APPENDIX
BNF Definition of APAREL's Syntax Language

```
⟨parse-request⟩ ::= ⟨parse-delimitator⟩⟨parse-request-name⟩:
  ⟨parse-alternative-list⟩⟨parse-delimitator⟩
⟨parse-alternative-list⟩ ::=
  ⟨parse-alternative-name⟩⟨parse-element-list⟩|
  ⟨parse-alternative-name⟩⟨parse-element-list⟩'|'
  ⟨parse-alternative-list⟩
⟨parse-element-list⟩ ::= ⟨parse-element⟩|
  ⟨parse-element⟩; ⟨parse-time-routine-name⟩|
  ⟨parse-element⟩⟨parse-element-list⟩|
  ⟨parse-element⟩. ⟨parse-element-list⟩|
  ⟨parse-element⟩ – ⟨parse-element-list⟩
⟨parse-element⟩ ::= ⟨parse-atom⟩|⟨parse-group⟩
⟨parse-group⟩ ::= '(' ⟨parse-alternative-list⟩')'
  '('⟨parse-request-name⟩:⟨parse-alternative-list⟩')'
⟨parse-atom⟩ ::= ⟨parse-name⟩|⟨text-literal⟩|
  ⟨primitive-parse-request-function⟩|⟨empty⟩
⟨parse-name⟩ ::= ⟨parse-request-name⟩|
  ⟨parse-request-sequence-name⟩
⟨parse-alternative-name⟩ ::= (⟨PL/1 identifier⟩)|⟨empty⟩
⟨parse-delimitator⟩ ::= ::
⟨parse-time-routine-name⟩ ::=
  ⟨name of a PL/1 bit valued function⟩ ⟨arguments⟩
⟨parse-request-name⟩ ::= ⟨PL/1 identifier⟩
⟨parse-request-sequence-name⟩ ::= ⟨PL/1 identifier⟩
⟨primitive-parse-request-function⟩ ::=
  ⟨reserved PL/1 identifier⟩ ⟨arguments⟩
⟨arguments⟩ ::= (⟨argument-list⟩)|⟨empty⟩
⟨argument-list⟩ ::= ⟨parse-atom⟩|⟨parse-atom⟩, ⟨argument-list⟩
```

Received September 1968; revised May 1969

REFERENCES

1. PL/I Language Specification. Form C28-6571-4, IBM Corp.
2. Balzer, R. M. Dataless programming. Proc. AFIPS 1967 Fall Joint Comput. Conf., Thompson Book Co., Washington, D.C., pp. 535-544. Also RM-5290-ARPA, Rand Corp., July 1967.
3. Strachey, C. (Ed.) *CPL Working* Papers. London Institute of Computer Science and the University Mathematical Laboratory; Cambridge, England, 1966.
4. Leavenworth, B. M. Syntax macros and extended translation. *Comm. ACM*, 9, 11 (Nov. 1966), 790-793.
5. Backus, J. W. The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. Proc. Intl. Conf. on Information Processing, UNESCO (1959), pp. 125-132.
6. Cheateam, T. E. The introduction of definitional facilities into higher level programming languages. Proc. AFIPS 1966 Fall Joint Comput. Conf., Spartan Books, New York, pp. 623-637.
7. Farber, D. J., Griswold, R. E. and Polonsky, I. P. "The SNOBOL3 programming language," *Bell Syst. Tech. J. 45*, 6, (July-Aug. 1966), 895-944.
8. Feldman, J. A., and Greis, D. Translator writing systems. *Comm. ACM 11*, (Feb. 1968), 77-113.
9. Galler, B., and Perlis, A. J. A proposal for definitions in ALGOL. *Comm. ACM 10*, 4 (Apr. 1967), 204-219.
10. Irons, E. T. A syntax directed compiler for ALGOL 60. *Comm. ACM* 4, 2 (Jan. 1961), 51-55.
11. McClure, R. M. TM6—A syntax-directed compiler. Proc. ACM 20th Nat. Conf., 1965, pp. 262-274.
12. Mondschein, L. VITAL compiler-compiler reference manual. TN 1967-1, Lincoln Lab., MIT, Lexington, Mass., Jan. 1967.

Algorithms

LLOYD D. FOSDICK, Editor

ALGORITHM 359
FACTORIAL ANALYSIS OF VARIANCE* [G1]

John R. Howell (Recd. 2 Aug. 1968 and 12 May 1969)
Department of Biometry, Medical Center, Virginia Commonwealth University, Richmond, VA 23219

* This investigation was supported in part by Public Health Service Research Grant FR 00016-05, from the National Institutes of Health.

KEY WORDS AND PHRASES: factorial variance analysis, variance, statistical analysis
CR CATEGORIES: 5.5

Comments. This subroutine transforms a vectory **y**, observed in a balanced complete $t_1 \times t_2 \times \ldots \times t_n$ factorial experiment, in to an interaction vector **z**, whose elements include mean and main effects.

The experimental observations y_s, ($s = (s_1, s_2, \ldots, s_n)$; $s_i = 0, 1, \cdots, t_i - 1$; $i = 1, 2, \cdots, n$) are assumed to be stored in the array Y in increasing order by the composite base integer s. After the transformation, the array Z will contain the interactions in natural order.

The method used is Good's [1, 2] modification of Yates's [5] interaction algorithm. In [1, p. 367], the interactions are expressed in the form $z = (M_1 \otimes M_2 \otimes \cdots \otimes M_n)\mathbf{y}$, where M_i is a $t_i \times t_i$ matrix of normalized orthogonal contrasts and where $\otimes$ denotes a direct (Kronecker, tensor) product. The interactions can also be written $z = (C_1C_2 \ldots C_n)\mathbf{y}$, where

$$C_1 = M_1 \otimes I_{t_2} \otimes \cdots \otimes I_{t_n}$$

$$C_2 = I_{t_1} \otimes M_2 \otimes \cdots \otimes I_{t_n}$$

$$\cdots\cdots\cdots\cdots\cdots\cdots$$

$$C_n = I_{t_1} \otimes I_{t_2} \otimes \cdots \otimes M_n$$

and where I_{t_i} is the $t_i \times t_i$ identity matrix.

