Algorithms

L. D. FOSDICK, Editor

ALGORITHM 364
COLORING POLYGONAL REGIONS [Z]

ROBERT G. HERRIOT (Recd. 30 Jan. 1967, 31 Oct. 1968 and 2 July 1969)
University of Wisconsin, Computer Science Department, Madison, WI 53706

KEY WORDS AND PHRASES: coloring polygonal regions, coloring planar surfaces, drawing pictures, shading enclosed regions
CR CATEGORIES: 4.9

```
procedure drawarea (x, y, firstpoint, lastpoint, section, numrows,
    numseats, regcolor, paintflag, paintcolor, sgn, dir, edge);
  value firstpoint, lastpoint, numrows, numseats, regcolor,
    paintflag, paintcolor, sgn, dir, edge;
  integer firstpoint, lastpoint, numrows, numseats, regcolor,
    paintcolor, sgn;
  real edge;
  Boolean paintflag, dir;
  real array x, y;
  integer array section;
```

comment This procedure is a part of a large program which produces the card stunts for the Stanford University football game half-times. The initial development was done by L. Breed, L. Tesler, and J. Sauter. The author (a Stanford student at the time) made many further developments on this program which included producing an algorithm for coloring in polygonal regions. Prior to the development of this algorithm, there were many cases which did not work. The larger program takes as input an English description of the stunts and produces as output an image of each flip (similar to a frame in a movie film), as a rectangle that has 45 rows with 77 seats in each row. The main program, which will be considered the driver program for the purpose of the procedure *drawarea*, does all of the handling of the definition of regions and also the printing of the images. It should be mentioned that the procedure *drawarea* in the actual program is just part of a larger procedure and that all of the parameters are global in order to increase efficiency. The purpose of *drawarea* is to take the current regions and draw them in the two-dimensional array *section*, which is to be declared as *section* [1: *numrows*, 1: *numseats*] (the array is 45 by 77 for Stanford). Each completed picture in *section* is then printed and also written out on tape. Another program later takes this tape and processes it to produce an instruction card for each student holding a set of colored cards in the rooters section.

The larger program allows objects of any shape to be defined by a series of *x*, *y*-coordinates. It will accept a series of points which are given an identifying name by the user and which can then be used as (1) a group of points, (2) a series of connected line segments, (3) a polygonal region enclosed by the points (with the first and last point connected by a straight line). It also allows ellipses to be defined. Once an object is defined, it can be expanded and contracted in size, rotated about any fixed point, or moved anywhere, including all or partially out of sight. As soon as all objects are in place, the user can ask that an image of the picture be made. Except for polygonal regions, producing the image of these objects is trivial. The procedure *drawarea* is the routine which places the polygonal regions in the array section.

The array *section* is presumed to have a background color associated with it. All objects, which also have an associated color, are then drawn into the array in a specified order so that the objects which are to be superimposed over other objects are drawn last. The procedure *drawarea* takes the coordinates of the point (which may not be integral) from arrays *x* and *y* with subscript values ranging from *firstpoint* to *lastpoint* and decides which seats in array *section* will form the left and right boundaries of this new region. After the boundary is determined, the interior must be colored in. The algorithm colors the region by taking each row and then examining each seat from left to right. For optimization, only the area of a minimal circumscribing rectangle is examined. At the beginning of each row the variable *count* is set to *leftcount* [*row*, 0]−*rightcount* [*row*, 0], which will be zero unless the object is partially out of sight on the left. Then as long as *count* remains zero, the seat is on the exterior and is not colored. As each seat is encountered, *leftcount* [*row*, *seat*] is added to *count*. When *count* is positive, the seat is in the interior or on a boundary and is colored. After each seat is processed, *rightcount* [*row*, *seat*] is subtracted from *count*. When *count* returns to zero, the seat is an exterior seat and is not colored. In any row it is possible to have the color turned on and off several times. Arrays *leftcount* and *rightcount* contain twice the number of left and right boundaries which pass through each individual seat. These two arrays solve the problem created by having several boundaries passing through one seat.

A further complication to the routine is added by allowing a region to be gradually changing color. Thus each region always has a color (*regcolor*) associated with it, and if the region is being swept with a new color, then *paintflag* is **true** and *paintcolor*, *sgn*, *dir*, and *edge* are used to determine the section of the region which is to be of the new color (*paintcolor*). The roles of the parameters for painting are: *sgn* and *dir* indicate the direction in which the imaginary paintbrush is moving. *dir* = **true** means the direction is horizontal and *dir* = **false** means vertical. *sgn* = −1 means the direction is left or down and *sgn* = 1 means the direction is right or up. *edge* is the row or seat (column) where the new color (*paintcolor*) ends and the old color (*regcolor*) begins. The driver program is expected to change *edge* with each new image so that the region looks as if it is being swept by a new color.

A related algorithm which determines whether a point is inside a polygon is presented in Algorithm 112 [1, 2].

REFERENCES:

1. HACKER, RICHARD. Certification of Algorithm 112, Position of point relative to polygon. *Comm. ACM 5* (Dec. 1962), 606.
2. SHIMRAT, M. Algorithm 112, Position of point relative to polygon. *Comm. ACM 5* (Aug. 1962), 434;

```
begin
  integer row, seat, toprow, rightseat, rit, lef, top, bot, iox, ioy,
    inx, iny, sdx, sdy, j, ix, iy, count;
  real ox, oy, nx, ny, dx, dy, dxdy, const;
  integer array leftcount, rightcount [0: numrows+1,
    0: numseats+1];
  integer procedure max(x, y); value x, y; integer x, y;
    max := if x ≥ y then x else y;
  integer procedure min(x, y); value x, y; integer x, y;
    min := if x ≤ y then x else y;
  toprow := numrows + 1;
  rightseat := numseats + 1;
  for row := 0 step 1 until toprow do
    for seat := 0 step 1 until rightseat do
      leftcount [row, seat] := rightcount [row, seat] := 0;
  ox := x[lastpoint]; rit := left := iox := ox;
  oy := y[lastpoint]; top := bot := ioy := oy;
  comment Draw the boundary by iterating through the points;
  for j := firstpoint step 1 until lastpoint do
```

```
begin
  nx := x[j];  inx := nx;
  ny := y[j];  iny := ny;
  dx := nx − ox;
  dy := ny − oy;
  sdx := if dx < 0 then −1 else 1;
  sdy := if dy < 0 then −1 else 1;
  if ioy = iny then
  begin
    comment The line is horizontal, or almost so;
    comment min and max keep the point in the section;
    row := max(min(ioy, toprow), 0);
    seat := max(min(max(iox, inx), rightseat), 0);
    rightcount [row, seat] := rightcount [row, seat] + 1;
    seat := max(min(min(iox, inx), rightseat), 0);
    leftcount [row, seat] := leftcount [row, seat] + 1;
  end horizontal line
  else
  begin
    comment The line is not horizontal;
    dxdy := dx/dy;
    const := if abs(dx) ≤ abs(dy)
             then ox − dxdy × oy
             else ox − dxdy × (oy − sdx/2) − sdy/2;
    comment Draw line between two points by stepping
      through each row and determining which seat should be
      marked as the boundary;
    for iy := ioy step sdy until iny do
    begin
      ix := dxdy × iy + const;
      row := max(min(iy, toprow), 0);
      seat := max(min(ix, rightseat), 0);
      comment Because end points are each processed twice,
        we add only 1 to them instead of the usual 2;
      if dy > 0 then
      begin
        comment Boundary on right side of area;
        rightcount[row,seat] := rightcount[row,seat]
          + (if iy=ioy∨iy=iny then 1 else 2)
      end
      else
      begin
        comment Boundary on left side of area;
        leftcount[row,seat] := leftcount[row,seat]
          + (if iy=ioy∨iy=iny then 1 else 2)
      end
    end drawing of line;
  end sloping line;
  comment Move on to next line segment;
  ox := nx;    iox := ox;
  oy := ny;    ioy := oy;
  comment Find rectangle which circumscribes the area;
  if rit < iox then rit := iox
  else if lef > iox then lef := iox;
  if top < ioy then top := ioy
  else if bot > ioy then bot := ioy;
end bordering area;
lef := max(1, lef);  rit := min(rit, numseats);
bot := max(1, bot);  top := min(top, numrows);
comment Color the area. It is only necessary to look within
  the circumscribing rectangle;
for row := bot step 1 until top do
begin
  count := leftcount [row, 0] − rightcount [row, 0];
  for seat := lef step 1 until rit do
  begin
    count := count + leftcount [row, seat];
    if count > 0 then
      section [row, seat] := if paintflag then
            (if sgn×((if dir then seat else row)−edge) > 0
              then
            regcolor
            else paintcolor)
          else regcolor;
    count := count − rightcount [row, seat];
  end coloring of one seat;
end coloring of one row;
end drawarea;
```

