

ALGORITHM 142

TRIANGULAR REGRESSION

W. L. HAFLEY AND J. S. LEWIS

Aluminum Company of America, Pittsburgh, Penn.

```

procedure trireg (n, nob, dep, pmax);
  real pmax; integer n, nob, dep;
  comment trireg is a multiple regression procedure which
    develops and inverts only the upper triangular portion of a
    correlation matrix of order n. The i,jth (i ≤ j) matrix element
    is r(ci+j) where the c's are cram numbers (ref. Algorithm 67,
    J. Caffey, Comm. ACM 4, July 1961). dep < n dependent
    variables are regressed simultaneously. Read (u) is an input
    procedure for single elements. The input consists of nob ob-
    servations on n variables. The first dep variables are con-
    sidered dependent and the remaining n - dep are considered
    independent variables. Independent variables are dropped
    when the pivotal element exceeds pmax during the inversion.
    Total variable storage is 14 + 3n + n(n+1)/2;
  begin integer i1, i2, i3, c1, c2, c3, df; integer array c[1:n];
    real d, p, a; real array r[1:n(n+1)/2], v[1:n], m[1:n];
  initial: df := 0; for i1 := 1 step 1 until n do m[i1] := 0;
    for i1 := 1 step 1 until n(n+1)/2 do r[i1] := 0;
  input: for i1 := 1 step 1 until nob do
    begin for i2 := 1 step 1 until n do Read (v[i2]);
      c1 := 0; for i2 := 1 step 1 until n do
        begin d := v[i2]; m[i2] := m[i2] + d;
          for i3 := i2 step 1 until n do
            begin c1 := c1 + 1; r[c1] := r[c1] + v[i3] × d end
            end i3;
          end i2;
        end i1;
  correlation: c1 := 1; a := 1/nob; for i1 := 1 step 1 until n do
    begin v[i1] := 1/sqrt(r[c1] - (m[i1]2)/a);
      r[c1] := 1; c1 := c1 + n - i1
    end i1;
    c1 := 1; for i1 := 1 step 1 until n do
      begin d := a × m[i1]; p := v[i1]; c1 := c1 + 1;
        m[i1] := d;
        for i2 := i1 + 1 step 1 until n do
          begin r[c1] := (r[c1] - d × m[i2]) × v[i2] × p;
            end i2;
          end i1;
      comment variable i may be dropped from the
        regression by setting vi = 0 and df equal to the
        number of variables dropped;
  cram: i1 := -n; i2 := n + 1; for i3 := 1 step 1 until n do
    begin i1 := i1 + i2 - i3; c[i3] := i1
    end i1;
  inversion: for i1 := dep + 1 step 1 until n do
    begin c1 := c[i1]; if v[i1] ≠ 0 then
      begin p := 1/r[c1+i1]; if p > pmax then
        begin df := df + 1; go to YY end else
          begin r[c1+i1] := p; for i2 := 1 step 1
            until i1 - 1 do
            begin c2 := c[i2]; a := p × r[c2+i1];
              for i3 := i2 step 1 until n while i3 ≠ i1 do

```

```

        begin if i3 < i1 then
          begin c3 := c[i3]; d := r[c3+i1] end
          else d := -r[c1+i3];
            r[c2+i3] := r[c2+i3] + d × a
          end i3;
        end i2;
      for i2 := i1 + 1 step 1 until n do
        begin a := p × r[c1+i2]; c2 := c[i2];
          for i3 := i2 step 1 until n do
            r[c2+i3] := r[c2+i3] - a × r[c1+i3];
          end i3;
        ZZ: for i2 := 1 step 1 until i1 - 1 do
          begin c2 := c[i2+i1]; r[c2] := - p × r[c2]
          end i2;
        for i2 := c1 + i1 + 1 step 1 until n + c1 do
          r[i2] := p × r[i2]
        end
      end else
        YY: begin p := 0; r[c1+i1] := 0; go to ZZ end
  coeff: d := 1/(nob - n + dep - 1 + df); for i1 := 1 step 1 until
    dep do
      if v[i1] ≠ 0 then
        begin a := 0; p := 1/v[i1]; c1 := c[i1]; for i2 := dep
          + 1 step 1 until n do
            begin if r[i2] ≠ 0 then
              begin r[c1+i2] := -r[c1+i2] × v[i2] × p; a :=
                a + r[c1+i2] × m[i2]
              end
            end i2;
            v[i1] := (2 - r[c1+i1]) × d / (v[i1]2)
          comment: v[1:dep] now contains the mean square
            deviations from regressions for the dependent vari-
            ables. The coefficients of determination R2 may be
            obtained as r[c1+i1] - 1;
          r[c1+i1] := m[i1] - a else
            begin c1 := c[i1]; for i2 := c1 + i1 step 1 until
              c1 + n do r[i2] := 0
            end
          end
        end
      comment The r-array now contains the constants and coeffi-
        cients of regression, and the inverse of the correlation matrix of
        the independent variables that have been kept. The following
        example will help to locate the information in the r array.
        Example: n = 6 dep = 3

