,  $\ell$ , ind; j :=  $\ell$  := i; ind := s := 0;
    L1: if a[ $\ell$ ,j] = 0 then
    begin ind := 1; if  $\ell$  < n then begin  $\ell$  :=  $\ell$  + 1;
    go to L1 end
    else begin s := 1; go to L2 end
    end;
    if ind = 1 then for k := 1 step 1 until 2  $\times$  n do
    begin real temp;
    temp := a[ $\ell$ ,k];
    a[ $\ell$ ,k] := a[i,k];
    a[i,k] := temp end k loop;
    for k := j step 1 until 2  $\times$  n do
    a[i,k] := a[i,k] / a[i,j];
    for  $\ell$  := 1 step 1 until n do
    if  $\ell \neq i$  then for k := 1 step 1 until 2  $\times$  n do
    a[ $\ell$ ,k] := a[ $\ell$ ,k] - a[i,k]  $\times$  a[i,j];
    end i loop;
    for i := 1 step 1 until n do
    for j := 1 step 1 until n do
    A1[i,j] := a[i,n+j];
    L2: end of procedure

```

ALGORITHM 43

CROUT WITH PIVOTING II

HENRY C. THACHER, JR.*

Argonne National Laboratory, Argonne, Illinois

```

real procedure INNERPRODUCT (u,v) index : (k) start : (s)
    finish : (f);
value s, f; integer k, s, f; real u, v;
comment INNERPRODUCT forms the sum of  $u(k) \times v(k)$  for
     $k = s, s+1, \dots, f$ . If  $s > f$ , the value of INNERPRODUCT is
    zero. The substitution of a very accurate inner product procedure
    would make CROUT more accurate;
comment INNERPRODUCT may be declared in the head of
    any block which includes the block in which CROUT is declared.
    It may be used independently for forming the inner
    product of vectors;
begin
    real h;
    h := 0; for k := s step 1 until f do h := h + u  $\times$  v;
    INNERPRODUCT := h
end INNERPRODUCT;

```

procedure CROUT II (A, b, n, y, pivot, det, repeat)
comment This procedure is a revision of Algorithm 16, Crout With Pivoting by George E. Forsythe, *Comm. ACM* 3, (1960) 507-8. In addition to modifications to improve the running of the program, and to conform to proper usage, it provides for the computation of the determinant, det, of the matrix A. The solution is obtained by Crout's method with row interchanges, as formulated in reference [1], for solving $Ay = b$ and transforming the augmented matrix $[A \ b]$ into its triangular decomposition LU with all $L(k,k) = 1$. If A is singular we exit to 'singular,' a nonlocal label. pivot (k) becomes the current row index of the pivot element in the k-th column. Thus enough information is preserved for the procedure to process a new right-hand side without repeating the triangularization, if the boolean parameter repeat is true. The accuracy obtainable from CROUT would be much increased by calling CROUT with a more accurate inner product procedure than INNERPRODUCT.

The contributions of Michael F. Lipp and George E. Forsythe by prepublication review and pointing out several errors are gratefully acknowledged;

comment Nonlocal identifiers appearing in this procedure are: (1) The nonlocal label 'singular', to which the procedure exits if $\det A = 0$, and (2) the real procedure 'INNERPRODUCT' given above;

value n; array A, b, y; integer n; integer array pivot; real det; Boolean repeat;

begin

integer k, i, j, imax, p; real TEMP, quot;
det := 1; if repeat **then go to** 6;

for k := 1 **step** 1 **until** n **do**

1: **begin**

TEMP := 0;

for i := k **step** 1 **until** n **do**

2: **begin**

A[i,k] := A[i,k] - INNERPRODUCT (A[i,p], A[p,k],
p, 1, k-1);

if abs(A[i,k]) > TEMP **then**

3: **begin**

TEMP := abs(A[i, k]); imax := i
end 3

end 2;

pivot [k] := imax;

