ALGORITHM 61
PROCEDURES FOR RANGE ARITHMETIC
ALLAN GIBB*
University of Alberta, Calgary, Alberta, Canada

begin
procedure RANGESUM (a, b, c, d, e, f);
real a, b, c, d, e, f;
comment The term “range number” was used by P. S. Dwyer,
Linear Computations (Wiley, 1951). Machine procedures for
range arithmetic were developed about 1958 by Ramon Moore,
“Automatic Error Analysis in Digital Computation,” LMSD
Report 48421, 28 Jan. 1959, Lockheed Missiles and Space Divi-
sion, Palo Alto, California, 59 pp. If a _< x -< b and c ~ y ~ d,
then RANGESUM yields an interval [e, f] such that e =< (x + y)
f. Because of machine operation (truncation or rounding) the
machine sums a + c and b + d may not provide safe end-points
of the output interval. Thus RANGESUM requires a non-local
real procedure ADJUSTSUM which will compensate for the
machine arithmetic. The body of ADJUSTSUM will be de-
pendent upon the type of machine for which it is written and so
is not given here. (An example, however, appears below.) It
is assumed that ADJUSTSUM has as parameters real v and w,
and integer i, and is accompanied by a non-local real procedure
CORRECTION which gives an upper bound to the magnitude
of the error involved in the machine representation of a number.
The output ADJUSTSUM provides the left end-point of the
output interval of RANGESUM when ADJUSTSUM is called
with i = --1, and the right end-point when called with i = 1.
The procedures RANGESUB, RANGEMPY, and RANGEDVD
provide for the remaining fundamental operations in range
arithmetic. RANGESQR gives an interval within which the
square of a range number must lie. RNGSUMC, RNGSUBC,
RNGMPYC and RNGDVDC provide for range arithmetic with
complex range arguments, i.e. the real and imaginary parts
are range numbers;

2: begin
if c ~ 0 then
3: begin
 e := a × c; f := b × d; go to 8
 end 3;
 e := b × c;
if d ~ 0 then
4: begin
f := b × d; go to 8
 end 4;
f := a × d; go to 8
end 2;
if b ~ 0 then
6: begin
if d ~ 0 then
begin
 e := MIN(a × d, b × c);
 e := MAX(a × c, b × d); go to 8
 end 6;
 e := b × c; f := a × c; go to 8
 end 5;
f := a × c;
if d ~ 0 then
7: begin
 e := b × d; go to 8
 end 7;
e := a × d;
8: e := ADJUSTPROD (e, −1);
f := ADJUSTPROD (f, 1)
end RANGEMPY;
procedure RANGEDVD (a, b, c, d, e, f);
real a, b, c, d, e, f;
comment If the range divisor includes zero the program
exists to a non-local label “zerodvsr”. RANGEDVD assumes a
non-local real procedure ADJUSTQUOT which is analogous
(possibly identical) to ADJUSTPROD;

begin
if c ~ 0 ∧ d ~ 0 then go to zerodvsr;
if c ~ 0 then
1: begin
if b ~ 0 then
2: begin
 e := b/d; go to 3
 end 2;
e := b/c;
3: if a ~ 0 then
4: begin
f := a/c; go to 8
 end 4;
f := a/d; go to 8
end 1;
if a ~ 0 then
5: begin
 e := a/c; go to 6
 end 5;
e := a/d;
6: if b ~ 0 then
7: begin
 f := b/c; go to 8
 end 7;
f := b/d;
8: e := ADJUSTQUOT (e, −1); f := ADJUSTQUOT (f, 1)
end RANGEDVD;
procedure RANGESQR (a, b, e, f);
real a, b, e, f;
comment ADJUSTPROD is a non-local procedure;
begin
if a ~ 0 then
real number is expressible to s significant figures, base b. Limitations on the machine or requirements of the user will limit the range of p to $b^m \leq |p| < b^{m+1}$ for some integers m and n. Appropriate integers must replace $s$, $b$, $m$ and $n$ below. Signal is a non-local label. The procedures of the example would be included in the same block as the range procedures above;

begin
if $b < 0$ then
begin
  e := b \times b;  f := a \times a;  go to 3
end 2;
  e := 0;  m := \text{MAX} (-a,b);  f := m \times m;  go to 3
end 1;
  e := a \times a;  f := b \times b;
3: ADJUSTPROD (e, -1);
ADJUSTPROD (f, 1)
end RANGESQR;
procedure RNGSUMC (aL, aR, bL, bU, eL, fL, fU);
real aL, aR, bL, bU, eL, fL, fU;
comment RNGSUMC is a non-local procedure;
begin
RANGESUM (aL, aR, eL, fL, fU);
RANGESUM (bL, bU, eL, fL, fU)
end RNGSUMC;
procedure RNGSUBC (aL, aR, bL, bU, eL, eR, fL, fU);
real aL, aR, bL, bU, eL, eR, fL, fU;
comment RNGSUBC is a non-local procedure;
begin
RANGESUM (aL, aR, eL, fL, fU);
RANGESUM (bL, bU, eL, fL, fU)
end RNGSUBC;
procedure RNGMPYC (aL, aR, bL, bU, cL, cR, dL, dU, eL, eR, fL, fU);
comment RNGMPYC is a non-local procedure;
begin
RANGESUM (aL, aR, cL, cR, L1, R1);
RANGESUM (bL, bU, eL, fL, fU)
end RNGMPYC;
procedure RNGDVDC (aL, aR, bL, bU, cL, cR, dL, dU, eL, eR, fL, fU);
comment RNGDVDC is a non-local procedure;
begin
RANGESUM (L1, R1, L2, R2, L3, R3, L4, R4);
RANGESUM (L3, R3, L4, R4, L5, R5);
RANGEMPY (L1, R1, L5, R5, eL, eR);
RANGEMPY (L2, R2, L5, R5, fL, fU)
end RNGDVDC;