By performing elementary operations (row and column interchanges) on the C_i we get $\mathbf{z} = (D_1D_2 \ldots D_n)\mathbf{y}$, where

$$D_i = \begin{pmatrix} M_{i1} \oplus \cdots \oplus M_{i1} \\ \hline M_{i2} \oplus \ldots \oplus M_{i2} \\ \cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots \\ M_{it_i} \oplus \ldots \oplus M_{it_i} \end{pmatrix}$$

and where M_{ij} is rowj of M_i. The symbol $\oplus$ denotes a direct sum. For an example of this for an unnormalized matrix, see Good [1, p. 362].

Since each row of D_i consists of a row of M_i and zeros, we only need M_i for forming **z**. The subroutine forms first $D_n\mathbf{y}$, then this result is premultiplied by D_{n-1}, and so on until we obtain **z**. The elements of **z** are the required interactions.

This method can be mechanized for hand computation in the following way. (The subroutine was written from this point of view.) Write the observations in the order specified above. Write row one of M_n down the right edge of a strip of paper using the same spacing as for the observations. Now place this movable strip alongside the observation vector so that the top element on the paper strip is opposite the top element of the observation vector. Multiply adjacent elements and write the sum of these products at the top of a new column. Now slide the paper strip down t_n spaces. Form the indicated inner product as before and write the result in the new column below the previous entry. Continue in this manner until all the observations have been used. Now write row two of M_n on a strip of paper and proceed as before. If we continue this process with all the rows of M_n we will get a new vector $\mathbf{z}_n$ whose elements are linear transformations of the observation vector **y.** The dimension of $\mathbf{z}_n$ is the same as that of y. Similarly form $\mathbf{z}_{n-1}$ from $\mathbf{z}_n$ and M_{n-1} . Continuing this process we finally obtain $\mathbf{z}_1 = \mathbf{z}$ which is the desired interaction vector.

In all the foregoing we used the normalized contrast matrices; thus the sums of squares are the squares of the elements of **z.** For hand computation, one might prefer using the unnormalized contrast matrices, since their elements are integers. But then we need a vector of divisors; it is obtained by performing the same operations on a column of ones as on **y,** except that we use the squares of the elements of the contrast matrices. Then the ith sum of squares equals z_i^2 divided by the corresponding divisor.

This method might be called a "paper strip method" for analysis of variance and is similar to paper strip methods used for operations with polynomials. For examples of this, see Lanczos [3] and Prager [4].

We require $2t_1t_2 \cdots t_n$ locations for storing y and **z** plus $\sup(t_1, t_2, \cdots, t_n)$ locations for storing a row of M_i . The number of multiplications required is $(\prod t_i)(\sum t_i + 1)$.

ACKNOWLEDGMENTS: The author wishes to thank Dr. A. E. Brandt for initiating his interest in programming analysis of variance. He wishes to thank Dr. W. H. Carter, Jr., and the referee, for helpful comments.

REFERENCES:

1. GOOD, I. J. The interaction algorithm and practical Fourier analysis. *J. Roy. Statist. Soc.* {B} *20*, 2 (1958), 361-372.
2. GOOD, I. J. The interaction algorithm and practical Fourier analysis: An addendum. *J. Roy. Statist. Soc.* {B} *22*, 2 (1960). 372-375.
3. LANCZOS, C. *Applied Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1956.
4. PRAGER, W. *Introduction to Basic Fortran Programming and Numerical Methods.* Blaisdell, Waltham, Mass., 1965.
5. YATES, F. The design and analysis of factorial experiments. Imperial Bureau of Soil Science, Harpenden, England, 1937.

```
      SUBROUTINE FNOVA
C    *
     *   (Y,Z,ROW,MSIZE,NCLS,NFCTR)
      DIMENSION  Y(1),Z(1),
     *           ROW(1),MSIZE(1)
C     LOOP FOR  NFCTR CONTRAST  MATRICES
      DO  5  NF  =  1,NFCTR
          I       =  1
C     GET SIZE OF THE MATRIX
          K       =  NFCTR-NF+1
          NRNC    =  MSIZE(K)
      33  3     J =  1,NRNC
C     ROW OF A CONTRAST MATRIX
      CALL AROW(ROW,NRNC,J)
C     PERFORM THE 'PAPER STRIP'
C     OPERATION FOR A MATRIX ROW
      DO  2     K =  1,NCLS,NRNC
          Z(I)    =  0.
      DO  1  L    =  1,NRNC
          KL1     =  K+L-1
    1     Z(I)    =  Z(I)+ROW(L)*Y(KL1)
    2     I       =  I+1
    3     CONTINUE
C     MOVE  Z  INTO  Y
      DO  4   J   =  1,NCLS
    4     Y(J)    =  Z(J)
    5     CONTINUE
      DO  6   J   =  1,NCLS
    6     Y(J)    =  Y(J)*Y(J)
      RETURN
      END
```

```
      SUBROUTINE AROW
     ****************
     *   (ROW,NRNC,J)
      DIMENSION ROW(1)
C     IF  ROW  ONE
      IF(J-1)3,1,3
    1     A       =NRNC
          EL      =  1./SQRT(A)
      DO   2   I  =  1,NRNC
    2     ROW(I)=EL
C     AND
      RETURN
C     ELSE
    3     JM1     =  J-1
          RJ      =  J
          A       =  SQRT(RJ*RJ-RJ)
          EL      =  1./A
      DO   4   I  =  1,JM1
    4     ROW(I)= EL
      DO  5    I  =  J,NRNC
    5     ROW(I)=0.
          ROW(J)=  (1.-RJ)/A
      RETURN
      END
```

ALGORITHM 360 SHORTEST-PATH FOREST WITH TOPOLOGICAL ORDERING [H]

ROBERT B. DIAL (Recd. 21 Nov. 1968, 27 Nov. 1968 and 30 Apr. 1969)
Alan M. Voorhees and Associates, Inc., McLean, VA 22101, and Department of Civil Engineering, University of Washington, Seattle, WA 98105