comment We have found that A[imax, k] is the largest pivot in column k. Now we interchange rows k and imax;

if imax $\neq$ k **then**

4: **begin** det := - det; for j := 1 **step** 1 **until** n **do**

5: **begin**

TEMP := A[k,j]; A[k,j] := A[imax, j]; A[imax, j]
:= TEMP

end 5;

TEMP := b[k]; b[k] := b[imax]; b[imax] := TEMP
end 4;

comment The row interchange is done. We proceed to the elimination;

if A[k,k] = 0 **then go to** singular;

quot := 1.0/A[k,k];

for i := k+1 **step** 1 **until** n **do**

A[i,k] := quot $\times$ A[i,k];

for j := k+1 **step** 1 **until** n **do**

A[k,j] := A[k,j] - INNERPRODUCT (A[k,p],
A[p,j], p, 1, k-1);

b[k] := b[k] - INNERPRODUCT (A[k,p], b[p],
p, 1, k-1)

end 1; **go to** 7;

comment The triangular decomposition is now finished, and we skip to the back substitution;

6: **begin comment** This section is used when the formal parameter repeat is **true**, indicating that the matrix A

has previously been decomposed into triangular form by CROUT II, with row interchanges specified by pivot, and that it is desired to solve the linear system with a new vector b, without repeating the triangularization;

for k := 1 **step** 1 **until** n **do**

begin

TEMP := b[pivot[k]]; b[pivot[k]] := b[k]; b[k] :=
TEMP; b[k] := b[k] - INNERPRODUCT
(A[k, p], b[p], p, 1, k-1) **end**;

end 6;

7: for k := n **step** - 1 **until** 1 **do**

8: **begin** if $\neg$ repeat **then** det := A[k,k] $\times$ det;

y[k] := (b[k] - INNERPRODUCT (A[k,p], y[p], p,
k+1, n)/A[k,k]

end 8;

end CROUT II;

REFERENCE:

- (1) J. H. WILKINSON, Theory and practice in linear systems. In John W. Carr III (editor), Application of Advanced Numerical Analysis to Digital Computers, pp. 43-100 (Lectures given at the University of Michigan, Summer 1958, College of Engineering, Engineering Summer Conferences, Ann Arbor, Michigan [1959]).

* Work supported by the U. S. Atomic Energy Commission.

ALGORITHM 44

BESSEL FUNCTIONS COMPUTED RECURSIVELY

MARIA E. WOJCICKI

RCA Digital Computation and Simulation Group,
Moorestown, New Jersey

procedure Bessfr(N, FX, LX, Z) Result: (J, Y);

value LX, FX, N;

real FX, LX, Z; real array J, Y; integer N;

comment Bessel Functions of the first and second kind, $J_P(X)$ and $Y_P(X)$, integral order P, are computed by recursion for values of X, $FX \leq X \leq LX$, in steps of Z. The functions are computed for values of P, $0 \leq P \leq N$. M[SUB], the initial value of P being chosen according to formulae in Erdelyi's *Asymptotic Expansions*. The computed values of $J_P(X)$ and $Y_P(X)$ are stored as column vectors for constant argument in matrices J, Y of dimension (N+1) by entier ((LX - FX)/Z + 1);
begin real PI, X, GAMMA, PAR, LAMDA, SUM, SUM1;

integer P, SUB, MAXSUB;

PI := 3.14159265;

GAMMA := .57721566;

PAR := 63.0 - 1.5 $\times \ln$ (2 $\times$ PI);

MAXSUB := entier ((LX - FX)/Z);

begin real array JHAT [0:N, 0:MAXSUB];

integer array M[0:MAXSUB];

SUB := 0;

for X := FX **step** Z **until** LX **do**

begin if (X > 0) $\wedge$ (X < 10) **then** M [SUB] := 2 $\times$ entier (X) + 9
else

begin real ALOG;

ALOG := (PAR - 1.5 $\times \ln$ (X))/X;

M [SUB] := entier (X $\times$ (exp (ALOG) + exp
(-ALOG))/2) **end**;

if N > M [SUB] **then**

begin for P := M [SUB] + 1 **step** 1 **until** N **do**

J [P, SUB] := 0 **end**;

JHAT [M [SUB], SUB] := 10 $\uparrow$ (-9);

comment Having set the uppermost $J_P(X)$ to a very small number we are now going to compute all the $J_P(X)$ down to

