Algorithms

H. J. WEGSTEIN, Editor

ALGORITHM 93
GENERAL ORDER ARITHMETIC

MILLARD H. PERSTEIN
Control Data Corp., Palo Alto, Calif.

```
procedure  arithmetic (a, b, c, op);
integer  a, b, c, op;
comment  This procedure will perform different order arithmetic
  operations with b and c, putting the result in a. The order of the
  operation is given by op. For op = 1 addition is performed. For
  op = 2 multiplication, repeated addition, is done. Beyond these
  the operations are non-commutative. For op = 3 exponentiation,
  repeated multiplication, is done, raising b to the power c. Beyond
  these the question of grouping is important. The innermost
  implied parentheses are at the right. The hyper-exponent is
  always c. For op = 4 tetration, repeated exponentiation, is
  done. For op = 5, 6, 7, etc., the procedure performs pentation,
  hexation, heptation, etc., respectively.
    The routine was originally programmed in FORTRAN for the
  Control Data 160 desk-size computer. The original program
  was limited to tetration because subroutine recursiveness in
  Control Data 160 FORTRAN has been held down to four levels in
  the interests of economy.
    The input parameter, b, c, and op, must be positive integers,
  not zero;
begin own integer d, e, f, drop;
      if op = 1 then
      begin a := b + c;  go to 1
      end if op = 2 then d := 0;
      else d := 1;  e := c; drop := op - 1;
      for f := 1 step 1 until e do
      begin arithmetic (a, b, d, drop);
            d := a
      end;
1:    end arithmetic
```

ALGORITHM 94
COMBINATION

JEROME KURTZBERG
Burroughs Corp., Burroughs Laboratories, Paoli, Pa.

```
procedure  COMBINATION (J, N, K);  value N, K;  integer
  array J;  integer N, K;
comment  This procedure generates the next combination of N
  integers taken K at a time upon being given N, K and the pre-
  vious combination. The K integers in the vector J(1) ··· J(K)
  range in value from 0 to N - 1, and are always monotonically
  strictly increasing with respect to themselves in input and
  output format. If the vector J is set equal to zero, the first
  combination produced is N-K, ···, N-1. That initial combina-
  tion is also produced after 0, 1, ···, N-1, the last value in that
  cycle;
begin  integer  B,  L;
       B := 1;
mainbody:  if J(B)≧B then begin A := J(B) - B-1;
             for L := 1 step 1 until B do J(L) := L + A;
             go to exit end;
           if B = K then go to initiate;
           B := B + 1; go to mainbody;
initiate:  for B := 1 step 1 until K do J(B) := N - K - 1 + B
exit:      end COMBINATION
```

ALGORITHM 95
GENERATION OF PARTITIONS IN PART-COUNT FORM

FRANK STOCKMAL
System Development Corp., Santa Monica, Calif.

```
procedure partgen(c,N,K,G);  integer N,K;  integer array c;
  Boolean G;
comment  This procedure operates on a given partition of the
  positive integer N into parts ≦ K, to produce a consequent
  partition if one exists. Each partition is represented by the
  integers c[1] thru c[K], where c[j] is the number of parts of the
  partition equal to the integer j. If entry is made with G = false,
  procedure ignores the input array c, sets G = true, and pro-
  duces the first partition of N ones. Upon each successive entry
  with G = true, a consequent partition is stored in c[1] thru c[K].
  For N = KX, the final partition is c[K] = X. For N = KX+r,
  1 ≦ r ≦ K-1, final partition is c[K] = X, c[r] = 1. When entry
  is made with array c = final partition, c is left unchanged and G
  is reset to false;
begin integer a,i,j;
      if ¬ G then go to first;
      j := 2;
      a := C[1];
test: if a < j then go to B;
      c[j] := 1 + c[j];
      c[1] := a - j;
zero: for i := 2 step 1 until j - 1
      do c[i] := 0;
      go to EXIT;
B:    if j = K then go to last;
      a := a + j × c[j];
      j := j + 1;
      go to test;
first: G := true;
      c[1] := N;
      j := K + 1;
      go to zero;
last: G := false;
EXIT: end partgen
```