The following algorithm by H. Bach relates to the paper by the same author in the Numerical Analysis department of this issue on pages 675-677.

This concurrent publication in Communications follows a policy announced by the Editors of the two departments in the March 1967 issue.

ALGORITHM 365
COMPLEX ROOT FINDING [C5]

H. Bach

Laboratory of Electromagnetic Theory, Technical University of Denmark, Lyngby, Denmark

(Recd. 18 Apr. 1968 and 15 July 1969)

KEY WORDS AND PHRASES: downhill method, complex relaxation method, complex iteration, complex equation, transcendental complex equation, algebraic complex equation

CR CATEGORIES: 5.15

Comment. The present subroutine determines, within a certain region, a root of a complex transcendental equation $f(z) = 0$, on which the only restriction is that the function $w = f(z)$ must be analytic in the region considered. The iterative method used, the downhill method, was originally described in [2] and is discussed and modified in [1].

The program uses a complex function subprogram FUNC(Z) for the computation of $f(z)$. From a given complex starting point ZS, the iteration is performed in steps of initial length HS. The iterations stop at the root approximation ZE when either the function value DE at the end point is less than the prescribed minimum deviation DM or when the step length HE has become less than the prescribed minimum step length HM. For reference, the subroutine also returns DS, the function value at the starting point ZS, and N, the number of iterations used. There are thus four input parameters, namely the starting point ZS, the initial step length HS, the minimum step length HM, and the minimum deviation DM.

Acknowledgment. Thanks are due to Mr. Frank Jensen, M.Sc., who helped in the testing of this algorithm.

References:

1. Bach, H. On the downhill method. *Comm. ACM 12* (Dec. 1969) 675–677.
2. Ward, J. A. The downhill method of solving $f(z) = 0$. *J. ACM 4* (Mar. 1957), 148–150.


```
      SUBROUTINE CRF(ZS,HS,HM,DM,FUNC,DS,ZE,HE,DE,N)
C
C   THE SUBROUTINE DETERMINES A ROOT OF A TRANSCEN-
C   DENTAL COMPLEX EQUATION F(Z)=0 BY STEP-WISE ITE-
C   RATION.(THE DOWN HILL METHOD)
C
C   INPUT-PARAMETERS.
C
C   ZS = START VALUE OF Z.(COMPLEX)
C   HS = LENGTH OF STEP AT START.
C   HM = MINIMUM LENGTH OF STEP.
C   DM = MINIMUM DEVIATION.
C
```

```
C   SUBPROGRAM.
C
C   FUNC(Z), A COMPLEX FUNCTION SUBPROGRAM FOR THE
C   CALCULATION OF THE VALUE OF F(Z) FOR A COMPLEX
C   ARGUMENT Z.
C
C   OUTPUT-PARAMETERS.
C
C   DS = CABS(FUNC(ZS))=DEVIATION AT START.
C   ZE = END VALUE OF Z.(COMPLEX)
C   HE = LENGTH OF STEP AT END.
C   DE = CABS(FUNC(ZE))=DEVIATION AT END.
C   N  = NUMBER OF ITERATIONS.
C
C   RESTRICTIONS.
C
C   THE FUNCTION W=F(Z) MUST BE ANALYTICAL IN THE
C   REGION WHERE ROOTS ARE SOUGHT.
C
      REAL W(3)
      COMPLEX ZO,ZS,ZE,ZD,ZZ,Z(3),CW,A,V,U(7),FUNC
      U(1)=(1.,0.)
      U(2)=(0.8660254,0.5000000)
      U(3)=(0.0000000,1.0000000)
      U(4)=(0.9659258,0.2588190)
      U(5)=(0.7071068,0.7071068)
      U(6)=(0.2588190,0.9659258)
      U(7)=(-0.2588190,0.9659258)
      H=HS
      ZO=ZS
      N=0
C
C   CALCULATION OF DS.
C
      CW=FUNC(ZO)
      WO=ABS(REAL(CW))+ABS(AIMAG(CW))
      DS=WO
      IF(WO-DM) 18,18,1
    1 K=1
      I=0
    2 V=(-1.,0.)
C
C   EQUILATERAL TRIANGULAR WALK PATTERN.
C
    3 A=(-0.5,0.866)
C
C   CALCULATION OF DEVIATIONS W IN THE NEW TEST POINTS.
C
    4 Z(1)=ZO+H*V*A
      CW=FUNC(Z(1))
      W(1)=ABS(REAL(CW))+ABS(AIMAG(CW))
      Z(2)=ZO+H*V
      CW=FUNC(Z(2))
      W(2)=ABS(REAL(CW))+ABS(AIMAG(CW))
      Z(3)=ZO+H*CONJG(A)*V
      CW=FUNC(Z(3))
      W(3)=ABS(REAL(CW))+ABS(AIMAG(CW))
      N=N+1
C
C   DETERMINATION OF W(NR), THE SMALLEST OF W(I).
C
      IF(W(1)-W(3)) 5,5,6
    5 IF(W(1)-W(2)) 7,8,8
    6 IF(W(2)-W(3)) 8,8,9
    7 NR=1
      GOTO 10
    8 NR=2
      GOTO 10
    9 NR=3
   10 IF(WO-W(NR)) 11,12,12
   11 GOTO (13,14,15),K
   12 K=1
      I=0
C
C   FORWARD DIRECTED WALK PATTERN.
C
      A=(0.707,0.707)
      V=(Z(NR)-ZO)/H
      WO=W(NR)
      ZO=Z(NR)
      IF(WO-DM) 18,18,4
   13 K=2
C   REDUCTION OF STEP LENGTH.
C
      IF(H.LT.HM) GOTO 18
      H=H*0.25
      GOTO 3
   14 K=3
C
C   RESTORATION OF STEP LENGTH.
C
      H=H*4.
      GOTO 2
   15 I=I+1
C
C   ROTATION OF WALK PATTERN.
C
      IF(I-7) 16,16,17
   16 V=U(I)
      GOTO 3
C
C   REDUCTION OF STEP LENGTH.
C
   17 IF(H.LT.HM) GOTO 18
      H=H*0.25
      I=0
      GOTO 2
   18 ZE=ZO
      HE=H
      DE=WO
      RETURN
      END
```