```

P := 0;
for P := M [SUB] step -1 until 1 do
  JHAT [P-1, SUB] := 2 × P/X × JHAT [P, SUB] - JHAT
    [P+1, SUB];
SUM := SUM1 := 0;
for P := 2 step 2 until (M [SUB] ÷ 2) do
  SUM := SUM + JHAT [P, SUB];
LAMDA := JHAT [0, SUB] + 2 × SUM;
for P := 0 step 1 until N do
  J [P, SUB] := JHAT [P, SUB] /LAMDA;
comment JP(X) have been computed by use of  $\hat{J}_P(X)$ ;
for P := 2 step 2 until (M [SUB] ÷ 2) do
  SUM1 := SUM1 + (-1) × (-1) ↑ P ÷ J [2 × P, SUB]
    /2/P;
Y [0, SUB] := 2/PI × (J [0, SUB] × (GAMMA +  $\ell_n(X/2)$ )
  + 4 × SUM1);
for P := 0 step 1 until (M[SUB]-1) do
  Y [P+1, SUB] := (-2/PI/P + J [P+1, SUB] × Y [P,
    SUB])/J [P, SUB];
SUB := SUB + 1 end end end

```

ALGORITHM 45

INTEREST

PETER Z. INGERMAN

University of Pennsylvania, Philadelphia, Pa.

procedure monpay (i, B, L, t, k, m, tol, goof)
comment This procedure calculates the periodic payment
 necessary to retire a loan when the interest rate on the loan
 varies (possibly from period to period) as a function of the as-
 yet-unpaid principal.

The formal parameters are: i, array identifier for the vector
 of interest rates; -B, array identifier for the minimum amounts
 at which the corresponding i applies; -L, the amount to be
 borrowed; -t, the number of periods for which the loan is to
 be taken out; -k, the number of different interest rates (and
 upper limit for vectors i and B); -m, the desired periodic pay-
 ment; -tol, the allowable deviation of m from some ideal;
 and goof, the error exit to use if convergence fails. The only
 output parameter is m. For further discussion, see *Comm.*
ACM 3 (Oct. 1960), 542;

```

begin array h, S [1:k, 1:t], M, X [1:k];
integer array T, a, b [1:k];
integer p, q, r, sa, sb, I, ib, mb, nb;
comment This section sets up the procedure;
for p := 1 step 1 until k do
  begin for q := 1 step 1 until t do
    begin hp,q := ipq;
      Sp,q := (hp,q - 1)/(ip - 1) end;
    if p = 1 then Xp := 0 else Xp := Bp × (ip-1 - ip);
    Mp := L × (hp,t/Sp,t) end;
  sa := sb := ib := mb := 0; nb := t;
  for p := 1 step 1 until k do
    begin ap := entier (Bp+1/Mp+1 + 0.5) - sa;
      sa := sa + ap;
      Tp := bp := entier (Bp+1/Mp - 0.5) - sb;
      sb := sb + bp;
      if bp > mb then
        begin ib := p; nb := nb - mb; mb := bp end
      else nb := nb - bp end;

```

T_{ib} := nb;

I := 1;

for p := 1 step 1 until k do

I := I × (a_p - b_p + 1);

comment Having counted the number of possible iterations
 and established a set of trial values for the T_n's, a trial m is
 found;

D := 1; E := F := 0;

newm: for p := 1 step 1 until k do

begin D := D × h_{p,T_p};

u := 1;

if p ≠ 1 then for q := 1 step 1 until p - 1

do u := u × h_{q,T_q};

E := E + S_{p,T_p} × u;

v := 0;

if p ≠ 1 then for r := 1 step 1 until p

do v := v + X_r;

F := F + u × v end;

m := (L × D + F)/E;

comment Now find out whether m is good enough

q := 1; F := D := 0;

for p := 1 step 1 until t do

begin get F: F := (D + m - E)/(1 + i_q);

if B_{q+1} ≥ F then D := F else q := q + 1;

if D ≠ F go to get F end;

if abs (D - L) ≤ tol then go to exit;

comment If not within tolerance, adjust T_n's and try
 again;

p := 0;

redo: p := p + 1;

if p ≠ ib then

begin if T_p ≥ a_p then

begin T_{ib} := T_{ib} + T_p - b_p

T_p := b_p end end

else begin

T_p := T_p + 1;

T_{ib} := T_{ib} - 1;

p := k end;

if p = k then I := I - 1 else go to redo;

go to if I > 0 then newm else goof;

exit: end monpay;

ALGORITHM 46

EXPONENTIAL OF A COMPLEX NUMBER

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

procedure EXPC (a, b, c, d); **value** a, b; **real** a, b, c, d;
comment This procedure computes the number, c+di, which
 is equal to e^(a+ib);

begin c := exp (a);

d := c × sin (b);

c := c × cos (b)

end EXPC;

ALGORITHM 47

ASSOCIATED LEGENDRE FUNCTIONS OF THE FIRST KIND FOR REAL OR IMAGINARY ARGUMENTS

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

procedure LEGENDREA (m, n, x, r); **value** m, n, x, r;
integer m, n; **real** x, r;

comment This procedure computes any P_n^m(x) or P_n^m(ix) for
 n an integer less than 20 and m an integer no larger than n.
 The upper limit of 20 was taken because (42)! is larger than
 10⁴⁹. Using a modification of this procedure values up to n=35
 have been calculated. If P_n^m(x) is desired, r is set to zero. If
 r is nonzero, P_n^m(ix) is computed;

begin