ALGORITHM 96
ANCESTOR

ROBERT W. FLOYD
Armour Research Foundation, Chicago, Ill

```
procedure  ancestor (m, n);  value n;  integer n;  Boolean
  array m;
```

```
comment Initially m [i, j] is true if individual i is a parent of
  individual j. At completion, m [i, j] is true if individual i is an
  ancestor of individual j. That is, at completion m[i, j] is true
  if there are k, l, etc. such that initially m[i, k], m[k, l], ···, m[p, j]
  are all true. Reference: WARSHALL, S. A theorem on Boolean
  matrices, J.ACM 9(1962), 11–12;
begin
integer i, j, k;
for i := 1 step 1 until n do
for j := 1 step 1 until n do
if m [j, i] then
for k := 1 step 1 until n do
if m [i, k] then
m [j, k] := true
end ancestor
```

ALGORITHM 97
SHORTEST PATH

ROBERT W. FLOYD
Armour Research Foundation, Chicago, Ill.

```
procedure shortest path (m, n); value n; integer n; array m;
comment Initially m[i, j] is the length of a direct link from
  point i of a network to point j. If no direct link exists, m [i, j] is
  initially ₁₀10. At completion, m [i, j] is the length of the shortest
  path from i to j. If none exists, m [i, j] is ₁₀10. Reference: WAR-
  SHALL, S. A theorem on Boolean matrices. J, ACM 9(1962), 11–12;
begin
integer i, j, k; real inf, s; inf := ₁₀10;
for i := 1 step 1 until n do
for j := 1 step 1 until n do
if m [j, i] < inf then
for k := 1 step 1 until n do
if m [i, k] < inf then
begin s := m [j, i] + m [i, k];
if s < m [j, k] then m [j, k] := s
end
end shortest path
```


ALGORITHM 98
EVALUATION OF DEFINITE COMPLEX LINE INTEGRALS

JOHN L. PFALTZ
Syracuse University Computing Center, Syracuse, N. Y.

```
procedure COMPLINEINTGRL(A, B, N, RSSUM);
  value A, B, N; real A, B, N; array RSSUM;
comment COMPLINEINTGRL approximates the complex line
  integral by evaluating the partial Riemann-Stieltjes sum
  ∑_{t=1}^{n} f(z_k)[z_t − z_{t−1}] where a ≦ t ≦ b and z_k ∈ (z_{t−1}, z_t). The
  programmer must provide 1) the procedures GAMMA(T, Z) to
  calculate z(t) on Γ, and FUNCT(Z, F) to calculate function
  values, and 2) the end points A and B of the parametric interval
  and N the number of subintervals into which [a, b] is to be
  partitioned;
begin integer I; real T, DELT; real array ZT, ZTL, DELZ,
      ZK, PART[1:2]; RSSUM[1] := 0.0; RSSUM[2] := 0.0;
      DELT := (B − A)/N; T := A;
line: GAMMA(T, ZT);
      if T = A then go to next;
      for I := 1 step 1 until 2 do
      begin
        DELZ[I] := ZT[I] − ZTL[I]; end;
      for I := 1 step 1 until 2 do
      begin
        ZK[I] := ZTL[I] + DELZ[I]/2.0; end;
      FUNCT(ZK, FZ);
      PART[1] := FZ[1] × DELZ[1] − FZ[2] × DELZ[2];
      PART[2] := FZ[1] × DELZ[2] + FZ[2] × DELZ[1];
      for I := 1 step 1 until 2 do
      begin
        RSSUM[I] := RSSUM[I] + PART[I]; end;
      if T < B − (0.25 × DELT) then go to next else go to
        exit;
next: for I := 1 step 1 until 2 do
      begin
        ZTL[I] := ZT[I]; end;
      T := T + DELT;
      go to line;
exit: end COMPLINEINTGRL.
```