ALGORITHM 366
REGRESSION USING CERTAIN DIRECT PRODUCT MATRICES [G2]

P. J. Claringbold (Recd. 10 May 1968 and 8 July 1969)
Division of Animal Genetics, C.S.I.R.O., P.O. Box 90, Epping, N.S.W., Australia, 2121

KEY WORDS AND PHRASES: analysis of variance, analysis of covariance, regression analysis, experimental design, matrix direct product, projection operator, orthogonal matrix
CR CATEGORIES: 5.14, 5.5

procedure *regressor* (*vec, kobs, levs, code, kfac, nfac, ndf*);
value *nfac*;
integer *kobs, levs, code, kfac, nfac, ndf*;
real *vec*;
comment The mathematical basis of the algorithm which forms the kernel of a very general analysis of variance and covariance procedure (Algorithm 367) is set out in [5, 6]. An overwhelming majority of the experimental designs in [2] may be analyzed in this way. Statistical nomenclature is given in parentheses.

A vector *vec*, of *nobs* elements (*observations*) traced by *kobs*, is replaced by $ndf \leq nobs$ elements (*regression coefficients*) obtained by the matrix product $C^T \cdot vec$, since the matrix is semiorthogonal. The number of initial elements is implied as the product of the *nfac* values of the variable *levs* which are traced by *kfac*. Values of *code*, similarly traced, specify matrices which enter a direct product [4] to form the transforming matrix C^T (*independent variates transposed*). As *code* takes the values 0, 1, or 2, the matrices selected are I, j, or V, i.e. the unit matrix of order *levs*, the unit vector of *levs* equal elements, or a matrix made up of $levs - 1$ mutually orthogonal unit vectors which are also orthogonal to the previous vector ($V^T \cdot j = 0$ and $V^T \cdot V = I$). A direct product of the transposes of the selected matrices forms the transforming matrix. An example of an actual call is shown to illustrate tracing: *example*: *regressor* (*vec*[*kobs*], *kobs*, *levs*[*kfac*], *code*[*kfac*], *kfac*, *nfac*, *ndf*).

The squared length of the resultant vector (*sum of squares on ndf degrees of freedom*) is equal to the squared length of the projection of the original vector in the subspace spanned by an idempotent symmetric matrix (*idix*) P. Eigenvectors associated with unit eigenvalues of this projection operator [1] comprise the rows of the transforming matrix.

$$l^2 = vec^T \cdot P \cdot vec = vec^T \cdot C \cdot C^T \cdot vec. \tag{1}$$

The cosine of the angle between two similarly transformed vectors (*correlation coefficient*) is obtained in an analogous manner from a scalar product (*sum of cross products*).

$$l_{vec} l_{wec} \cos(\theta) = vec^T \cdot P \cdot wec. \tag{2}$$

Prior evaluation of direct products is very wasteful of operations [3], and use is made of an identity which involves ordinary $(\cdot)$ and direct $(\times)$ products:

$$(A \times B \times C) \cdot y = (A \times I \times I) \cdot (I \times B \times I) \cdot (I \times I \times C) \cdot y. \tag{3}$$

Although shown for a triple product the identity obviously holds for any number of factors. The identity, however, is only valid for square matrices and the rectangular j or V factors must therefore be bordered by zeros to satisfy. In the algorithm multiplication by these zeros is bypassed, and after each transformation the vector is packed ready for the next.

Another identity:

$$(A \times B) \cdot (C \times D) = (A \cdot C) \times (B \cdot D), \tag{4}$$

implies that the ordinary products in (3) may be taken in any order, since the direct product factors commute. The transformations should therefore be taken in the order which achieves the largest reduction in the number of elements. Since *j-factors* achieve a reduction in the ratio *levs*:1, while *V-factors* merely achieve *levs*:*levs* − 1, the transformations are arranged in descending order of levels for *j-factors* followed by an ascending order of levels for *V-factors*. Transformations requiring the unit matrix are, of course, skipped.

REFERENCES:

1. BANERJEE, K. S. A note on idempotent matrices. *Ann. Math. Statist. 35* (1964), 880–882.
2. COCHRAN, W. G. and COX, GERTRUDE M. *Experimental Designs* (2 Ed.) Wiley, New York, 1957.
3. GOOD, I. J. The interaction algorithm and practical Fourier analysis. *J. Roy. Statist. Soc.* {B} *20* (1958), 361–373.
4. MARCUS, M. Basic theorems in matrix theory. *Nat. Bur. Standards Appl. Mathl Ser. 57* (1960), Washington, D.C.
5. NELDER, J. A. The analysis of randomised experiments with orthogonal block structure. I. Block structure and the null analysis of variance. *Proc. Roy. Soc.* {A} *283* (1965), 147–162.
6. NELDER, J. A. The analysis of randomised experiments with orthogonal block structure. II. Treatment structure and the general analysis of variance. *Proc. Roy. Soc.* {A} *283* (1965), 163–178;


```
begin
  integer ifac, jgo, nlft, nrgt, jfac, jump, ilft, irgt, jumphold, ilev,
    jumpo, jumper, iup, idown, nlev, maxp;
  real x, v;
  integer array ranks[1:nfac];
  maxp := ndf := 1;
  for kfac := 1 step 1 until nfac do
  begin
    comment Transmit levels and determine largest factor;
    ranks[kfac] := nlev := levs; ndf := ndf×nlev;
    if nlev > maxp then maxp := nlev
  end with degrees of freedom set in null case;
  maxp := −(maxp+1);
  for jgo := 1, 2 do
  begin
    comment Averaging before differencing transformations;
mfac:
    begin
      comment Search for best remaining factor;
      nlev := maxp; ifac := 0;
      for kfac := 1 step 1 until nfac do
      begin
        ilev := (3−2×jgo) × ranks[kfac];
        if code = jgo ∧ ranks[kfac] = levs ∧ ilev > nlev then
        begin
          nlev := ilev; ifac := kfac
        end if a better factor
      end search;
      if ifac > 0 then
      begin
        comment Process a factor;
        kfac := ifac; nlev := levs; nlft := nrgt := 1;
        for jfac := 1 step 1 until nfac do
          if ifac ≠ jfac then
          begin
            comment Determine orders of unit matrices to left
              and right;
            if jfac < ifac then nlft := nlft × ranks[jfac]
            else nrgt := nrgt × ranks[jfac]
          end products;
        begin
          comment Evaluate normalization constants;
          array root[jgo : if jgo=1 then 1 else nlev];
          if jgo = 1 then root[1] := sqrt(1/nlev)
          else
          for ilev := 2 step 1 until nlev do
            root[ilev] := sqrt(1/(ilev×(ilev−1)));
          comment Begin transformation of vector;
          jump := 0;
          comment Loop over all combinations to the left;
          for ilft := 1 step 1 until nlft do
          begin
            jump := jump + 1;
            comment Loop over all combinations to the right;
            for irgt := 1 step 1 until nrgt do
            begin
              jumphold := jump; jump := jump − nrgt; x := 0;
              comment Loop over active factor;
              for ilev := 1 step 1 until nlev do
              begin
                comment Form sum;
                jumpo := jump; kobs := jump := jump + nrgt;
                if jgo = 2 ∧ ilev > 1 then
                begin
                  comment Form difference when appropriate;
                  v := vec; kobs := jumpo;
                  vec := (x−(ilev−1)×v)×root[ilev]
                end now do sum;
                kobs := jump; x := x + vec
              end sum and difference loop;
              if jgo = 1 then
              begin
                comment Insert normalized average;
                kobs := jumphold; vec := x × root[1]
              end insertion;
              jumper := jump; jump := jumphold + 1
            end loop over all combinations to the right;
            jump := jumper;
          end loop over all combinations to the left
        end block;
        iup := nrgt × nlev; idown := if jgo = 1 then nrgt else
          iup − nrgt;
        for ilft := 2 step 1 until nlft do
        begin
          comment Compact vector;
          for irgt := 1 step 1 until nrgt do
          for ilev := 2 step 1 until nlev do
            if ilev < 3 ∨ jgo = 2 then
            begin
              kobs := iup := iup + 1; v := vec;
              kobs := idown := idown + 1; vec := v
            end within block moves;
          iup := if jgo = 1 then iup + (nlev−1) × nrgt else
            iup + nrgt
        end block moves;
        comment Adjust dimensions of pseudoarray;
        ranks[ifac] := if jgo = 1 then 1 else nlev − 1;
        ndf := idown;
        go to mfac
      end
      else go to end jgo
    end labeled compound statement;
end jgo:
  end loop over factor types
end regressor
```