```

integer i, j; array Gamma [1:41];
real p, z, w, y;
if n = 0 then
  begin p := 1;
  go to gate end;
if n < m then
  begin p := 0;
  go to gate end;
z := 1; w := z;
if n=m then go to main;
for i := 1 step 1 until n-m do
  z := x × z;
main: Gamma [1] := 1;
for i := 2 step 1 until n+n+1 do
  begin Gamma [i] := w × Gamma [i-1];
  w := w+1 end;
w := 1; y := w/(x × x);
if r=0 then
  begin y := -y;
  w := -w end;
if x=0 then
  begin i := (n-m)/2;
  if (i+i) ≠ (n-m) then
    begin p := 0;
    go to gate end;
  p := Gamma [m+n+1]/(Gamma [i+1] × Gamma
    [m+i+1]);
  go to last end;
j := 3; p := 0;
for i := 1 step 1 until l2 do
  begin if (n-m+2)/2 < i then go to last end;
  p := p + Gamma [n+n-i-i+3] × z/(Gamma
    [i] × Gamma [n-i+2] × Gamma [n-i-i-
    m+j]);
  z := z × y end;
last: z := 1;
for i := step 1 until n do
  z := z+z;
p := p/z;
if r ≠ 0 then
  begin i := n-n/4;
  if 1 < i then
    p := -p end;
if m = 0 then go to gate;
j := m/2; z := abs(w+x × x);
if m ≠ (j+j) then
  begin z := sqrt (z);
  j := m end;
for i := step 1 until j do
  p := p × z;
gate: LEGENDREA := p
end LEGENDREA;

```

ALGORITHM 48

LOGARITHM OF A COMPLEX NUMBER

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

procedure LOGC(a, b, c, d); value a, b; real a, b, c, d;
comment This procedure computes the number, c+di, which
is equal to loge(a+bi);
begin c := sqrt (a × a + b × b);
d := arctan (b/a);
c := log (c);
if a < 0 then d := d+3.1415927
end LOGC;