KEY WORDS AND PHRASES: shortest path, tree, network, directed graph
CR CATEGORIES: 5.32, 5.42

```
procedure MOORE (INDEX, J, D, maxd, n, DIST, I, NEXT,
    LAST, maxdist, ROOT, m);
  value maxd, n, maxdist, m;
  integer array INDEX, J, D, DIST, I, NEXT, LAST, ROOT;
  integer maxd, n, maxdist, m;
comment Given a subset (called "roots") of the nodes (numbered from 1 to n) spanned by a directed graph composed of arcs of known length, MOORE finds for each node in the network the shortest path connecting it to its closest root node. The result is a disjoint set of shortest-path trees, referred to here as a "shortest-path forest." MOORE's output describes all the paths in the forest and gives their lengths. It also provides two lists which sequence the nodes spanned by the forest in forward and backward topological order. In the algorithm's terminology, "forward topological order" is a sequence in which any given node is listed after any other node which lies on the path between it and its root node. Conversely, the "backward topological order" has the nodes arranged in decreasing distance from their nearest root node.
```

The procedure MOORE implements a well-known, widely-used algorithm by E. F. *Moore* [1] and is particularly suited for a large, sparse network whose arc lengths are short and which have a small variance, e.g. an urban highway system. As an indication of its efficiency, an Assembly Language routine patterned after *MOORE* for the IBM 360 model 65 found all shortest paths from a single root node to the remaining 12,000 nodes of a 36,000-arc network (i.e. built a minimum-path tree) in one (1) second. In general, for a connected graph, *MOORE*'s "running time" is directly proportional to the number of arcs in the network and is independent of the number of roots. The mechanics of the algorithm are summarized in the following three steps :

0. Mark each root node r "reached but not scanned" and associate with it a distance of zero ($DIST[r]=0$). Mark each nonroot node i "not reached" and associate with it a distance of infinity (i.e. $DIST[i]=maxdist$). Go to Step 1.
1. From among the nodes marked "reached but not scanned," select the node i whose distance is smallest. If there is no node so marked, the forest is complete. Otherwise go to Step 2.
2. For each arc (i, j) in the network (i.e. all arcs exiting the selected node i), compare $DIST[j]$ with the sum of $DIST[i]$ and the arc length of (i, j). Whenever this latter sum is less than the former quantity, set $DIST[j]$ equal to it, mark node j "reached but not scanned," and put the arc (i, j) in the forest, removing any other arc whose final node is j. When all arcs exiting node i have been so examined mark node i "reached and scanned" and go to Step 1.

While Moore's algorithm possesses the important attribute of examining each arc in the network only once, the speed achieved in its implementation depends primarily on its efficiency in tep 1. To facilitate this node selection, the procedure below ses a topological ordering of the final nodes of the arcs in the artial forest. It effects Step 1 by referring to a forward-ordering list, *NEST*, to determine which node should be selected next from the "reached but not scanned" category. A backward-ordering list, *LAST*, aids updating the ordering when a previously found path to a node is superseded by a newly found, shorter one. Also used in this updating process are two short local vectors, *HEAD* and *TAIL*. *HEAD*[*d*] and *TAIL*[*d*] contain the first and last node of a sublist of nodes, whose associated distance is riot less than the distance of the node selected in Step 1 nd is congruent to *d* modulo the net's maximum arc length. he use of these latter two arrays becomes clear while studying the **ALGOL** below.

Besides the *m* root nodes stored in *ROOT*[1], ··· ,*ROOT*[*m*], input to *MOORE* consists of a network description in three vectors, *J*, *D*, and *INDEX*, together with thescalar parameters *n*, *maxd*, and *maxdist*. The array *J* contains the final node numbers of all arcs in the network stored in ascending sequence with respect to their initial node number, The second vector, *D*, is parallel to the array *J* and holds the corresponding arc lengths—against which paths are to be minimized. *INDEX*[*i*] points to the first element of *J* representing an arc exiting node *i*. *INDEX* is dinsioned from 1 to *n* + 1, where the parameter *n* is the highest node number in the network, and *INDEX*[*n*+1] contains one plus the total number of arcs in the network. The arc lengths stored in the array *D* must be positive integers strictly less than the parameter *maxd*. Similarly, as *maxd* exclusively limits the length of an arc, so does the other input scalar parameter *maxdist* limit the length of a path. *MOORE* only considers paths which are shorter than *maxdist*.

The algorithm's output describes the minimum-path forest in two vectors, *I* and *DIST*. *I*[*j*] contains the initial node of the f rest's unique are whose final node is *j*. Thus the sequence of des representing the shortest path from the nearest root to *j* is found in reverse order by looking at *I*[*j*], *I*[*I*[*j*]], etc., until a root node is encountered. *DIST*[*j*] returns the minimized distance from,the closest root node to *j*. If *j* is not reachable from any root node via a path shorter than *maxdist*, *MOORE* returns with *DIST*[*j*] = *mazdist* and *I*[*j*] = 0. The forest's topological orderings are returned in list form in the pointer vectors *NEXT* and *LAST*. *NEXT* is a circular successor list. The number of the node closest to its root node is stored in *NEXT*[*ROOT*[1]]. The next closest node is contained in *NEXT*[*NEXT*[*ROOT*[1]]], e c., until *ROOT*[1] is encountered in some *NEXT*[*j*], where *j* is the number of the node farthest from its root node. Similarly, *LAST* is a circular predecessor list. The backward topological order is obtained by starting at *LAST*[*ROOT*[1]], which contains the number of the most distant node. *LAST*[*LAST*[*ROOT*[1]]] has the next most distant,, etc., until *LAST*[*j*] = *ROOT*[1], *j* being the closest node to its root. When no path shorter than *maxdist* exists between a root node and *j*, then *j* appears in neither the *NEXT* nor the *LAST* list.