ALGORITHM 367
ANALYSIS OF VARIANCE FOR BALANCED EXPERIMENTS [G2]

P. J. CLARINGBOLD (Recd. 27 May 1968 and 8 July 1969)
Division of Animal Genetics, C.S.I.R.O., P.O. Box 90, Epping, N.S.W., Australia, 2121

KEY WORDS AND PHRASES: analysis of variance, analysis of covariance, regression analysis, experimental design, balanced experiment, missing data, interblock estimate, intrablock estimate
CR CATEGORIES: 5.14, 5.5

integer procedure *balanced anova* (*y, missing y, x, fixed effect, estimate, error level, error code, all y, all x, length y, length x, pooled beta, se beta, normalized beta, error, df total, df error, tolcor, tolength, tolmpss, ispace, nspace, ires, jres, nres, itrt, ntrt, iobs, nobs, ifac, nfac, max cycle, check diagonality, projector, putpy, getpy, putpx, getpx*);

value *tolcor, tolength, tolmpss, nspace, nres, ntrt, nobs, nfac, max cycle, check diagonality*;

real *y, x, all y, all x, length y, length x, pooled beta, se beta, normalized beta, error, tolcor, tolength, tolmpss*;

integer *error level, error code, df total, df error, ispace, nspace, ires, jres, nres, itrt, ntrt, iobs, nobs, ifac, nfac, max cycle*;

Boolean *missing y, fixed effect, estimate, check diagonality*;

procedure *projector, putpy, getpy, putpx, getpx*;

comment The algorithm provides analyses of variance, covariance, and regression for data collected according to a wide variety of experimental designs. The vector of elements comprising either a response (*y or dependent*) or a treatment (*x or independent*) variate forms a conceptual complete array of *nfac* dimensions. The implied subscripts are a set of discrete variables which define an error classification. Designs of this type include the *fully randomized, randomized block, incomplete block, split (to any order) plot, Latin (and higher) squares, lattices*, et cetera, and make up the overwhelming majority in use [3]. By means of an appropriate transformation the frequency data of contingency tables may be processed to provide partitions of chi-square [1]. A comprehensive account of the mathematical basis is given in [4, 5].

In this implementation extensive use is made of the *call-by-name* facility so that generators and routines involving auxiliary store may freely be used for all input variables. Usually data sets are quite small and storage of intermediate quantities within the immediate access store is possible. In the following notes on the formal parameters relevant tracer variables are shown in brackets. An arrow (→) indicates that the variable is used only as a source of information.

balanced anova: If the projection of *x-variate* numbered *jtrt* has a correlation coefficient exceeding *tolcor* with the projection of *x-variate* numbered *ktrt* in subspace *ispace* of the design, then abnormal termination is forced with *balanced anova* $= 10^6 \times$ *ispace* $+ 10^3 \times$ *jtrt* $+$ *ktrt*. Zero is returned as the value of the procedure in the case of normal termination. Note that this time-consuming check of the *balance* of the treatment model with respect to the error model is only performed if *check diagonality* is set **true**.

y, missing y (*ires, iobs*) → : The *y-variate* generator or array must provide trial values, e.g. the average of present elements for the variate, for any missing data. These elements are flagged by **true** in the **Boolean** *missing y* which may take the form of an expression in terms of *ires, iobs*, and **integer** constants.

x (*itrt, iobs*) → : A complete specification of the orthogonal decomposition of the total sum of squares (and products) using polynomials or some other form of contrast representation is required. In the case of treatment classifications (for example *factorial experiment*) the *x-variate* values may be generated as a direct product (or as a selection of elements from such a matrix) of a number of small contrast matrices, i.e. orthogonal matrices with first column having elements greater than zero (usually constant).

fixed effect (*itrt, ispace*) → : By setting this variable **true** the flagged regression coefficients, i.e. *beta* number *itrt* in estimation subspace number *ispace*, are declared to be error free or invariants. In most practical cases this facility is only relevant to the constant term of the regression model.

estimate (*itrt, ispace*) → : By setting this variable **false** the flagged regression coefficients are declared to be zero and are not estimated in the indicated subspaces. Usually this facility is not required, and the constant **true** is used as actual parameter.

error level (*ifac*) → : The variable sets the number of levels of the error classifications. If it is assumed that the conceptual subscripts have unit lower bounds, then the upper bounds are set. Variates (traced by *iobs*) must be in lexical order by the implied subscripts, and use of a permutation array or function may be required to achieve this end.

error code (*ifac, ispace*) → : Error sources of variation (estimation or error subspaces) are specified by integer codes 0, 1, or 2. The codes could be generated by means of a procedure which interpreted a string of input characters denoting the *error structure* of the experimental design, see [4, 5]. A set of *nfac* integers specifies a projection operator which spans a subspace. The operator is formed as the direct product of (0) *identity matrix* I, (1) *averaging matrix* J, or (2) *differencing matrix* $K = I - J$. Every element of the averaging matrix is equal to the reciprocal of the order,

e.g.: 2, 0, 1, 2, 1 $\leftrightarrow K_1 \times I_2 \times J_3 \times K_4 \times J_5 = P_i$, *say*.