```

ALGORITHM 49

SPHERICAL NEUMANN FUNCTION

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

real procedure SPHBEN (r,x); value r,x; real r,x;
comment This procedure computes the spherical Neumann
function (π/2x)1/2Nr+1/2(x). Infinity is represented by 1047;
begin real z, g, t;
if x=0 then
  begin s := 10 ↑ 47;
  go to gate
end;
s := -cos (x)/x;
if r = 0 then
  go to gate;
t := sin (x)/x;
for g := 1 step 1 until r do
  begin z := s;
  s := s × (g+g-1)/(x-t);
  t := z
end;
gate: SPHBEN := s
end SPHBEN;

```

ALGORITHM 50

INVERSE OF A FINITE SEGMENT OF THE HILBERT MATRIX

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

procedure INVHILBERT (n,S); value n; real n;
real array S;
comment This procedure computes the elements of the inverse
of an n × n finite segment of the Hilbert matrix and stores them
in the array S;
begin real i, j, k;
S[1, 1] = n × n;
for i := 2 step 1 until n do
  begin
    S[i, i] := (n+i-1) × (n-i+1)/((i-1) × (i-1));
    S[i, i] := S[i-1, i-1] × S[i, i] × S[i, i]
  end;
for i := 1 step 1 until n-1 do
  begin
    for j := i+1 step 1 until n do
      begin
        k := j-1;
        S[i, j] := -S[i, k] × (n+k) × (n-k)/(k × k)
      end
    end;
for i := 2 step 1 until n do
  begin S[i, i] := S[i, i]/(i+i-1);
  for j := 1 step 1 until i-1 do
    begin S[j, i] := S(j, 1)/(i+j-1);
    S[i, j] := S[j, i]
    end
  end
end INVHILBERT;

```

ALGORITHM 51

ADJUST INVERSE OF A MATRIX WHEN AN ELEMENT IS PERTURBED

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

procedure ADJUST (n, d, i, j, A, B); value i, j, n, d;
    integer i, j, n; real d; real array A, B;
comment If the  $n \times n$  matrix  $A=M^{-1}$  and a change, d, is made
    in the i, j-th element of M this procedure will calculate the
    corrected matrix for  $M^{-1}$  by adjusting matrix A. The adjusted
    matrix is stored in B;
begin    integer r, s;
        real t;
         $t := d/(A[j, i] \times d+1)$ ;
        for r := 1 step 1 until n do
            begin for s := 1 step 1 until n do
                 $B[r, s] = A[r, s] - t \times A[r, i] \times A[j, s]$  end
            end
        end ADJUST

```

ALGORITHM 52

A SET OF TEST MATRICES

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

procedure TESTMATRIX (n,A); value n; integer n;
    real array A;
comment This procedure places in A an  $n \times n$  matrix whose
    inverse and eigenvalues are known. The n-th row and the n-th
    column of the inverse are the set: 1, 2, 3, ..., n. The matrix
    formed by deleting the n-th row and the n-th column of the
    inverse is the identity matrix of order n-1;
begin    integer i, j;
        real t, c, d, f;
         $c := t \times (t+1) \times (t+t-5)/6$ ;
         $d := 1/c$ ;
         $A[n, n] := -d$ ;
        for i := 1 step 1 until n-1 do
            begin f := i;
                 $A[i, n] := d \times f$ ;
                 $A[n, i] := A[i, n]$ ;
                 $A[i, i] := d \times (c-f \times f)$ ;
                for j := 1 step 1 until i-1 do
                    begin t := j;
                         $A[i, j] := -d \times f \times t$ ;
                         $A[j, i] := A[i, j]$ 
                    end
                end
            end
        end TESTMATRIX;

```

ALGORITHM 53

NTH ROOTS OF A COMPLEX NUMBER

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

procedure NTHROOT (n, r, u, REAL, UNREAL); value
    n, r, u; integer n;
    real r, u; real array REAL, UNREAL;
comment This procedure computes the n roots of the equation
 $x^n = r+ui$ . The real parts of the roots are stored in the vector
    REAL [ ]. The imaginary parts are stored in the corresponding
    locations in the vector UNREAL [ ];
begin    integer n1, n2; real en, th, s, th 1;
         $REAL[n] := 0$ ;

```