EXAMPLE

real procedure CORRECTION (p);  real p;
comment CORRECTION and the procedures below are intended for use with single-precision normalized floating-point arithmetic for machines in which the mantissa of a floating-point number is expressible to s significant figures, base b. Limitations on the machine or requirements of the user will limit the range of p to $b^m \leq |p| < b^{m+1}$ for some integers m and n. Appropriate integers must replace $s$, $b$, $m$ and n below. Signal is a non-local label. The procedures of the example would be included in the same block as the range procedures above;

begin
  integer w;
  for $w := m$ step 1 until $n$ do
1: begin
if $(b \uparrow w \leq |p|) \land (|p| < b \uparrow (w + 1))$ then
2: begin
  CORRECTION := $b \uparrow (w + 1 - s)$;  go to exit
end 2
end 1;
  go to signal;
exit: end CORRECTION;
real procedure ADJUSTSUM (w, v, i);  integer i;
real w, v;
comment ADJUSTSUM exemplifies a possible procedure for use with machines which, when operating in floating point addition, simply shift out any lower order digits that may not be used. No attempt is made here to examine the possibility that every digit that is dropped is zero. CORRECTION is a non-local real procedure which gives an upper bound to the magnitude of the error involved in the machine representation of a number;

begin
  real r, cw, cr, cv;
  r := w + v;
if $w = 0 \lor v = 0$ then go to 1;
cw := CORRECTION (w);
v := CORRECTION (v);
cr := CORRECTION (r);
if $w = v \land cr \leq cw$ then go to 1;
if $|w| \times |v| \times |r| = -1$ then go to 1;
ADJUSTSUM := $r + i \times \text{MAX} (cw, cr, cv)$;  go to exit;
1: ADJUSTSUM := $r$;
exit: end ADJUSTSUM;
real procedure ADJUSTPROD (p, i);  real p;  integer i;
comment ADJUSTPROD is for machines which truncate when lower order digits are dropped. CORRECTION is a non-local real procedure;

begin
  if $p \times i \leq 0$ then
1: begin
ADJUSTPROD := p;  go to out
end 1;
ADJUSTPROD := $p + i \times \text{CORRECTION} (p)$;
out: end ADJUSTPROD;
comment Although ordinarily rounded arithmetic is preferable to truncated arithmetic, for these range procedures truncated arithmetic leads to closer bounds than rounding does.

* These procedures were written and tested in the Burroughs 220 version of the ALGOL language in the summer of 1960 at Stanford University. The typing and editorial work were done under Office of Naval Research Contract Nonr-225(37). The author wishes to thank Professor George E. Forsythe for encouraging this work and for assistance with the syntax of ALGOL 60.

ALGORITHM 62
A SET OF ASSOCIATE LEGENDRE POLYNOMIALS OF THE SECOND KIND*

JOHN R. HERNDON
Stanford Research Institute, Menlo Park, California

comment This procedure places a set of values of $Q_n^m(x)$ in the array $Q[n]$ for values of $n$ from 0 to $n_{\text{max}}$ for a particular value of $m$ and a value of $x$ which is real if $r$ is 0 and is purely imaginary, $ix$, otherwise. $R[n]$ will contain the set of ratios of successive values of $Q$. These ratios may be especially valuable when the $Q_n^m(x)$ of the smallest size is too small as to underflow the machine representation (e.g. $10^{-10}$ if $10^{-8}$ were the smallest representable
number). 9.9 × 10^5 is used to represent infinity. Imaginary values of x may not be negative and real values of x may not be smaller than 1.

Values of Q_{a}(x) may be calculated easily by hypergeometric series if x is not too small nor (n - m) too large. Q_{a}(x) can be computed from an appropriate set of values of P_{a}(x) if x is near 1.0 or ix is near 0. Loss of significant digits occurs for x as small as 1.1 if n is larger than 10. Loss of significant digits is a major difficulty in using finite polynomial representations also if n is larger than m. However, QLEG has been tested in regions of x and n both large and small;