ALGORITHM 99
EVALUATION OF JACOBI SYMBOL

STEPHEN J. GARLAND AND ANTHONY W. KNAPP
Dartmouth College, Hanover, N. H.

```
procedure Jacobi (n,m,r); value n,m;
integer n, m, r;
comment Jacobi computes the value of the Jacobi symbol (n/m),
  where m is odd, by the law of quadratic reciprocity. The param-
  eter r is assigned one of the values −1, 0, or 1 if m is odd. If m
  is even, the symbol is undefined and r is assigned the value 2.
  For odd m the routine provides a test of whether m and n are
  relatively prime. The value of r is 0 if and only if m and n have
  a nontrivial common factor. In the special case where m is prime,
  r = −1 if and only if n is a quadratic nonresidue of m;
begin
  integer s;
  Boolean p, q;
  Boolean procedure parity (x); value x; integer x;
    comment The value of the function parity is true if x is
      odd, false if x is even;
    begin
      parity := x ÷ 2 × 2 ≠ x
    end parity;
```

```
  if ¬ parity (m) then begin r := 2;  go to exit end;
  p := true;
  loop:  n := n − n ÷ m × m;
         q := false;
         if n ≦ 1 then go to done;
  even:  if ¬ parity (n) then
         begin
           q := ¬ q;
           n := n ÷ 2;
           go to even
         end n now odd;
         if q then if parity ((m↑2 − 1)÷8) then p := ¬ p;
         if n = 1 then go to done;
         if parity ((m−1) × (n−1) ÷ 4) then p := ¬ p;
         s := m;  m := n;  n := s;  go to loop;
  done:  r := if n = 0 then 0 else if p then 1 else −1;
exit:  end Jacobi
```

ALGORITHM 101
REMOVE ITEM FROM CHAIN-LINKED LIST

PHILIP J. KIVIAT
United States Steel Corp., Appl. Res. Lab., Monroeville, Penn.

```
procedure  outlist (vector,m,list,n,first,flag,addr);
integer  n,m,first,flag;  integer array vector,list,addr;
comment  outlist removes the first entry (information pair with
  lowest order key) from list(i,j) and puts it in vector(k);
begin integer i;
for i := 1 step 1 until m+1 do vector[i] := list [first,i];
for i := n−1 step −1 until 1 do addr [i+1] := addr [i];
addr [1] := first;
if list [first,m+3] = flag then
  begin list [1,m+2] := flag;  first := 1;
  for i := 1 step 1 until n do addr [i] := i end;
else begin first := list [first,m+3];
list [first,m+2] := flag end;
for i := 1 step 1 until m+3 do list [addr [1], i] := 0
end outlist
```

ALGORITHM 100
ADD ITEM TO CHAIN-LINKED LIST

PHILIP J. KIVIAT
United States Steel Corp., Appl. Research Lab., Monroeville, Penn.

```
procedure  inlist (t,info,m,list,n,first,flag,addr,listfull);
integer  n,m,first,flag,t;  integer array info,list,addr;
comment  inlist adds the information pair {t,info} to the chain-
  link structured matrix list (i,j), where t is an order key ≧ 0, and
  info(k) an information vector associated with t. info(k) has di-
  mension m, list(i,j) has dimensions (n × (m+3)). flag denotes
  the head and tail of list(i,j), and first contains the address of the
  first (lowest order) entry in list(i,j).  addr(k) is a vector con-
  taining the addresses of available (empty) rows in list(i,j).
  Initialization: list(i,m+2) = flag, for some i ≦ n. If list(i,j) is
  filled exit is to listfull;
begin integer i, j, link1, link2;
0:  if addr [1] = 0;  then go to listfull;  i := 1;
1:  if list [i,1] ≦ t
      then begin if list [i,2] ≠ 0 then begin link1 := m+2;
        link2 := m+3;  go to 2 end;  else begin if
        list [i,m+2] = flag then begin i := flag;
        link1 := m+3;  link2 := m+2;  go to 3 end;
        else begin i := i+1;  go to 1 end end end;
      else begin link1 := m+3;  link2 := m+2 end;
2:  if list [i,link2] ≠ flag
      then begin k := i;  i := list [i,link2];
        if (link2 = m+2 ∧ list [i,1] ≦ t) ∨
          (link2 ≠ m+2 ∧ list [i,1] > t) then go to 4;
        else go to 1 end;
      else begin list [i,link2] := addr [1] end;
3:  j := addr [1];  list [j,link1] := i;
    list [j,link2] := flag;  if link2 = m+2 then
    first := addr [1];  go to 5;
4:  j := addr [1];  list [j,link1] := list [i,link1];
    list [i,link1] := list [k,link2] := addr [1];
    list [j,link2] := i;
5:  list [j,1] := t;  for i := 1 step 1 until m do
    list [j,i+1] := info [i];  for i := 1 step 1 until n−1 do
    addr [i] := addr [i+1];  addr [n] := 0
end  inlist
```