REFERENCE:

1. MOORE, E. F. Theshortest path through a maze. In *International Symposium on the Theory of Switching Proceedings*. Harvard U. Press, Cambridge, Mass., Apr. 1957, pp. 285–292;

```
begin
  integer procedure mod(d, w d ) ; value d, maxd; integer
    d, maxd; mod := d − mazd × entier(d ÷ maxd);
  integer array HEAD[0:maxd−1], TAIL[0:maxd−1]; integer
    i, pt, k, v, j, q, ct;
  for i := 1 step 1 until maxd−1 do HEAD[i] := TAIL[i] := 0;
  for i := 1 step 1 until n do
  begin DIST[i] := maxdist; I[i] := 0 end;
  for i := 2 step 1 until m do
  begin
    NEXT[ROOT[i−1]] := ROOT[i]; LAST[ROOT[i]] := ROOT
      [i−1];
    DIST[ROOT[i]] := 0
  end;
  LAST[ROOT[1]] := NEXT[ROOT[m]] := DIST[ROOT[1]] :=
    pt := 0;
  i := HEAD[0] := ROOT[1]; TAIL[0] := ROOT[m];
comment Examine all exits from selected node (Step 2 above);
r: for k := INDEX[i] step 1 until INDEX[i+1] − 1 do
  begin
    v := DIST[i] + D[k]; j := J[k];
    if v < DIST[j] then
    begin
      comment Path to j via i is shortest so far—put arc (i, j)
      in forest;
      if DIST[j] ≠ maxdist then
      begin
        comment Delete node j from its prior sublist;
        q := mod(DIST[j], maxd);
        if HEAD[q] = j then HEAD[q] := NEXT[j]
        else
        begin
          if TAIL[q] = j then
          begin TAIL[q] := LAST[j]; NEXT[LAST[j]] := 0
            end
          else
          begin LAST[NEXT[j]] := LAST[j]; NEXT[LAST
            [j]] := NEXT[j] end
        end
      end;
      comment Hook j to its new sublist, and put arc (i, j) in
        forest;
      q := mod(v, maxd);
      if HEAD[q] = 0 then
      begin HEAD[q] := j; LAST[j] := 0 end
      else
      begin LAST[j] := TAIL[q]; NEXT[TAIL[q]] := j end;
      comment Update forest and forward ordering;
      I[j] := i; DIST[j] := v; TAIL[q] := j; NEXT[j] := 0
    end
  end;
  comment Select next node i whose exit arcs are to be examined
    (Step 1 above);
  if NEXT[i] ≠ 0 then
  begin
    comment Sublist containing i not empty—use successor of
    i; i := NEXT[i]; go to r
  end;
  comment Sublist containing i empty—use first node in next
    nonempty sublist;
  HEAD[pt] := 0;
  for d := 1 step 1 until maxd − 1 do
  begin
    pt := mod(pt+1, maxd);
    if HEAD[pt] ≠ 0 then
    begin
      comment Found a nonempty sublist—hook it to lists;
      LAST[HEAD[pt]] := i; i := NEXT[i] := HEAD[pt];
        go to r
    end;
  end;
  comment All sublists empty, forest built—circularize lists
    and quit;
  LAST[ROOT[1]] := i; NEXT[i] := ROOT[1]
end MOORE
```

ALGORITHM 361
PERMANENT FUNCTION OF A SQUARE MATRIX I AND II [G6]

Bruce Shriver, P. J. Eberlein, and R. D. Dixon (Recd. 19 Feb. 1969, 7 Mar. 1969 and 9 July 1969)
State University of New York at Buffalo, Amherst, NY 14226

KEY WORDS AND PHRASES: matrix, permanent, determinant
CR CATEGORIES: 5.30

```
real procedure per1(A, n);
  integer n; array A;
```

comment Let A be an $n \times n$ real matrix, $n > 1$. The permanent function of A, denoted per(A), is computed by H. J. Ryser's [1] expansion formula:

$$\mathrm{per}(A) = \sum_{r=0}^{n-1} (-1)^r \sum_{x \in T_{n-r}} \prod_{i=1}^{r} x_i$$

where T_j, $j = n, n-1, \cdots, 2, 1$, is the set of vectors $x = (x_i)$, $i = 1, 2, \ldots, n$ which are obtained by adding j columns of A together in all $\binom{n}{j}$ possible ways. To effect the sum over vectors in T_j, $n-1$ sums are computed. The natural 1-1 map from the binary integers to all r-combinations, $r = 1, 2, \cdots, n-1$, is used to increment the sums over the sets T_j.

Reference:
1. Ryser, H. J. *Combinatorial Mathematics, Carus Monograph* #14, Wiley, New York, 1963, p. 27;

```
begin
  real sig, pera, prod, rowsum;
  integer number, limit, mod, gen, g, i, j, r;
  array sum[0:n-1];
  integer array d[1:n];
  sig := -1; pera := 0; limit := (2 ↑ n) - 1;
  for r := 0 step 1 until n - 1 do sum[r] := 0;
  for number := 1 step 1 until limit do
  begin
    r := 0; gen := number;
    for mod := 1 step 1 until n do
    begin
      g := gen ÷ 2; if (gen-g×2) = 1 then
      begin r := r + 1; d[r] := mod end;
      gen := g
    end;
    prod := 1;
    for i := 1 step 1 until n do
    begin
      rowsum := 0;
      for j := 1 step 1 until r do
      rowsum := rowsum + A[i, d[j]];
      prod := prod × rowsum
    end;
    sum[n-r] := sum[n-r] + prod
  end;
  for r := 0 step 1 until n - 1 do
  begin sig := -sig; pera := pera + sig × sum[r] end;
  per := pera
end of real procedure per1;
real procedure per2(A, n);
  integer n; array A;
```

comment Let A be an $n \times n$ real matrix, $n > 1$. The permanent function of A, denoted by per(A) is computed by Jurkat and Ryser's [1] method of inductively generating the vectors $p_1, \cdots, p_n$ where p_r is the vector of permanents of r by r submatrices of the first r rows of A. This vector has $\binom{n}{r}$ components indexed by the r-combinations of $\{1, \cdots, n\}$. The natural 1-1 map from the binary integers $\{1, \cdots, 2 \uparrow n-1\}$ to the r-combinations of $\{1, \cdots, n\}$ **for** $r = 1, \cdots, n$ is used to index the p's and thus they are generated in an order somewhat different from that of Jurkat and Ryser.