It is required that the error subspaces be mutually orthogonal, $P_iP_j = \delta_{ij}P_i$.

Code Sets for Some Common Designs

Design	Codes								P_1+P_2
Fully randomized	1	2							0
Randomized or incomplete block	11	21	02						01
Split plot	111	211	021	002					011
Split split plot	1111	2111	0211	0021	0002				0111
Square or rectangle	11	21	12	22					01
Revlicated square or rectangle	111	211	021	012	022				011
Three-way crossed error	111	211	121	112	221	212	122	222	011

In certain circumstances it may be desired to work *mod*$(J \times J \times \cdots \times J)$, that is the *y-variates* are adjusted to have zero mean. In this case the first code is omitted from the analysis. Usually it is convenient to pool the subspaces defined by $J \times J \times \cdots \times J$ and $K \times J \times \cdots \times J$ yielding (by addition) $I \times J \times \cdots \times J$, and if this is required the first two columns of the table are replaced by the rightmost auxiliary column.

all y [*ires*], *all x* [*itrt*], *length y* [*ires, ispace*], *length x* [*itrt, ispace*]: The lengths of the y, x, projected y, and projected x vectors are returned. Null variates (which have zero length) should be indicated in, or excluded from, analysis of variance tables (et cetera) derived from an activation of the procedure.

pooled beta, se beta [*ires, itrt*]: The weighted mean regression coefficient relating *y-variate* number *ires* to *x-variate* number *itrt* is returned in *pooled beta*, and the standard error of the estimate in *se beta*.

normalized beta [*ires, itrt, ispace*]: Within each subspace the regression coefficients are scaled so that it may be assumed that the sum of squares of each (nonnull) projected *x-variate* is unity. The *dyad* obtained by forming all pairwise products over the tracer *ires* (fixing the other tracers) is a single degree of freedom contribution due to treatment (*x-variate*) number *itrt* to subspace number *ispace* of the analysis of variance (and covariance if *nres* > 1).

error [*ires, jres, ispace*]: For each subspace an error covariance matrix is computed. This is the only variable bearing the tracer *jres* which is constrained so that *jres* $\leq$ *ires*. The calling program may make provision to pack the matrices in triangular form using a subscript function: *pack*[*ires*] $+$ *jres*, where *pack*[*ires*] $= (ires \times (ires - 1)) \div 2$.

df total, df error [*ispace*]: The variables return the total and error degrees of freedom for each subspace.

tolcor: If the activation calls for a check of the orthogonality of projected *x-variates*, then this constant sets the value of the correlation coefficient, which should not be exceeded in the test.

tolength: A projected vector is assigned zero length if the ratio of the computed length to that of the unprojected vector, multiplied by the square root of the ratio of the number of observations to degrees of freedom of the subspace, fails to exceed this criterion.

tolmpss: As a single measure of all missing data a sum of squares is computed. If the ratio of the absolute value of the difference between this sum and that of the previous iteration (or 0), to the current sum, fails to exceed this constant, no further iterations are made.

ispace, nspace, ires, jres, nres, itrt, ntrt, iobs, nobs, ifac, nfac: The identifiers with initial letter *i* or *j* are tracers mnemonically related to the remaining identifiers which define the number of subspaces, *y-variates*, *x-variates*, observations and error factors, respectively.

max cycle: An upper limit to the number of iterations required for the convergence of estimates of missing data is provided by this parameter.

check diagonality: If this parameter is **true** then the projected *x-variates* are checked for orthogonality. While computing time is saved by the opposite setting, incorrect results are computed if an invalid assumption of orthogonality is made.

projector: In order to compute the consequences of projection of variates, a choice between at least two procedures is made: $P \cdot x = C \cdot C^T \cdot x$ or $C^T \cdot x$. The idempotent symmetric projection operator P (see [4, 5]), or the rectangular matrix made up of the eigenvectors corresponding with unit eigenvalues (see [2]) is used. The second alternative is preferred since the transforming matrix is then thin, and Algorithm 366 is an implementation of this approach.

putpy, getpy, putpx, getpx: These procedures are concerned with the transmission of transformed variates between arrays internal to the algorithm and auxiliary store. While immediate access store may be used as auxiliary store with small problems, backing media such as magnetic drum, disk, or tape are required for large problems. The procedure *putpy* transmits all *nelm* elements of a transformed *y-variate* to auxiliary store, while *getpy* performs the reverse transmission. Similar actions on the *x-variates* are carried out by the other two procedures. All four routines have similar calling sequences: (*vec*[*ielm*], *ielm, nelm, ivar, ispace*), where *vec* identifies the vector to be moved, *ielm* traces the elements of the vector, *nelm* (returned by *projector*) specifies the number of elements to be moved, *ivar* gives the variate number, and *ispace* gives the subspace number. The elements to be moved are in the leading position in *vec*, and an appropriate instruction begins **for** *ielm* := 1 **step** 1 **until** *nelm* **do**. The last two formal parameters may be used to index an array listing the starting positions of the vectors in auxiliary storage.

References:

1. Claringbold, P. J. The use of orthogonal polynomials in the partition of chi-square. *Austral. J. Statist. 3* (1961), 48–63.
2. Claringbold, P. J. Algorithm 366. Regression using certain direct product matrices. *Comm. ACM 12* (Dec. 1969), 687–688.
3. Cochran, W. G., and Cox, Gertrude M. Experimental Designs (Ed. 2). Wiley, New York, 1957.
4. Nelder, J. A. The analysis of randomised experiments with orthogonal block structure. I. Block structure and the null analysis of variance. *Proc. Roy. Soc.* {*A*} *283* (1965), 147–162.
5. Nelder, J. A. The analysis of randomised experiments with orthogonal block structure. II. Treatment structure and the general analysis of variance. *Proc. Roy. Soc.* {*A*} *283* (1965), 163–178;


```
begin
  array yy, xx[1:nobs];  real s, t, v, ssmp;
  integer i cycle, ndf, jtrt, ktrt, kres, nelm, nmis;
  real procedure sigma (x, i, n);
    value n;
    real x;  integer i, n;
begin
  real xx;  xx := 0;
  for i := 1 step 1 until n do xx := xx + x;
  sigma := xx
end sigma;
comment  Count missing data items;
nmis := 0;  ssmp := 0;
for ires := 1 step 1 until nres do
for iobs := 1 step 1 until nobs do
  if missing y then nmis := nmis + 1;
begin
  comment  Get space for estimates of missing data;
  array y missing[1 : if nmis=0 then 1 else nmis];
  comment  Set up loop for missing data iteration;
  for i cycle := 1 step 1 until max cycle do
  begin
    comment  Analyze data in various error subspaces;
    for ispace := 1 step 1 until nspace do
    begin
      comment  Determine subspace degrees of freedom;
      if i cycle = 1 then
      begin
        comment  Only compute degrees of freedom once;
        ndf := 1;
        for ifac := 1 step 1 until nfac do
          ndf := ndf × (if error code=0 then error level
              else if error code=1 then 1 else error level−1);
        df total := ndf
      end
      else ndf := df total;
      comment  Project response vectors;
      nmis := 0;
      for ires := 1 step 1 until nres do
      begin
        comment  Fetch a vector, and possibly fit missing
          data;
        for iobs := 1 step 1 until nobs do
          if missing y then
          begin
            nmis := nmis + 1;
            if ispace = 1 then y missing[nmis] := if i cycle = 1
              then y
                else sigma (pooled beta×x, itrt, ntrt);
            yy[iobs] := y missing[nmis]
          end
          else yy[iobs] := y;
        if ispace = 1 then all y := sqrt(sigma(yy[iobs]↑2, iobs,
          nobs));
        projector(yy[iobs], iobs, error level, error code, ifac, nfac,
          nelm);
        jres := ires;
        error := sigma(yy[iobs]↑2, iobs, nelm);
        length y := if sqrt((error×nobs)/ndf)/all y > tolength
            then  sqrt(error)  else  0;
        putpy(yy[iobs], iobs, nelm, jres, ispace);
        for jres := 1 step 1 until ires − 1 do
        begin
          comment  Determine sums of cross products;
          getpy(xx[iobs], iobs, nelm, jres, ispace);
          error := sigma(yy[iobs]×xx[iobs], iobs, nelm)
        end cross products
      end dependent variates;
      comment  In the first cycle project treatment vectors;
      if i cycle = 1 then
      for jtrt := 1 step 1 until ntrt do
        if estimate then
        begin
          comment Only work on variates included in regres-
            sion;
```

```
    itrt := jtrt;
    for iobs := 1 step 1 until nobs do xx[iobs] := x;
    if ispace = 1 then all x := sqrt(sigma(xx[iobs] ↑ 2,
      iobs, nobs));
    projector(xx[iobs], iobs, error level, error code, ifac, nfac,
      nelm);
    t := sigma(xx[iobs] ↑ 2, iobs, nelm);
    s := length x := if sqrt((t×nobs)/ndf)/all x > tolength
        then sqrt(t) else 0;
    if s > 0 then
    begin
      comment Null variates are skipped;
      putpx(xx[iobs], iobs, nelm, itrt, ispace);
      if check diagonality then
      for ktrt := 1 step 1 until jtrt − 1 do
        if estimate then
        begin
          comment Orthogonality checked for variates
            in regression;
          itrt := ktrt;  v := length x;
          if v > 0 then
          begin
            comment Null variates are skipped;
            getpx(yy[iobs], iobs, nelm, itrt, ispace);
            if abs(sigma(xx[iobs]×yy[iobs], iobs, nelm))/
              (s×v) > tolcor then
            begin
              comment Force termination since ex-
                cessive correlation;
              balanced anova := 1000 × (1000×ispace+
                jtrt) + ktrt;
              go to exit
            end large correlation
          end if secondary variate has projection
        end secondary variate loop
    end if primary variate has projection
  end primary variate loop;
comment Compute normalized regression coefficients;
for itrt := 1 step 1 until ntrt do
  if length x > 0 ∧ estimate then
  begin
    comment Skip null or not in regression independent
      variates;
    ndf := ndf − 1;
    getpx(xx[iobs], iobs, nelm, itrt, ispace);
    for ires := 1 step 1 until nres do
      if length y > 0 then
      begin
        comment Skip null dependent variates;
        getpy(yy[iobs], iobs, nelm, ires, ispace);
        normalized beta := sigma(xx[iobs]×yy[iobs], iobs,
          nelm)/length x
      end
      else normalized beta := 0
  end
  else for ires := 1 step 1 until nres do normalized beta
      := 0;
df error := ndf;
comment Reduce sums of squares and products for
  regression;
for itrt := 1 step 1 until ntrt do
  if length x > 0 ∧ estimate then
  begin
    for kres := 1 step 1 until nres do
    for jres := 1 step 1 until kres do
    begin
      ires := jres;  s := normalized beta;
      ires := kres;  error := error − s × normalized beta
    end dyad reduction loops
  end normalized regression coefficient computation;
    comment Determine true regressions and information;
    for ires := 1 step 1 until nres do
    begin
      for jres := 1 step 1 until ires do
      error := if length y = 0 ∨ ndf = 0 then 0 else error/ndf;
      jres := ires;
      for itrt := 1 step 1 until ntrt do
      begin
        comment Clear areas at start;
        if ispace = 1 then pooled beta := se beta := 0;
        if estimate then
        begin
          comment Set information as unity for fixed
            effects;
          t := if fixed effect ∧ length x > 0 then 1 else
          if ndf = 0 then 0 else length x ↑ 2/ (if error=0
            then 1 else error);
          se beta := se beta + t;
          pooled beta := pooled beta + t × (if length x=0
            then 0 else normalized beta/length x)
        end of addition to pools
      end independent variate loop
    end dependent variate loop
  end error subspace loop;
  for ires := 1 step 1 until nres do
  for itrt := 1 step 1 until ntrt do
    if se beta > 0 then
    begin
      comment Compute weighted means and standard
        errors;
      pooled beta := pooled beta/se beta;
      se beta := sqrt(1/se beta)
    end average;
  if nmis > 0 then
  begin
    comment Check convergence of missing items;
    s := sigma(y missing[iobs] ↑ 2, iobs, nmis);
    if abs(s−ssmp)/s > tolmpss then ssmp := s
    else go to finish
  end missing data convergence test
 end cycle;
finish: balanced anova := 0;
exit:
  end block
end balanced anova
```

CERTIFICATION OF ALGORITHM 147 [S14]

PSIF [D. Amit, *Comm. ACM* 5 (Dec. 1962), 605]

Ronald G. Parsons* (Recd. 7 Dec. 1966 and 5 Aug. 1969)

Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94305

* Present address: Department of Physics, The University of Texas, Austin, TX 78712. Work supported by the US Atomic Energy Commission.

KEY WORDS AND PHRASES: gamma function, logarithmic derivative, factorial function, psi function

CR CATEGORIES: 5.12

The following errors were noted in this algorithm in addition to those noted by Thacher [2].

a. (4) in the comment should read "For $-x < -1$ we use: (4) $\Psi(-x) = \Psi(x-1) + \pi \cot(\pi x)$".

b. At the end of the first comment add: "Note that *psif*$(x) \equiv \Psi(x)$ is $\psi(x+1)$ as defined, for example, by Jahnke-Emde-Lösch" (see [1]).

c. The statement in the algorithm before the label *pos* should read: *psi* := *pei* × *cos* (*pei*×*x*)/*sin* (*pei*×*x*); These errors caused the procedure to give incorrect results for *psif*(*x*, *a*) for $x < -1$.
d. The arguments *tan* and *ln* should be deleted from the parameter list and **real procedure** *tan*, *ln*; should be deleted from the specification part of the procedure heading.

With these changes and those of Thacher, the procedure was translated into Burroughs B5500 extended ALGOL and run on the Stanford B5500. *psif*(*x*, *a*) was tabulated for $x = -2.9(0.1)5.0$ with $a = 8.0$ The results agreed with tabulated values to within $1/(240a^8)$.

REFERENCES:
1. JAHNKE-EMDE-LÖSCH. *Tables of Higher Functions* (6th Ed.). McGraw-Hill, New York, 1960.
2. THACHER, H. C., JR. Certification of Algorithm 147. *Comm. ACM 6* (Apr. 1963), 168.