**procedure** QLEG(m, nmax, x, ri, R, Q); value m, nmax, x, ri; real m, nmax, x, ri; real array R, Q;

begin real t, i, n, q0, s;
  n := 20;
  if nmax > 13 then
    n := nmax + 7;
  if ri = 0 then
    begin if m = 0 then
        Q[0] := 0.5 × log((x + 1)/(x - 1))
    for i := 1 step 1 until m do
      begin s := (x + x) × (i - 1) × q0 + (3i - i × i - 2) × q0;
         q0 := Q[0];
         Q[0] := s end end;
  if x = 1 then
    Q[0] := 9.9 × 10^45;
  R[n + 1] := x - sqrt(x × x - 1);
  for i := n step -1 until 1 do
    R[i] := (i + m)/(x + i + 1) × x + (x - 1) × R[i + 1];
  go to the end;
  if m = 0 then
    begin if x < 0.5 then
        Q[0] := -arctan(x) - 1.5707963 else
        Q[0] := arctan(x) - 1.5707963
    for i := 2 step 1 until m do
      begin s := (x + x) × (i - 1) × t × Q[0] + (3i + i × i - 2) × q0;
         q0 := Q[0];
         Q[0] := s end end;
  R[n + 1] := x - sqrt(x × x - 1);
  for i := n step -1 until 1 do
    R[i] := (i + m)/(x + i + 1) × x + (x - 1) × R[i + 1];
  go to the end;
end QLEG;

* This procedure was developed in part under the sponsorship of the Air Force Cambridge Research Center.

**ALGORITHM 63**

**PARTITION**
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

**procedure** partition (A,M,N,J); value M,N; array A; integer M,N,J;

**procedure** quicksort (A,M,N); value M,N; array A; integer M,N;

**procedure** find (A,M,N,K); value M,N,K; array A; integer M,N,K;

**ALGORITHM 64**

**QUICKSORT**
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.

**ALGORITHM 65**

**FIND**
C. A. R. Hoare
Elliott Brothers Ltd., Borehamwood, Hertfordshire, Eng.
begin integer i, j, s; real array v[l:n]; real y, pivot;

JOHN CAFFREY

ALGORITHM 66

INVRS

JOHN CAFFREY

Director of Research, Palo Alto Unified School District, Palo Alto, California

procedure Invrs (t) size : (n); value n; real array t; integer n;

comment Inverts a positive definite symmetric matrix t, of order n, by a simplified variant of the square root method. Replaces the n(n+1)/2 diagonal and superdiagonal elements of t with elements of \( t^{-1} \), leaving subdiagonal elements unchanged. Advantages: only n temporary storage registers are required, no identity matrix is used, no square roots are computed, only n

places the n(n-4-1)/2 diagonal and superdiagonal elements of t

order n, by a simplified variant of the square root method. Re-

begin pivot := 1.0/t[1,1];
for i : = 1 step 1 until n do
begin for j : = i step 1 until n do t[i,j] := --t[i,j]
begin s := 0.0;
end;
end;
end Invrs

begin integer i, j, k, m; real array v[l:n]; real s;

integer array c[l:n];
table: j := n - n; k := n + 1; for i := 1 step 1 until n do begin
j := j + k - i; c[i] := j end;
load: for i := 1 step 1 until n do begin for j := 1 step 1 until n do READ (v[i,j]); m := c[i];
for k := 1 step 1 until n do s[m + k] := v[k] end;
premult: for i := 1 step 1 until n do begin
s := 0.0;
for k := 1 step 1 until n do f[k, j] := v[k] end;
end CRAM

REMARK ON ALGORITHM 53

NTH ROOTS OF A COMPLEX NUMBER (John R. Herndon, Comm. ACM 4, Apr. 1961)

C. W. NESTOR, JR.

Oak Ridge National Laboratory, Oak Ridge, Tennessee

A considerable saving of machine time for \( N \geq 3 \) would result from the use of the recursion formulas for the sine and cosine in place of an entry into a sine-cosine subroutine in the do loop associated with the Nth roots of a complex number. That is, one could use

\[
\sin (n + 1)\theta = \sin n\theta \cos \theta + \cos n\theta \sin \theta
\]

\[
\cos (n + 1)\theta = \cos n\theta \cos \theta - \sin n\theta \sin \theta
\]
at the cost of some additional storage.

We have found this procedure to be very efficient in problems dealing with Fourier analysis, as suggested by G. Goerzel in chapter 24 of Mathematical Methods for Digital Computers.

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 67

CRAM

JOHN CAFFREY

Director of Research, Palo Alto Unified School District, Palo Alto, California

procedure CRAM (n, r, a) Result: (f); value n, r; integer n, r; real array a, f;

comment CRAM stores, via an unspecified input procedure READ, the diagonal and superdiagonal elements of a square symmetric matrix e, of order n, as a pseudo-array of dimension \( 1:n(n + 1)/2 \). READ (u) puts one number into u. Elements \( e[i, j] \) are addressable as \( a[2i - j + 1] \), where \( c = (2n - i)(i - 1)/2 \) and \( c[i + 1] \)

may be found as \( c[i] + n - i \). Since \( c[1] = 0 \), it is simpler to develop a table of the \( e[i] \) by recursion, as shown in the sequence labelled "table". Further manipulation of the elements so stored is illustrated by premultiplying a rectangular matrix f, of order n, r, by the matrix e, replacing the elements of f with the new values, requiring a temporary storage array v of dimension \( 1:n; \)

322 Communications of the ACM