Reference:
1. Jurkat, W. B. and Ryser, H. J. Matrix factorizations of determinants and permanents. *J. Algebra 3* (1966), 1-27;

```
begin
  integer number, limit, mod, gen, g, r, dig, sub, j;
  array list[1:2 ↑ n-1];
  limit := 2 ↑ n - 1;
  comment Initialize list aa accumulators;
  for j := 1 step 1 until limit do list[j] := 0;
  for j := 1 step 1 until n do list[2 ↑ (j-1)] := A[1, j];
  for number := 1 step 1 until limit do
  begin
    if list[number] ≠ 0 then
    begin
      r := 1; gen := number;
      for mod := 1 step 1 until n do
      begin
        g := gen ÷ 2;
        if gen - 2 × g = 1 then r := r + 1;
        gen := g
      end count of 1's in number;
      dig := 1; gen := number;
      for mod := 1 step 1 until n do
      begin
        g := gen ÷ 2;
        if gen - 2 × g = 0 then
        begin
          sub := number + dig;
          list[sub] := list[sub] + list[number] × A[r, mod]
        end;
        gen := g; dig := 2 × dig
      end computations with list[number];
    end
  end;
  per := list[limit]
end of real procedure per2;
```

Note. On the Permanent Function of a Square Matrix I and II: Program I is slower than Program II. However Program II uses approximately 2^n more locations of store. The running times for both programs double when n is incremented by 1.

ALGORITHM 362
GENERATION OF RANDOM PERMUTATIONS [G6]

J. M. Robson (Recd. 1 Apr. 1969)
Programming Research Group, 45 Banbury Road, Oxford, England

KEY WORDS AND PHRASES: permutation, random permutation, transposition
CR CATEGORIES: 5.5

```
procedure perm(n, r, A); value n, r; integer n, r; teger
  array A;
```

comment This procedure produces in the vector A a permutation on the integers $1, 2, \cdots, n$, each of the $n!$ permutations being given by one value of r between 1 and $n!$ inclusive. It is thus similar in effect to the procedure given in [1] but it is considerably faster, especially for large values of n, since it uses single loop rather than a double one.

A permutation is generated as the product of $n - 1$ transpositions of which the jth transposes $A[n+1-j]$ and $A[x]$ for some $x \leq n + 1 - j$.

the line

```
for i := 1 step 1 until n do A[i] := i
```

is omitted the procediire will permute the originnl values $A[1], \cdots, A[n]$ in the same manner.

REFERENCE:

1. ROBINSON, C. L. Algorithm *317,* Permutation. *Comm. ACM 10* (Nov. 1967), 729;

```
begin
  integer i, x, y;
  for i := 1 step 1 until n do A[i] := i;
  for i := n step −1 until 2 do
  begin
      := r − (r÷i) × i + 1;  r := r ÷ i;
    , := A[x];  A[x] := A[i];  A[i] := y
  end
end
```

ALGORITHM 363
COMPLEX ERROR FUNCTION* [S15]

WALTER GAUTSCHI (Recd. 11 June 1969)
Computer Sciences Department, Purdue University, Lafayette, IN 47907

* Work supported, in part, by the National Aeronautics and Space Administration (NASA) under grant NGR 15-005-039 and, in part, by Argonne National Laboratory.

KEY WORDS AND PHRASES: error function for complex argument, Voigt function, Laplace continued fraction, Gauss-Hermite quadrature, recursive computation
CR CATEGORIES: **5.12**

```
procedure wofz(x, y, re, im);  value x, y;  real x, y, re, im;
comment This procedure evaluates the real and imaginary
  part of the function w(z) = exp(−z²)erfc(−iz) for argument
  z = x + iy in the first quadrant of the complex plane. The accu
  racy is 10 decimal places after the decimal point, or better.
  For the underlying analysis, see W. Gautschi, "Efficient com-
  putation of the complex error function," to appear in SIAM
  J. Math. Anal.;
begin
  integer capn, nu, n, npl;
  real h, h2, lambda, rl, r2, s, sl, s2, 11, 12, c;
  Boolean 6;
  if y < 4.29 ∧ x < 5.33 then
  hegin
    s := (1−y/4.29) × sqrt(1−x × x/28.41);
    h := 1.6 × s;  h2 := 2 × h;
    mpn := 6 + 23 × s;  nu := 9 + 21 × s
  end
  else
  begin h := 0;  capn := 0;  nu := 8 end;
  if h > 0 then lambda := h2 ↑ capn;
  b := h = 0 ∨ lambda = 0;
  rl := r2 := sl := s2 := 0;
  for n := nu step − 1 until 0 do
  begin
    npl := n + 1;
    tl := y + h + npl × rl;  t2 := x − npl × r2;
    c := .5/(tl × tl + t2 × t2);
    rl := c × tl;  r2 := c × 12;
    if h > 0 ∧ n ≦ capn then
    begin
      11 := lambda + sl;  sl := rl × tl − r2 × s2;
      s2 := r2 × tl + rl X s2;
      lambda := lambda/h2
    end
  end;
  re := if y = 0 then exp(−x×x) else
      1.12837916709551 × (if b then rl else sl);
  im := 1.128879113700551× (if b then r2 else s2)
end wofz
```

CERTIFICATION OF ALGORITHM 47 [S16] ASSOCIATED LEGENDRE FUNCTIONS OF THE FIRST KIND FOR REAL OR IMAGINARY ARGUMENTS [John R. Herndon, *Comm. ACM 4* (Apr. 1961), 178]

S. M. COBB (Recd. 6 Feb. 1969, 12 May 1969 and 9 July 1969)
The Plessey Co. Ltd., Roke Manor, Romsey, Hants, England

KEYWORDS AND PHRASES. Legendre function, associated Legendre function, real or imaginary arguments
CR CATEGORIES: **5.12**

This procedure was tested and run **on** the I.C.T. Atlas computer.

In addition to the errors mentioned in the certification of August *1963* [2] the following points were noted.

1. The requirement that when $n < m$ $p := 0$ must take precedence over $p := 1$ when $n = 0$. Hence the order **of** the first two **if** statements must be interchanged.

2. Most computers fail on division by zero. Hence the statement beginning **if** $x = 0$ **then** and ending with **go** *to last* **end**; should be inserted between w := I; and $y := w/(x \times x)$.