CERTIFICATION OF ALGORITHM 229 [B1] ELEMENTARY FUNCTIONS BY CONTINUED FRACTIONS [James C. Morelock, *Comm. ACM 7* (May 1964), 296]

T. A. BRAY (Recd. 18 June 1964)
Boeing Scientific Research Laboratories, Seattle, WA 98124

KEY WORDS AND PHRASES: continued factions, Padé table
CR CATEGORIES: 5.19

Algorithm 229 was coded in FORTRAN II and run on the IBM 1620 computer for $x = 0.50$ and 0.75, for $n = 1, 2, 3, 4$, and for *parm* = 1, 2, 3, 4, 5, 6, 7.

For $x = 0.50$ my values agree with the author's up to $\pm 10^{-11}$.

For $x = 0.75$ and $n = 4$, my values of sin *x*, cos *x*, tan *x*, and exp *x* agree with tabulated values to within $\pm 10^{-11}$. For the same *x* and *n* my values of sinh *x*, and cosh *x*, and tanh *x* agree with tabulated values to within $\pm 10^{-10}$; no tables were available to check the 11th decimal.

REMARK ON ALGORITHM 300 [S22] COULOMB WAVE FUNCTIONS [J. H. Gunn, *Comm. ACM 10* (Apr. 1967), 244]; CERTIFICATION OF ALGORITHM 300 [K. S. Kölbig, *Comm. ACM 12* (May 1969), 279]

K. S. KÖLBIG (Recd. 14 Apr. 1969)
Data Handling Division, European Organization for Nuclear Research (CERN), 1211 Geneva 23, Switzerland

KEY WORDS AND PHRASES: Coulomb wave functions, wave functions, special functions, function evaluation
CR CATEGORIES: 5.12

Recently, Isacson [1] pointed out that the coefficient of $\eta^{-16/3}$ in the known asymptotic expansion for the irregular Coulomb wave function $G_0(\eta, \rho)$ on the transition line $\rho = 2\eta$ was erroneous.

In addition, he gave the expansions for F_0, G_0, F_0' and G_0' up to order η^{-6}, whereas the old expansions were given to order $\eta^{-16/3}$ only.

Therefore, and for reasons of speed, the relevant part of Algorithm 300 should be changed as follows:

```
begin comment G[0] and Gd[0] are calculated on the transition
  line for rhom = 2 × eta, ref. Isacson in remark;
  array et[1:12]; real et1;
  et[1] := eta ↑ (−2/3);
  for i := 2 step 1 until 12 do et[i] := et[1] × et[i−1];
  et1 := eta ↑ (1/6);
  G[0] := 1.223404016 × et1 × (1 + 0.04959570165 × et [2]
  −0.008888888889 × et [3] + 0.002455199181 × et [5]
  −0.0009108958061 × et [6] + 0.0008453619999 × et [8]
  −0.0004096926351 × et [9] + 0.0007116506205 × et [11]
  −0.00002439615603 × et [12]);
  Gd[0] := (−0.7078817734/et1) × (1 − 0.1728260369 × et [1]
  + 0.0003174603174 × et [3] − 0.003581214850 × et [4]
  +0.0003117824680 × et [6] − 0.0009073966427 × et [7]
  +0.0002128570749 × et [9] − 0.0006215584171 × et [10]
  +0.00003685244766 × et [12]);
  rhom := 2 × eta
end;
```

Furthermore, it was found in this connection that replacing the first line of the fourth *if* statement of the algorithm by

if *eta* < 4 ∧ *eta* < *rho*/2 **then**

gives, together with the above expansions, better results for $\rho = 2\eta$ in test (iii) and for $\rho = 3, \eta = 5$ in test (i) of the Certification.

The relevant statements in test (iii) of the Certification should therefore be replaced by the following ones:

F_0 − 1 unit for $\rho = 5$, $\rho = 6$, and $\rho = 8.5$.
F_0' − 1 unit for $\rho = 6$.
G_0 − 1 unit for $\rho = 5.5$, $\rho = 16$, and $\rho = 30$.
G_0' − 1 unit for $\rho = 5.5$.

REFERENCE:
1. ISACSON, T. Asymptotic expansion of Coulomb wave functions on the transition line. *BIT 8* (1968), 243–245.

REMARK ON ALGORITHM 341 [H] SOLUTION OF LINEAR PROGRAMS IN 0-1 VARIABLES BY IMPLICIT ENUMERATION [J. L. Byrne and L. G. Proll, *Comm. ACM 11* (Nov. 1968), 782]

L. G. PROLL (Recd. 5 Dec. 1968 and 18 Aug. 1969)
University of Southampton, Department of Mathematics, Hampshire, England

KEY WORDS AND PHRASES: linear programming, zero-one variables, partial enumeration
CR CATEGORIES: 5.41

The published algorithm contains an error in the assembly of the initial partial solution, *s*, if a priori information is given. In certain cases this can cause premature termination of the algorithm. The error may be corrected by replacing the following lines of the procedure body, from

```
  begin
    for j := 1 step 1 until n do
```

to

```
    e := n;  z := A[0, 0];  go to L0;
```

by

```
  begin
    e := 0;
    for j := 1 step 1 until n do
      if x[j] = 0 then v[j] := 0
    else
    begin
      e := e + 1;   s[e] := j;  v[j] := 3;
      for i := 1 step 1 until m do
        A[i, 0] := A[i, 0] + A[i, j];
    end;
    z := A[0, 0];  go to L0;
```

and by deleting the line

```
  if api then begin api := false;  go to L4 end;
```

Index by Subject to Algorithms, 1969

[Algorithms not in CACM have been included, when known to us.]