```

    en := 1/n;
    if u=0 then
        begin s := (abs(r))  $\uparrow$  en;
            th := 0,
            go to main end;
    if r=0 then
        begin s := (abs(u))  $\uparrow$  en;
            th := 1.5707963;
            if u < 0 then
                th := -th
            go to main end;
        s := (r  $\times$  r+u  $\times$  u)  $\uparrow$  (en/2);
        th := arctan (u/r);
main:    if r < 0 then
            th := th + 3.1415926;
        th := en  $\times$  th;
        th1 := 6.2831853  $\times$  en;
        for n2 := 1 step 1 until n do
            begin REAL [n2] := s  $\times$  cos (th);
                UNREAL [n2] := s  $\times$  sin (th);
                th = th+th 1 end
        end NTHROOT;

```

ALGORITHM 54

GAMMA FUNCTION FOR RANGE 1 TO 2

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

real procedure Q(x); value x; real x,
comment This procedure computes  $\Gamma(x)$  for  $1 \leq x \leq 2$ . This is
    a reference procedure for the more general gamma function
    procedure.  $\Gamma(x) = Q(x-1)$ ;
begin     $Q := ((((((0.035868343 \times x - 0.19352782) \times x$ 
        + 0.48219939)  $\times x - 0.75670408) \times x$ 
        + 0.91820686)  $\times x - 0.89705694) \times x$ 
        + 0.98820589)  $\times x - 0.57719165) \times x + 1.0$ 
    end    Q;

```

ALGORITHM 55

COMPLETE ELLIPTIC INTEGRAL OF THE FIRST KIND

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

real procedure ELLIPTIC 1(k); value k; real k;
comment This procedure computes the elliptic integral of the
    first kind  $K(k, \pi/2)$ ;
begin    real t;
         $t := 1-k \times k$ ;
        ELLIPTIC 1 := (((0.032024666  $\times t +$ 
            0.054555509)  $\times t$ 
            + 0.097932891)  $\times t + 1.3862944$ )
            - (((0.010944912  $\times t + 0.060118519) \times t$ 
            + 0.12475074)  $\times t + 0.5) \times \log (t)$ 
    end    ELLIPTIC 1;

```

ALGORITHM 56

COMPLETE ELLIPTIC INTEGRAL OF THE SECOND KIND

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

real procedure ELLIPTIC 2(k); value k; real k;

```

```

comment This procedure computes the elliptic integral of the
second kind  $E(k, \pi/2)$ ;
begin      real t;
           t := 1 - k  $\times$  k;
           ELLIPTIC 2 := (((0.040905094  $\times$  t +
                        0.085099193)  $\times$  t
                        + 0.44479204)  $\times$  t + 1.0 - (((0.01382999  $\times$  t
                        + 0.08150224)  $\times$  t + 0.24969795)  $\times$  t)  $\times$  log (t)
end      ELLIPTIC 2;

```

ALGORITHM 57

BER OR BEI FUNCTION

JOHN R. HERNDON

Stanford Research Institute, Menlo Park, California

```

real procedure BERBEI (r, z); value r, z; real r, z;
comment This procedure computes ber(z) if r is set equal to
zero. bei(z) is produced if r equals 1.0;
begin

```

```

    real s, k, c, f, t;
    if r = 0 then
        s := 1
    else
        s := (z  $\times$  z)/4;
        k := s;
        f := z  $\times$  z;
        f := f  $\times$  f;
        for c := 2 step 2 until 100 do
            begin
                if s = s + k then
                    go to gate;
                t := (c+r)  $\times$  (c+r-1);
                k := -0.0625  $\times$  k  $\times$  f/(t  $\times$  t);
                s := s+k end;

```

```

gate: BERBEI := s
end BERBEI;

```

REMARK

ON FREQUENTLY OCCURRING ERRORS IN ALGOL-60 PROGRAMS

W. BÖRSCH-SUPAN

National Bureau of Standards, Washington 25, D. C.

There are some features in the syntax of ALGOL 60 which are often neglected by people writing algorithms. This fact may be due in part to the lack of redundancy in the ALGOL 60 report, in part to some confusion with other languages like ALGOL 58 or FORTRAN. Therefore it may be worthwhile to mention these frequently occurring errors in order to avoid them in the future.

There is some confusion between specifications and declarations in procedures. Specifications are given for the formal parameters of a procedure, i.e. for the quantities that connect the procedure to the main program. Declarations are given for the variables local to the procedure body, i.e. for the subsidiary quantities not accessible to the main program. Specifications may be omitted in case of formal parameters called by name, but they may be helpful to the user of the procedure and to the compiler. Specifications of arrays must not contain information about the dimensionality and subscript bounds. Nevertheless this information often is needed by the user of the subroutine. Therefore it should be included in a comment, where a notation similar to declarations could be used.