```

r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	b_{01}	b_{11}	b_{21}	b_{31}
r_{71}	r_{72}	r_{73}	r_{74}	r_{75}	r_{76}	b_{02}	b_{12}	b_{22}	b_{32}
r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	b_{03}	b_{13}	b_{23}	b_{33}
r_{16}	r_{17}	r_{18}	r_{19}	r_{20}	r_{21}	r^{11}	r^{12}	r^{13}	
						r^{22}	r^{23}		
							r^{33}		

The variances and covariances of the regression coefficients for the *j*th dependent variable can be determined by—

$$\text{Var} (b_{ij}) = r^{ii} \times v_j \times v_i^2$$

$$\text{Covar} (b_{ij}b_{kj}) = r^{ik} \times v_j \times v_i \times v_k;$$

end *trireg*

ALGORITHM 143

TREESORT 1

ARTHUR F. KAUPÉ, JR.

Westinghouse Electric Corp., Pittsburgh, Penn.

```

procedure TREESORT 1 (UNSORTED, n, SORTED, k);
  value n, k;
integer n, k; array UNSORTED, SORTED;
comment TREESORT 1 is a revision of TREESORT (AL-
  GORITHM 113) which requires neither the "packed" array m
  nor the machine procedures pack, left half, right half, and mini-
  mum. The identifier infinity is used as nonlocal real variable
  with value greater than any element of UNSORTED;
begin integer i, j; array m1 [1:2×n-1];
  integer array m2 [1:2×n-1];
procedure minimum; if m1 [2×i] ≤ m1 [2×i+1] then
  begin m1[i] := m1 [2×i]; m2[i] := m2 [2×i] end else
  begin m1[i] := m1 [2×i+1]; m2[i] := m2 [2×i+1] end mini-
  num;
for i := n step 1 until 2 × n - 1 do begin m1[i] := UNSORTED
  [i-n+1]; m2[i] := i end
for i := n - 1 step -1 until 1 do minimum;
for j := 1 step 1 until k do
  begin SORTED [j] := m1 [1]; i := m2 [1]; m1[i] := infinity;
  for i := i ÷ 2 while i > 0 do minimum end
end TREESORT 1

```

ALGORITHM 144

TREESORT 2

ARTHUR F. KAUPÉ, JR.

Westinghouse Electric Corp., Pittsburgh, Penn.

```

procedure TREESORT 2 (UNSORTED, n, SORTED, k, ordered);
  value n, k;
integer n, k; array UNSORTED, SORTED; Boolean proce-
dure ordered;
comment TREESORT 2 is a generalized version of TREESORT
  1. The Boolean procedure ordered is to have two real argu-
  ments. The array SORTED will have the property that ordered
  (SORTED[i], SORTED[j]) is true when j > i if ordered is a
  linear order relation;
begin integer i, j; array m1 [1:2×n-1]; integer array m2
  [1:2×n-1];
procedure minimum; if ordered (m1 [2×i], m1 [2×i+1]) then
  begin m1[i] := m1 [2×i]; m2[i] := m2 [2×i] end else
  begin m1[i] := m1 [2×i+1]; m2[i] := m2 [2×i+1] end mini-
  num;
for i := n step 1 until 2 × n - 1 do begin m1[i] := UNSORTED
  [i-n+1]; m2[i] := i end
for i := n - 1 step -1 until 1 do minimum;
for j := 1 step 1 until k do
  begin SORTED[j] := m1 [1]; i := m2 [1]; m1[i] := infinity;
  for i := i ÷ 2 while i > 0 do minimum end
end TREESORT 2

```

ALGORITHM 145

ADAPTIVE NUMERICAL INTEGRATION BY
SIMPSON'S RULE

WILLIAM MARSHALL McKEEMAN*

Stanford University, Stanford, Calif.