A1		REAL ARITHMETIC, NUMBER THEORY	
A1	356	PRIME NUMBER GENERATOR	10-69(563)
A1	357	PRIME NUMBER GENERATOR	10-69(563)
B1		TRIG AND INVERSE TRIG FUNCTIONS	
B1	229	ELEMENTARY FCNS.BY CONT.FRACT.	5-64(296),12-69(692)
C1		OPERATIONS ON POLYNOMIALS AND POWER SERIES	
C1	337	POLY.AND DERIV.BY HORNER SCHEME	9-68(633),1-69(39)
C2		ZEROS OF POLYNOMIALS	
C2	340	RT-SQUARING AND RESULTANT METH.	11-68(779),5-69(281)
C2		ROTATING-CROSS METHOD	BIT 1967(244)
C5		ZEROS OF ONE OR MORE TRANSCENDENTAL EQUATIONS	
C5	314	N FUNCTIONAL EQNS.IN N UNKNOWNS	11-67(726),1-69(38)
C5	315	DAMPED TAYLR SERIES-NONLIN.SYS.	11-67(726),9-69(513)
C5	365	COMPLEX ROOT FINDING	12-69(686)
C6		SUMMATION OF SERIES, CONVERGENCE ACCELERATION	
C6	255	FOURIER COEFFICIENTS	5-65(279),11-69(636)
C6	339	FAST FOURIER WITH ARB. FACTORS	11-68(776),3-69(187)
C6	345	CONVOLUTION BASED ON FFT	3-69(179),10-69(566)
D1		QUADRATURE	
D1	331	GAUSSIAN QUADRATURE FORMULAS	6-68(432),5-69(280)
D1	351	MODIFIED ROMBERG QUADRATURE	6-69(324)
D1	353	FILON QUADRATURE	8-69(457)
E1		INTERPOLATION	
E1		SPLINE INTERPOLN OF DEGREE 3	COMP.J.V12(198)
E1		QUINTIC SPLINES INTERPOLATION	COMP.J.V12(292)
E1		SMOOTH CURVE INTERPOLATION	BIT 1969(69)
E2		CURVE AND SURFACE FITTING	
E2	296	LEAST SQ.FIT-ORTHOG.POLYS.	2-67(87),6-67(377),
E2	296	11-69(636)	
E2		EXPONENTIALLY-DAMPED LINEAR FIT	COMP.J.V12(100)
E2		RATIONAL CHEBYSHEV APPROX.	NUM.MATH.V9(177)
E2		RATIONAL CHEBYSHEV APPROX.	NUM.MATH.V12(242)
E4		MINIMIZING OR MAXIMIZING A FUNCTION	
E4	178	DIRECT SEARCH	6-63(313),9-66(684),
E4	178	7-68(498),11-69(637),11-69(638)	
E4	251	FUNCTION MINIMIZATION	3-65(169),9-66(686),
E4	251	9-69(512)	
E4	315	MINIMIZING SUM OF SQUARES	11-67(726),9-69(513)
F1		MATRIX OPERATIONS, INCLUDING INVERSION	
F1	274	HILBERT DERIVED TEST MATRIX	1-66(11),7-69(407)
F1	298	SQ.RT.OF A POS.DEFINITE MATRIX	3-67(182),6-69(325)
F1	348	MTRX SCALING BY INTGR PROGRMING	4-69(212)
F1	358	SING.VAL.DECOMP.-COMPLEX MATRIX	10-69(564)
F1		TRIDIAG. OF SYM. BAND MATRIX	NUM.MATH.V12(234)
F1		SIM.RED.GEN.MTX.HESSENBERG FORM	NUM.MATH.V12(354)
F2		EIGENVALUES AND EIGENVECTORS OF MATRICES	
F2		MOD.LR.-CMPLX HESSENBERG MATRIX	NUM.MATH.V12(372)
F2		IMPLICIT QL ALGORITHM	NUM.MATH.V12(379)
F2		BALANCING A MATRIX	NUM.MATH.V13(298)
F4		SIMULTANEOUS LINEAR EQUATIONS	
F4	328	CHEBY SOLN-OVERDET LINEAR SYS	6-68(428),6-69(326)
F4	358	SING.VAL.DECOMP.-COMPLEX MATRIX	10-69(564)
F5		ORTHOGONALIZATION	
F5	358	SING.VAL.DECOMP.-COMPLEX MATRIX	10-69(564)
G1		SIMPLE CALCULATIONS ON STATISTICAL DATA	
G1	359	FACTORIAL ANALYSIS OF VARIANCE	11-69(631)
G2		CORRELATION AND REGRESSION ANALYSIS	
G2	366	REGRESSION-DIR.PROD.OF MATRICES	12-69(687)
G2	367	ANALYSIS OF VAR.-DIRECT EXPMT.	12-69(688)
G5		RANDOM NUMBER GENERATORS	
G5	334	NORMAL RANDOM DEVIATES	7-68(498),5-69(281)
G6		PERMUTATIONS AND COMBINATIONS	
G6	308	PERMUT.IN PSEUDOLEXIC.ORDER	7-67(452),11-69(638)
G6	329	DISTR OF INDISTINGUISHABLE OBJ	6-68(430),3-69(187)
G6	361	PERMANENT FNC.OF SQUARE MATRIX	11-69(634)
G6	362	RANDOM PERMUTATIONS	11-69(634)
G6		PARTITION FUNCTIONS-MOD(D)	BIT 1969(83)
H		OPERATIONS RESEARCH, GRAPH STRUCTURES	
H	333	MINIT ALGORITHM FOR LIN PROG	6-68(437),7-69(408)
H	341	LINEAR PGMS.IN 0-1 VARIABLES	11-68(782),12-69(692)
H	350	SIMPLEX METHOD-LU DECOMPOSITION	5-69(275)
H	354	SPANNING TREE GENERATOR	9-69(511)
H	360	SHORTEST PATH FOREST-TOPOL.ORD.	11-69(632)
J6		PLOTTING	
J6		A CURVE PLOTTING PROCEDURE	COMP.J.V12(291)
K2		RELOCATION	
K2	302	TRANSPOSE VECTOR STORED ARRAY	5-67(292),6-69(326)
M1		SORTING	
M1	347	SORT WITH MINIMAL STORAGE	3-69(185)
M1		A SEARCHING ALGORITHM	COMP.J.V12(101)
R2		SYMBOL MANIPULATION	
R2	268	ALGOL 60 REF.LANG.EDITOR	11-65(667),7-69(407)
S		APPROXIMATION OF SPECIAL FUNCTIONS	
S		FUNCTIONS ARE CLASSIFIED S01 TO S22, FOLLOWING	
S		FLETCHER-MILLER-ROSENHEAD, INDEX OF MATH. TABLES	
S14	147	DERIVATIVE OF GAMMA FUNCTION	12-62(605),4-63(168),
S14	147	12-69(691)	
S14	322	F-DISTRIBUTION	2-68(116),1-69(39)
S14	344	STUDENT'S T-DISTRIBUTION	1-69(37)
S14	346	F-TEST PROBABILITIES	3-69(184)
S14	349	POLYGAMMA FNS - ARB. PRECISION	4-69(213)
S15	304	NORMAL CURVE INTEGRAL	6-67(374),6-67(377),
S15	304	4-68(271),10-69(565)	
S15	363	COMPLEX ERROR FUNCTION	11-69(635)
S15		AREAS UNDER THE NORMAL CURVE	COMP.J.V12(197)
S16	47	ASSOCIATED LEGENDRE FUNCTION	4-61(178),8-63(446),
S16	47	11-69(635)	
S21	165	ELLIPTIC INTEGRAL	4-63(163),1-69(38)
S21		ELLIPTIC INTEGRALS-KINDS 1,2,3	NUM.MATH.V7(85),
S21		V7(353),V13(309)	
S22	292	REGULAR COULOMB WAVE FCNS.	11-66(793),5-69(278),
S22	292	5-69(280)	
S22	300	COULOMB WAVE FUNCTIONS	4-67(244),5-69(279),
S22	300	12-69(692)	
S22	352	CHAR.VALS.SLNS OF MATHIEU'S DE.	7-69(399)
Z		ALL OTHERS	
Z	355	GENERATE ISING CONFIGURATIONS	10-69(562)
Z	364	COLORING POLYGONAL REGIONS	12-69(685)

Key. The first column, A1, B1, C1, is the key to the classification system categories; second column: number of the algorithm if in *CACM*; third column: algorithm title; fourth column: month, year, pages (in parentheses) in *CACM* or reference elsewhere. This 1969 index is the first supplement to the Index by Subject to Algorithms, 1960–1968 (*Comm. ACM 11,* 12 (Dec. 1968), 827–830).