ALGORITHM 102
PERMUTATION IN LEXICOGRAPHICAL ORDER

G. F. SCHRACK AND M. SHIMRAT
University of Alberta, Calgary, Alberta, Canada

```
procedure  PERMULEX(n,p);
integer  n;  integer array p;
comment  Successive calls of the procedure will generate all
  permutations p of 1,2,3,···,n in lexicographical order. Before the
  first call, the non-local Boolean variable 'flag' must be set to
  true. If after an execution of PERMULEX 'flag' is false,
  additional calls will generate further permutations—if true, all
  permutations have been obtained;
begin integer array q[1:n];  integer i, k, t;  Boolean flag2;
if flag then
  begin for i := 1 step 1 until n do
  p[i] := i;  flag2 := true;  flag := false;
  go to EXIT
  end initialize;
if flag2 then
  begin t := p[n];  p[n] := p[n−1];  p[n−1] := t;
  flag2 := false;  go to EXIT
  end bypass;
flag2 := true; for i := n−2 step −1 until 1 do
  if p[i] < p[i+1] then go to A;
  flag := true;  go to EXIT;
A:  for k := 1 step 1 until n do q[k] := 0;
    for k := i step 1 until n do q[p[k]] := p[k];
    for k := p[i] + 1 step 1 until n do
    if q[k] ≠ 0 then go to B;
B:  p[i] := k; q[k] := 0;
    for k := 1 step 1 until n do
    if q[k] ≠ 0 then begin i := i + 1; p[i] := q[k] end
    else if i ≧ n then go to EXIT;
EXIT:
end PERMULEX
```

ALGORITHM 103
SIMPSON'S RULE INTEGRATOR

GUY F. KUNCIR
UNIVAC Division, Sperry Rand Corp., San Diego, Calif.

```
procedure SIMPSON (a, b, f, I, i eps, N);
  value a, b, eps, N; integer N;
  real a, b, I, i, eps; real procedure f;
comment This procedure integrates the function f(x) using a
  modified Simpson's Rule quadrature formula. The quadrature is
  performed over j subintervals of [a,b] forming the total area I.
  Convergence in each subinterval of length (b−a)/2^n is indicated
  when the relative difference between successive three-point and
  five-point area approximations
      A3,j = (b−a)(g0 + 4g2 + g4)/(3·2^(n+1))
      A5,j = (b−a)(g0 + 4g1 + 2g2 + 4g3 + g4)/(3·2^(n+2))
  is less than or equal to an appropriate portion of the over-all
  tolerance eps (i.e., |(A5,j − A3,j)/A5,j | ≦ eps/2^n with n ≦ N).
  SIMPSON will reduce the size of each interval until this con-
  dition is satisfied.
    Complete integration over [a,b] is indicated by i = b. A value
  a ≦ i < b is indicates that the integration was terminated, leav-
  ing I the true area under f in [a,i]. Further integration over [i,b]
  will necessitate either the assignment of a larger N, a larger eps,
  or an integral substitution reducing the slope of the integrand in
  that interval. It is recommended that this procedure be used
  between known integrand maxima and minima.;
begin integer m, n; real d, h; array g[0:4], A[0:2], S[1:N, 1:3];
  I := i := m := n := 0;
  g[0] := f(a);
  g[2] := f((a + b)/2);
  g[4] := f(b);
  A[0] := (b − a) × (g[0] + 4 × g[2] + g[4])/2;
AA: d := 2↑n; h := (b − a)/4/d;
    g[1] := f(a + h × (4 × m + 1));
    g[3] := f(a + h × (4 × m + 3));
    A[1] := h × (g[0] + 4 × g[1] + g[2]);
    A[2] := h × (g[2] + 4 × g[3] + g[4]);
    if abs (((A[1] + A[2]) − A[0])/(A[1] + A[2])) > eps/d
      then begin m := 2 × m; n := n + 1;
            if n > N then go to CC;
            A[0] := A[1]; S[n,1] := A[2];
            S[n,2] := g[3]; S[n,3] := g[4];
            g[4] := g[2]; g[2] := g[1]; go to AA
           end
    else begin I := I + (A[1] + A[2])/3;
          m := m + 1; i := a + m × (b − a)/d;
BB:       if m = 2 × (m ÷ 2) then
            begin m := m ÷ 2; n := n − 1; go to BB end
          if (m ≠ 1) ∨ (n ≠ 0) then
            begin A[0] := S[n,1]; g[0] := g[4];
                  g[2] := S[n,2]; g[4] := S[n,3]; go to AA end
         end
CC: end SIMPSON
```

REMARK ON ALGORITHM 19
RINOMIAL COEFFICIENTS (Richard R. Kenyon, *Comm. ACM*, Oct. 1960)

BICHARD STECK
Armour Research Foundation, Chicago 16, Ill.