3. When $x = 0$, if the argument of the Legendre function is to be considered as real p must be multiplied by $(-1)^i$. This is achieved by inserting after the statement beginning $p := Gamma[m+n+1]$ the **if** statement

```
iff then p := p × (−1) ↑ i;
```

(For a change in the meaning of r see item **5** below.)

4. After the label ***last*** in the compound statement beginning **if** $r \# 0$ the statement $i := n - n \div 4$; is wrong. This should read

```
i := n − 4 × (n÷4);
```

5. Since r is used only as an indicator it is better that it be declared **as Boolean.** It can then be given the value **true** if the argument of the Legendre function is x and **false** if it is ix. The following program changes are then necessary. The statement beginning

```
if r = 0 then
```

becomes

```
if r then
```

The statement beginning

```
if r ≠ 0 then
```

becomes

```
if ¬ r then
```

6. Computing time can be saved in several ways. First we should declare another integer k and set it equal **to** $n - m$. The first statement **of** the procedure is then

```
k := n − m;
```

The next statement will begin

```
if k < 0 then
```

(This replaces **if** $n < m$ **then** whose position has been changed **in** accordance with item 1 above.)

$n - m$ is then replaced by k in the lines

for i := 1 **step** 1 **until** $n - m$ **do**

and

if $(i+1) \neq (n-m)$ **then**

Removing j as suggested in the previous certification leaves it free to be set to $k \div 2$. This requires the following modification: instead of the unnecessary statement **if** $n = m$ **then go to** *main* put

```
j := k ÷ 2;
```

In the statement beginning **if** $x = 0$ **then** replace the line

```
begin i := (n - m) ÷ 2;
```

by

```
begin a := j;
```

In the **for** loop beginning **for** i := 1 **step** 1 **until** 12 **do** a further small saving in computer time could be achieved by setting k to $n - i$. The loop thus becomes

```
for i := 1 step 1 until 12 do
begin if j + 1 < i then go to last;
  k := n - i;
  p := p + Gamma[2×k+3] × z/Gamma[i] × Gamma[k+2] ×
    Gamma[k-i-m+3]);
  z := z × y
  end
```

For real argument the program was tested as follows.

(i) $x = 0(0.1)1$, $m = 0(1)3$, $n = 0(1)3$
(ii) $x = 1.2(0.2)2.8$, $m = 0(1)2$, $n = 0(1)2$
(iii) $m = 0$, $n = 9$, $x = 0(0.2)1$, $2(2)10$.

For imaginary argument we used

$x = 0(0.2)2$, $m = 0(1)2$, $n = 0(1)2$.

Checking for real argument was carried out where possible using [1], agreement being obtained in all cases to the maximum number of figures available, which varied between 6 and 8. For all other cases [3] had to be used, giving only a 5 figure check

REFERENCES:

1. ABRAMOWITZ, M., AND STEGUN, I. A. Handbook of mathematical functions. AMS 55, Nat. Bur. Stand. US Govt. Printing Off., Washington, D.C., 1964.
2. GEORGE, R. Certification of Algorithm 47. *Comm. ACM 6*, (Aug. 1963), 446.
3. MORSE, P. M., AND FESBACH, H. *Methods of Theoretical Physics Pt. II.* McGraw Hill, New York, 1953.

CERTIFICATION OF ALGORITHM 255 [CG] COMPUTATION OF FOURIER COEFFICIENTS [Linda Teijelo, *Comm. ACM 8* (May 1965), 279]

GILLIAN HALL* AND VALERIE A. RAY† (Recd. 31 Mar. 1969 and 1 July 1969)
National Physical Laboratory, Teddington, Middlesex, England

* M.R.C. team, Division of Computer Science (formerly of Division of Numerical and Applied Mathematics).
† Division of Numerical and Applied Mathematics.

KEY WORDS AND PHRASES: numerical integration, Fourier coefficients, Filon's method
CR CATEGORIES: 5.16

The algorithm was translated using the KDF9 Kidsgrove ALGOL compiler, and needed the following correction.

The tests for convergence on lines 51 and 83 should read respectively:

```
if abs(prevint2-int2) < eps × abs(int2) ∧ n > 5 then
if abs(prevint1-int1) < eps × abs(int1) ∧ n > 5 then
```

With this alteration, the program was tested successfully on a series of functions $F(x)$ using a range of values of m and *eps* for each function. The parameter *subdivmax* was set at the recommended value, 10. For $F(x) = x^2$, for which the method is exact, results were obtained correct to machine accuracy, i.e. $10\frac{1}{2}$ decimal places.

Remarks. (i) It would be better to declare the identifier *tnl* as type **integer**, i.e. to replace lines 20 and 21 of the text by:

```
c0, c1, s0, s1, int1, int2, prevint1, prevint2, t3, temp;
integer n, i, tnl; Boolean bool;
```

(ii) There is no indication, after execution of the algorithm, whether the computation was terminated because of apparent convergence or because the number of times, n, that the interval was halved became greater than *subdivmax*. The following modification provides such an indication; it has the effect that *cosine* and *sine* will retain their entry values except in the case where *cosine* or *sine* has the value **true** on entry and n becomes greater than *subdivmax* in the course of computation. In this case the value on exit will be **false**.

Line 3 becomes:

```
value eps, subdivmax, m; real eps, cint, sint;
```

Line 57 becomes:

```
sint := int2; sine := false; go to L0
```

Line 88 becomes:

```
cosine := false; go to exit end;
```

(iii) To avoid the repeated evaluation of $F(0)$, $F(1.0)$ the following modification is suggested:

Declare a new variable *term1* of type **real** on line 20.
Replace lines 23 and 24 by:

```
term1 := F(1.0) × cos(k);
sumcos := (F(0)+term1) × 0.5;
sumsine := 0;
term1 := 2 × (sumcos-term1);
```

Replace lines 44, 45 and 49, 50 by:

```
prevint2 := (a×term1+b×sumsine+g×oddsine) × 0.5;
begin int2 := h × (a×term1+b×sumsine+g×oddsine);
```

Replace lines 76, 77 and 81, 82 by:

```
prevint1 := (b×sumcos+g×oddcos) × 0.5;
begin int1 := h × (b×sumcos+g×oddcos);
```

The work described above has been carried out at the National Physical Laboratory.