When the delimiters “**end**” and “;” come together the fol-

lowing should be kept in mind: The “;” separates subsequent statements. The “**begin**” and “**end**” tie together a sequence of statements to form a compound statement. Therefore, as a rule, “**end**” or a string of several “**end**” must be followed by a “;”, if another statement follows. In the string “; **end**” the “;” always can be dropped since it only introduces a dummy statement without label between “;” and “**end**”.

An integer followed by a decimal point is no ALGOL-60 number. Write “1” or “1.0” instead of “1.”.

According to the paragraph 2.3 of the ALGOL-60-report, the comment must not be given before the procedure heading. The reason is very formal: The report declares what “**comment**” preceded by “;” or “**begin**” means, but “**comment**” preceded by nothing is undefined.

There is no rule about what the comment of a procedure should include. But, I think that users of procedures would like writers of procedures to include all the information necessary to ensure a correct use of procedures without reading the procedure body.

Two other things may be helpful to the reader of algorithms: Using simple and multiple indentation in a systematic manner may clarify the nesting of statements quite a bit. In a similar way one may improve the readability by putting notes after the delimiter “**end**” which indicate the delimiter “**begin**” to which they belong.

CERTIFICATION OF ALGORITHM 3

SOLUTION OF POLYNOMIAL EQUATION BY

BARSTOW-HITCHCOCK (A. A. Grau, *Comm. ACM* Feb. 1960)

JOHN HERNDON

Stanford Research Institute, Menlo Park, California

Bairstow was transliterated into BALGOL and tested on the Burroughs 220. The corrections supplied by Thatcher, *Comm. ACM*, June 1960, were incorporated. Results were correct for equations for which the method is suitable. $x^4 - 16 = 0$ is one of those which gave nonsensical results. Seven-digit results were obtained for 12 test equations, one of which was $x^6 - 2x^5 + 2x^4 + x^3 + 6x^2 - 6x + 8 = 0$.

CERTIFICATION OF ALGORITHM 10

CHEBYSCHEV POLYNOMIAL $T_n(x)$ (Galler, *Comm. ACM*, June, 1960)

JOHN HERNDON

Stanford Research Institute, Menlo Park, California

When transliterated into BALGOL and tested on the Burroughs 220, Ch(n, x) gave better than 7-digit accuracy for $n = 0, 1, 4, 8$ and $x = .01, .2, .7$. It gave answers when $x > 1$ which corresponded to the value of the series with x substituted.

CERTIFICATION OF ALGORITHM 13

LEGENDRE POLYNOMIAL $P_n(x)$ (Galler, *Comm. ACM*, June 1960)

JOHN HERNDON

Stanford Research Institute, Menlo Park, California

When transliterated into BALGOL and tested on the Burroughs 220, Le(n, x) gave 7-digit accuracy for $n = 0, 1, 4, 9$ and $X = .01, .2, .7, 1.9, 5.0$.

CERTIFICATION OF ALGORITHM 20
 REAL EXPONENTIAL INTEGRAL (S. Peavy, *Comm. ACM*, Oct. 1960)

WILLIAM J. ALEXANDER* and HENRY C. THACHER, JR.*
 Argonne National Laboratory, Argonne, Illinois

Expint (x) was programmed for the LGP-30 computer, using both a 7S floating-point compiler (ACT III) and an 8S floating-point interpretive code (24.2). Constants given to more than 7S (or to 8S for the 24.2 program) were rounded to 7S (or 8S).

After changing the constant .005519968 to .05519968, both programs gave acceptable accuracy over the range tested.

The 8S (24.2) program was compared with the 9D values given for $-E_1(-x)$ in Mathematical Tables Project, *Tables of Sine, Cosine, and Exponential Integrals, Volume II* (1940) for the set of values $x = 0.1(0.1)1.0(1.0)10.0$. The largest discrepancy found was -16×10^{-8} for $x = 0.1$. For x greater than 1, all values tested were good to 8S.

For computing real values of the exponential integral, this algorithm is much faster than EKZ (Algorithm 13). For $x < 1$, the ratio of speeds was of the order of 20.