The **for** clause of Algorithm 19 should read:

```
for i := 0 step 1 until b−1 do
```

With this correction the algorithm was certified on the Armour Research Foundation UNIVAC 1105.

The recursion formula stated in the **comment** should read:

$$C_{i+1}^{n} = (n-i)\ C_i^n/(i+1).$$

CERTIFICATION OF ALGORITHM 46
EXPONENTIAL OF A COMPLEX NUMBER (J. R. Herndon, *Comm. ACM 4* (Apr., 1961), 178)

A. P. RELPH
Atomic Power Div., The English Electric Co., Whetstone, England

Algorithm 46 was translated using the DEUCE ALGOL compiler, no corrections being required, and gave satisfactory results.

CERTIFICATION OF ALGORITHM 48
LOGARITHM OF A COMPLEX NUMBER (J. R. Herndon, *Comm. ACM 4* (Apr., 1961), 179)

A. P. RELPH
Atomic Power Div., The English Electric Co., Whetstone, England

Algorithm 48 was translated using the DEUCE ALGOL compiler, after certain modifications had been incorporated, and then gave satisfactory results.

The original version will fail if $a = 0$ when the procedure for arctan is entered. It also assumes that $-\pi/2 < d < 3\pi/2$, whereas the principal value for logarithm of a complex number assumes $-\pi < d \leqq \pi$.

Incidentally, the ALGOL 60 identifier for natural logarithm is ln, not log.

The modified procedure is as follows:

```
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 the principal value of log_e (a + bi). If a = 0 then c is
  put equal to −10^47 which is used to represent "− infinity";
begin integer m,n
  m := sign (a); n := sign (b);
  if a = 0 then begin c := −10^47;
                      d := 1.5707963 × n;
                      go to k
  end;
  c := sq rt(a × a + b × b);
  c := ln (c);
  d := 1.5707963 × (1−m) × (1+n−n×n) + arctan (b/a);
k: end LOGC;
```

CERTIFICATION OF ALGORITHM 58
MATRIX INVERSION (Donald Cohen, *Comm. ACM 4*, May 1961)

RICHARD A. CONGER
Yalem Computer Center, St. Louis University, St. Louis, Mo.

Invert was hand-coded in FORTRAN for the IBM 1620. The following corrections were found necessary:

The statement $a_{k,j} := a_{k,i} - b_j \times c_k$ *should be*

$$a_{k,j} := a_{k,j} - b_j \times c_k$$

The statement **go to** back *should be changed to*

```
i := z_k ; z_k := z_j ; z_j := i; go to back
```

After these corrections were made, the program was checked by inverting a 6 × 6 matrix and then inverting the result. The second result was equal to the original matrix within round-off.

CERTIFICATION OF ALGORITHM 66
INVRS (J. Caffrey, *Comm. ACM*, July 1961)

JOHN CAFFREY
Palo Alto Unified School District, Palo Alto, California

INVRS was translated using the Burroughs 220 Algebraic Computer (BALCOM) at Stanford University, using 8-digit floating-point arithmetic. The misprint noted by Randell and Broyden (*Comm. ACM*, Jan. 1962, p. 50) was corrected, and the same example (Wilson's 4 × 4 matrix) was used as a test case. The resulting inverse was:

68.0000	−41.0000	−17.0000	10.0000
	25.0000	10.0000	−6.0000
		5.0000	−3.0000
			2.0000

It may also be useful to note that the determinant of the matrix may be obtained as the successive product of the pivots. That is, if t_i ($=T(1,1)$) is the ith pivot of a matrix of order n,

$$\text{determinant} = \prod_i^n t_i .$$

For the above *input* example,

determinant = 1.0

Randell and Broyden's observation concerning the *apparent* limitation of INVRS to positive definite cases is correct: That is, any nonsingular real symmetric matrix (positive, indefinite, or negative) may be inverted using this algorithm. The original INVRS should therefore be modified as follows:

if pivot = 0 **then go to** singular;

Randell and Broyden's second example (of order 5) was also used as a test case, with the resulting inverse:

−.0000	.9999	.0000	.0000	.9999
	1.5333	−.7333	−.1333	.7999
		−.8666	−1.0666	−.5999
			−1.4666	−.1999
				.2000

determinant = −14.999999

An attempt to invert the *inverse* of the 4 × 4 segment of the Hilbert matrix, as presented by Randell (*Comm. ACM*, Jan. 1962, p. 50), yielded the following results:

.9999	.4999	.3333	.2499
	.3333	.2499	.1999
		.1999	.1666
			.1428

determinant = 6048020.6

CERTIFICATION OF ALGORITHM 67
CRAM (J. Caffrey, *Comm. ACM 4* (July 1961), 322)

A. P. RELPH
Atomic Power Div., The English Electric Co., Whetstone, England

CRAM was translated using the DEUCE ALGOL compiler with the following corrections:

V[i] = S *was changed to* V[i] := S
f[k,j] = V[k] *was changed to* f[k,j] := V[k]

It is quicker not to use the table of the C[i] in the "load" sequence and instead use the following sequence:

```
load: m := n × (n+1)/2;
      for i := 1 step 1 until m do READ (a[i]);
```

REMARK ON ALGORITHM 76
SORTING PROCEDURES (Ivan Flores, *Comm. ACM 5*, Jan. 1962)

B. RANDELL
Atomic Power Div., The English Electric Co., Whetstone, England

The following types of errors have been found in the Sorting Procedures:

1. Procedure declarations not starting with **procedure**.
2. Bound pair list given with array specification.
3. = used instead of :=, in assignment statements, and in a **for** clause.
4. A large number of semicolons missing (usually after **end**).
5. Expressions in bound pair lists in array declarations depending on local variables.
6. Right parentheses missing in some procedure statements.
7. Conditional statement following a **then.**
8. No declarations for A, or z, which is presumably a misprint.
9. In several procedures attempt is made to use the same identifier for two different quantities, and sometimes to declare an identifier twice in the same block head.
10. In the Presort quadratic selection procedure an array, declared as having two dimensions, is used by a subscripted variable with only one subscript.
11. At one point a subscripted variable is given as an actual parameter corresponding to a formal parameter specified as an array.
12. In several of the procedures, identifiers used as formal parameters are redeclared, and still assumed to be available as parameters.
13. In every procedure K is given in the specification part, with a parameter, whilst not given in the formal parameter list.

No attempt has been made to translate, or even to understand the logic of these procedures. Indeed it is felt that such a grossly inaccurate attempt at ALGOL should never have appeared as an algorithm in the *Communications*.

CERTIFICATION OF ALGORITHM 77
AVINT (Paul E. Hennion, *Comm. ACM 5*, Feb., 1962)

VICTOR E. WHITTIER
Computations Res. Lab., The Dow Chemical Co., Midland, Mich.

AVINT was transliterated into BAC-220 (a dialect of ALGOL-58) and was tested on the Burroughs 220 computer. The following minor errors were found:

1. The first statement following label L11 should read:
   dif := 2 × a × xarg + b;
2. The semicolon (;) at the end of the line beginning with the label L16 should be deleted.
3. There appears to be a confusion between "1" (numeric) and "l" (alphabetic) following label L12. This portion of the program should read:
   L12: sum := 0; syl := xlo; jul := nop − 1; ib := 2;

After making the above corrections the procedure was tested for interpolation, differentiation, and integration using e^x, log X, and sin X in the range ($1.0 \leqq X \leqq 5.0$). Twenty-one values of each of these functions, evenly spaced with respect to X and accurate to at least 7 significant digits, were tabulated in the above range. Then the procedure was tested. The following table indicates approximately the accuracy obtained:

	Number of Significant Digits		
Function	*Interpolation*	*Differentiation*	*Integration*
e^x	$\geqq 4$*	$\geqq 2$	$\geqq 4$
log X	$\geqq 4$*	$\geqq 2$	$\geqq 3$
sin X	$\geqq 4$*	$\geqq 2$	$\geqq 4$

* Except for interpolation between the first two points in the table.

The above results are quite reasonable in view of the relatively large increment in X. Tests using smaller increments in X and uneven spacing of X were also satisfactory.

It was also discovered that for integration the following restrictions must be observed:

1. xlo ≦ xa (1).
2. xup ≧ xa (nop).