CERTIFICATION OF ALGORITHM 296 [E2] GENERALIZED LEAST SQUARES FIT BY ORTHOGONAL POLYNOMIALS [G. J. Makinson, *Comm. ACM* 10 (Feb. 1967), 87]

WAYNE T. WATSON (Recd. 11 Feb. 1969 and 21 Mar. 1969)
Service Bureau Corp., Development Laboratory, 11 West St. John Street, San Jose, CA 95113

KEY WORDS AND PHRASES: least squares, curve fitting, orthogonal polynomials, three-term recurrence, polynomial regression, approximation, Forsythe's method
CR CATEGORIES: 5.13, 5.5

LSFITUW was compiled and tested in CALL/360:PL/I. No modifications were made to the algorithm, and the computations were made in long precision (about 15 significant floating point digits). In addition, *POLYS* [2] was used to transform the results of *LSFITUW* from the interval $(-2,2)$ to the interval (x_1, x_m).

To generally test the algorithm, several small sets of data were used with *LSFITUW* and the results were compared with those obtained from an independently written polynomial curve fitting algorithm which does not use the method of orthogonal polynomials. Only polynomials of degree less than 5 were used to fit the data. Agreement between coefficients and standard errors was good.

As a more comprehensive test of the algorithm, all experiments that could be duplicated from the article by Ascher and Forsythe [1] were performed; a slight modification to *LSFITUW* was required to transform the data to the interval $(-1,1)$ instead of $(-2,2)$. Briefly, the experiments included:

(1) For certain equally spaced data, a comparison of the α_i and β_i calculated by the program against those values of α_i and β_i obtained from known formulas ($\alpha_i = 0$ for equally spaced data).

(2) A fit of the function $f(x) = |x|$ over the interval $(-1,1)$ for equally spaced data for polynomials of degree as high as 30.

(3) A fit of the function $f(x) = e^x$ for unequally spaced data inside the interval $(-1,1)$ for polynomials of degree as high as 32.

The results of experiment (1) showed that *LSFITUW* produced values of β_i differing only in the last significant digit (15) from those calculated by the known formula. The values of α_i produced were in the range of the floating point round-off error (10^{-15}). The results of duplicating experiments (2) and (3) were better than those reported in [1] because of the greater precision used in the calculations (about 10.8 versus about 15 significant floating digits). While conducting the last two experiments, it was noted that for data values of x symmetric about the origin, the value of b in the transformation equation $x = at + b$ may be computed to be a number in the floating point round-off range instead of exactly zero. When fitting polynomials of a sufficiently high degree, this may cause an underflow at line 4 of *POLYS*, the transformation routine. The user may find it desirable to branch on an underflow in *POLYX* and reset b to zero.

To check the computations of the σ_k^2 obtained by the recursive definition of σ_k^2 used in the algorithm, the σ_k^2 were compared with results computed directly from the equation

$$\sigma_k^2 = \sum_{j=1}^{m} (f_j - y_k(x_j))^2/(m-k-1) \qquad (*)$$

where y_k is the best fitting polynomial of degree k for the data x_j, f_j. Experience with the algorithm indicates that a loss of accuracy in computing σ_k^2 occurs at smaller values of k when using the recursive definition than when using (*). If the values of σ_k^2 are of importance to the user, he may find it useful to compute them using (*) instead.

A comprehensive test of the algorithm's feature which uses the σ_k^2 to automatically select the best fitting polynomial was not made, but the feature did work properly for the polynomials used. In connection with this feature, the user should be aware, though, of the possible difficulty mentioned above in computing σ_k^2 accurately using the recursive definition. In this case, the user should not expect the algorithm to select the best fitting polynomial. This difficulty was experienced several times while testing the algorithm, but was circumvented by using (*) to calculate σ_k^2. In order to detect a possible loss in accuracy, the σ_k^2 should be examined carefully or compared with those obtained by (*).

Comprehensive tests were not made using weights; however, no problems were encountered with a moderate usage of this feature.

References:

1. Ascher, M., and Forsythe, G. E. SWAC experiments on the use of orthogonal polynomials for data fitting. *J. ACM* 6 (Jan. 1958), 9–21.
2. MacKinney, John G. Algorithm 29, Polynomial transformer. *Comm. ACM* 3 (Nov. 1960), 604.

REMARK ON ALGORITHM 178 [E4]
DIRECT SEARCH [Arthur F. Kaupe, Jr., *Comm. ACM* 6 (June 1963), 313]; [as revised by M. Bell and M. C. Pike, *Comm ACM* 9 (Sept. 1966), 684]

F. K. Tomlin and L. B. Smith (Recd. *IT* May 1968, 9 Sept. 1968 and 30 June 1969)
Stanford Research Institute, Menlo Park, CA 94025, and CERN, DD Division, Geneva, Switzerland

KEY WORDS AND PHRASES: function minimization, search direct search
CR CATEGORIES: 5.19

The procedure *DIRECT SEARCH*, as modified by M. Bell and M. C. Pike [1], does not always provide the determined minimum. In addition, the maximum number of function evaluations permitted is almost always exceeded whenever the step-length is greater than *delta* at the time that number of function evaluations is greater than or equal to *maxeval*. Finally, the label 3 is not used.

To insure that the determined minimum is always provided, the test on the number of evaluations should be moved to a point where the minimum has been properly provided.

In [2] DeVogelaere remarks correctly that the procedure does not exit as specified and gives changes which will indeed cause the procedure to terminate when the number of function evaluations exceeds the specified limit (and not some number of evaluations later). However it is felt that DeVogelaere's solution to this problem causes excessive testing. Therefore the test should be performed after an exploratory move as in [1] but it should also be performed when the step-length is reduced. This method of testing violates the letter of the specified rise of *maxeval* but not the intent, which is to provide an escape from excessive calculation.

To obtain the determined minimum, to provide a means for reducing the number of function evaluations when step-length is greater than *delta*, and to eliminate the unused label:

(1) The lines

```
2:  if eval ≥ maxeval then
        begin converge := false
          go to EXIT
        end;
```

should be removed.