* Work supported by the U.S. Atomic Energy Commission.

CERTIFICATION OF ALGORITHM 43
 CROUT II (Henry C. Thacher, Jr., *Comm. ACM*, 1960)
 HENRY C. THACHER, JR.*
 Argonne National Laboratory, Argonne, Illinois

CROUT II was coded by hand for the Royal Precision LGP-30 computer, using a 28-bit mantissa floating point interpretive system (24.2 modified).

The program was tested against the linear system:

$$A = \begin{bmatrix} 12.1719 & 27.3941 & 1.9827 & 7.3757 \\ 8.1163 & 23.3385 & 9.8397 & 4.9474 \\ 3.0706 & 13.5434 & 15.5973 & 7.5172 \\ 3.0581 & 3.1510 & 6.9841 & 13.1984 \end{bmatrix} \quad b = \begin{bmatrix} 6.6355 \\ 6.1304 \\ 4.6921 \\ 2.5393 \end{bmatrix}$$

with the following results:

$$A' = \begin{bmatrix} 12.171900 & 27.394100 & 1.9827000 & 7.3756999 \\ 0.25226957 & 6.6327021 & 15.097125 & 5.6565352 \\ 0.25124262 & -0.56260107 & 14.979620 & 14.527683 \\ 0.66680633 & 0.76468695 & -0.20207132 & -1.3606142 \end{bmatrix}$$

$$b' = \begin{bmatrix} 6.6354999 \\ 3.0181653 \\ 2.5702026 \\ -0.082780734 \end{bmatrix} \quad \text{pivot} = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 4 \end{bmatrix} \quad y = \begin{bmatrix} 0.15929120 \\ 0.14691771 \\ 0.11257482 \\ 0.060840712 \end{bmatrix}$$

$\det = -1645.4499$. All elements of $Ab - y$ were less than 10^{-7} in magnitude. Identical results were obtained with the same b , and repeat true. With the same b and the last row vector of A replaced by (19.1927, 33.4409, 25.1298, 5.2811), i.e. $A_4, j = A_1 j + 2A_2, j - 3A_3, j$, the results were:
 $\det = 0.10924352 \times 10^{-3}$,
 $y = (0.29214425 \times 10^8, -0.12131172 \times 10^8, 0.72411923 \times 10^7, -0.51018392 \times 10^7)$

Failure to recognize this singular matrix is due to roundoff, either in the data input or in the calculation.

* Work supported by the U.S. Atomic Energy Commission.

Standards

Further Survey of Punched Card Codes

H. McG. Ross, *Ferranti Ltd., London*

The valuable "Survey of Punched Card Codes" prepared by Smith and Williams (*Comm. ACM* 3, Dec. 1960, 638) unfortunately omits the card codes of European equipment, other than IBM. These are presented in the table on page 181. This information has been extracted from a Ferranti publication, "Collected Information on Punched Card Codes" (List CS 266) and has been set out in much the same way as the table by Smith and Williams.

A valuable step forward has been made by the British Standards Institution in publishing Standard 3174, "Alpha-Numeric Punching Codes for Data Processing Cards". As well as decimal numbers and letters, this Standard also gives single-column punching codes for pence, for shillings, for months, and for days within the month, etc.

In the table, the card rows are identified by 12, 11, 0, 1, ..., 9, from top to bottom; it should be noted, however, that modern practice in Britain prefers the top row to be numbered 10. Where nothing is punched in a zone or position the symbol $\bar{b}$ is used. The abbreviation Sp is used for the space obtained in a printer from an entirely blank column. Letter O is shown with a dot in it, to distinguish it from zero.

The table gives examples of the treatment of days, months, pence, etc., particularly when a single column is used, often with two holes in it. However, even within one type of code, 10 and 11 may be reversed.

The "old Hollerith" 4-zone code, which is widely used, is designed to "cycle" within the zones; this brings the letters into alphabetical order.

The asterisks refer to the Bull codes in which rows 7, 8, and 9 are used to punch the zones, in place of 12, 11, and 0 respectively. Another special optional feature is the "mechanical zero" punched in row 12; this is for left-hand zeros, and is treated as zero by the accounting machine but does not print.