(2) The line (16th line from the end of the procedure given in [1])

```
for k := 1 step 1 until K do
```

should be changed to

```
2:  for k := 1 step 1 until K do
```

(3) The line

```
Spsi := SS;  SS := Sphi := S(phi);  eval := eval + 1;  E;
```

should have the following code inserted after the statement *Spsi* := *SS*;

```
if eval ≥ maxeval then
  begin
3:  converge := false;
    go to EXIT
  end;
```

(4) The line

```
3:  if DELTA ≥ delta then
```

should be changed to

```
if DELTA ≥ delta then
```

(5) The line

begin *DELTA* := *rho* × delta;

should be changed to

```
begin if eval > maxeval then go to 3 else
  DELTA := rho × delta;
```

REFERENCES:
1. BELL, M. AND PIKE, M. C. Remark on Algorithm 178. *Comm. ACM 9* (Sept. 1966), 684.
2. DEVOGELAERE, R. Remark on Algorithm 178. Comm. ACM 11 (July 1968), 498.

REMARK ON ALGORITHM 178 [EX]
DIRECT SEARCH [Arthur F. Kaupe, Jr., *Comm. ACM 6* (June 1963), 313; as revised by M. Bell and M. C. Pike, *Comm. ACM 9* (Sept. 1966), 684]

LYLE B. SMITH* (Recd. 9 Sept. 1968)
Stanford Linear Accelerator Center, Stanford, CA 94305
* Present address. CERN, Data Handling Division, 1211 Geneva 23, Switzerland

KEY WORDS AND PHRASES: function minimization, search, direct search
CR CATEGORIES: 5.19

Algorithm 178, as modified by Bell and Pike [1], has been used successfully by the author on a number of different problems and in a variety of languages (e.g. Burroughs Extended ALGOL on a B5500, SUBALGOL on an IBM 7090, and FORTRAN on the IBM/360 series machines). A modification which has been found to be useful involves tailoring the step size to be meaningful for a wide variation in the magnitudes of the variables.

As currently specified [1], each variable is incremented (or decremented) by *DELTA* as a minimum is sought. For a function such that the values of the variables differ by several orders of magnitude at the minimum, a universal step size causes some parameters to be essentially ignored during much of the searching process. For example, if a function of two variables has a minimum near (100.0, 0.1), a step size of 10.0 will be useful in minimizing with respect to the first parameter, but it will be meaningless with respect to the second parameter until it has been reduced to near 0.01. On the other hand, a step size of 0.01 would be useful on the second variable but on the first variable it would take an undesirably large number of steps to approach the minimum.

A modification to direct search which circumvents this scaling problem involves the use of a different step size for each variable. This is easily implemented since an array is already used to hold the signed step size for each variable. The change is accomplished by removing the statement labeled *Start* and replacing it by the following statement:

```
Start:  for k := 1 step 1 until K do
        begin s(k) := DELTA × abs (psi(k));
          if s(k) = 0.0 then s(k) := DELTA;
        end;
```

This change sets the step size for each variable to *DELTA* times the magnitude of the starting value, or if the starting value is 0.0 the step size is set equal to *DELTA*. Thus *DELTA* is the fraction of the original value of each variable to be used as an initial step size. Subsequent reductions in step size are handled correctly without further modifications to the procedure.

As an example of the usefulness of the above modification, consider the function

$$f(X_1, X_2, X_3) = (X_1 - 0.01)^2 + (X_2 - 1.0)^2 + (X_3 - 100.0)^2$$

with a minimum at (0.01, 1.0, 100.0). The following table shows the results of using direct search on this function with and without the modified step size. The results were computed on an IBM 360/75 computer using single precision with rho = 0.1, *delta* 0.001, *DELTA* = 0.2 for the modified step size (giving 20 percent of initial value for initial step size) and *DELTA* = [average magnitude of initial guesses for the variables] for the algorithm as published.

TABLE I. $f = (X_1 - 0.01)^2 + (X_2 - 1.0)^2 + (X_3 - 100.0)^2$

	DELTA	*Number of function evaluations*	*Minimum value of f*	*Final values of the variables*		
				X_1	X_2	X_3
For initial values of (0.0, 0.0, 200.0):						
Direct search	66.6667	153	0.841×10^{-7}	0.00999995	0.999995	100.000
Modified direct search	.2	112	0.597×10^{-7}	0.00999998	0.999990	100.000
For initial values of (0.05, 5.0, 500.0):						
Direct search	168.36	174	0.934×10^{-7}	0.0100263	0.998958	99.9999
Modified direct search	.2	75	0.559×10^{-6}	0.00999988	0.999998	99.9992

Note that the modified method will tend to yield the same relative accuracy for each parameter, whereas with a fixed step size direct search will tend to give the same absolute accuracy for all parameters. In most cases a relative accuracy is probably more desirable than an absolute accuracy.

REFERENCES
1. BELL, M., AND PIKE, M. C. Remark on algorithm 178. *Comm* ACM 9 (Sept. 1966), 684.

REMARK ON ALGORITHM 308 [G6]
GEYERATION OF PERMUTATIONS IN PSEUDO-LEXICOGRAPHIC ORDER [R. J. Ord-Smith, *Comm. ACM 10* (July 1967), 452]

R. J. ORD-SMITH (Recd. 21 May 1969)
Computing Laboratory, University of Bradford, England

KEY WORDS AND PHRASES: permutations, lexicographic order, lexicographic generation, permutation generation
CR CATEGORIES: 5.39

Following the construction of the very fast lexicographic permutation Algorithm 323 [1] it has become clear that the permutation sequence generated by the Algorithm 308 can be obtained more quickly. In fact, replacement of

```
trstart:m := q[k];  t := x[m];  x[m] := x[k];  x[k] := t;
  q[k] := m + 1;  k := k - 1;
```

by

```
trstart:  q[k] := q[k] + 1;
```

in Algorithm 323 produces the *ECONOPERM* sequence of Algorithm 308.

The times are as follows on an ICT 1905, in seconds

	t_7	t_8
Algorithm 323	6	47
New *ECONOPERM*	5.9	45
Old *ECONOPERM*	6.2	50.6

REFERENCE:
1. ORD-SMITH, R. J. Algorithm 323: Generation of permutations in lexicographic order. Comm. *ACM 11* (Feb. 1968), 117.