* This work was supported in part by the Office of Naval Research under contract Non4 225(37).

```

real procedure Integral (F) limits: (a, b) tolerance: (eps);
real procedure F; real a, b, eps;

```

begin comment Integral will numerically approximate the integral of the function F between the limits a and b by the application of a modified Simpson's rule. Although eps is a measure of the relative error of the result, the actual error may be very much larger (e.g. whenever the answer is small because a positive area cancelled a negative area). The procedure attempts to minimize the number of function evaluations by using small subdivisions of the interval only where required for the given tolerance;

```

integer level;
real procedure Simpson (F, a, da, Fa, Fm, Fb, absarea, est, eps);
real procedure F; real a, da, Fa, Fm, Fb, absarea, est, eps;
begin comment Recursive Simpson's rule;
  real dx, x1, x2, est1, est2, est3, F1, F2, F3, F4, sum;
  dx := da/3.0; x1 := a + dx; x2 := x1 + dx;
  F1 := 4.0 × F(a+dx/2.0); F2 := F(x1);
  F3 := F(x2); F4 := 4.0 × F(a+2.5×dx);
  est1 := (Fa+F1+F2) × dx/6.0;
  est2 := (F2+Fm+F3) × dx/6.0;
  est3 := (F3+F4+Fb) × dx/6.0;
  absarea := absarea-abs(est) + abs(est1) + abs(est2) + abs(est3);
  sum := est1 + est2 + est3;
  level := level + 1;
  Simpson := if (abs(est-sum) ≤ eps × absarea ∧ est ≠ 1.0) ∨
  level ≥ 7 then sum
  else Simpson (F, a, dx, Fa, F1, F2, absarea, est1, eps/3.0)
  + Simpson (F, x1, dx, F2, Fm, F3, absarea, est2, eps/3.0)
  + Simpson (F, x2, dx, F3, F4, Fb, absarea, est3, eps/3.0);
  level := level - 1;
end Simpson;
level := 1;
Integral := Simpson (F, b-a, F(a), 4.0 × F((a+b)/2.0), F(b),
  1.0, 1.0, eps)
end Integral 13

```

ALGORITHM 146

MULTIPLE INTEGRATION

WILLIAM MARSHALL McKEEMAN*

Stanford University, Stanford, Calif.

* This work was supported in part by the Office of Naval Research under contract Non4 225(37).

```

real procedure MultipleIntegral (F) limits: (a, b) order: (n)
  tolerance: (eps);
real procedure F; real array a, b; real eps; integer n;
begin comment F is a function of n variables which are stored
  in an internal array x. MultipleIntegral approximates the
  multiple integral of F between the n pairs of limits stored in
  the parameter arrays a and b. For a mesh of k steps on each
  axis, the number of function evaluations required for an
  integral of nth order is approximately k↑n. One consequence
  is that the practical limit on n is quite small. Another is that
  any inefficiency in the (undefined) procedure Integral will
  reflect itself to the nth power in MultipleIntegral. The adap-
  tive procedure Integral is recommended;
real array x[1:n+1]; integer axis;
real procedure Integral (F) limits: (a, b) tolerance: (eps);
real procedure F; real a, b, eps;
begin comment The body of procedure Integral is left
  undefined. For it one may substitute any procedure of the
  same name that evaluates the integral of a function of a single
  variable between the real limits a and b;
end Integral;
real procedure MI(y); real y;

```

```

begin comment Recursive multiple integration;
   $x[axis] := y$ ;
   $axis := axis - 1$ ;
   $MI := \text{if } axis = 0 \text{ then } F(x) \text{ else}$ 
     $Integral (MI, a[axis], b[axis], eps/n)$ ;
   $axis := axis + 1$ ;
end MI;
 $axis := n + 1$ ;
 $MultipleIntegral := MI(0)$ 
end MultipleIntegral

```

ALGORITHM 147

PSIF

D. AMIT

Ministry of Defense, Israel

```

real procedure  $psif(x, a, tan, ln)$  exit: ( $errexit$ );
value  $x, a$ ; label  $errexit$ ; real procedure  $tan, ln$ ;
comment Computes the logarithmic derivative of the factorial
  function defined by:

```

$$\Psi(x) = \frac{(x)'}{x!} = \frac{\Gamma'(x+1)}{\Gamma(x+1)}.$$

We make use of the expansion: (1) $\Psi(x) = \ln x + 1/2x - 1/12x^2 + 1/120x^4 - 1/252x^6 + \epsilon$, (2) $\epsilon < 1/240x^2$ and of the recursion relation, (3) $\Psi(x) = \Psi(x+n) - (1/(x+1) + \dots + 1/(x+n))$. For $x < -1$ we use: (4) $\Psi(-x) = \pi \tan \pi(x+0.5) + \Psi(x-1)$. The value of x is increased up to a . Then Ψ is calculated by (3) and (1). The error is then less than $1/240a^8$;

```

begin real  $psi, pei$ ;  $psi := 0$ ;
if  $x > -1 \wedge x \neq 0$  then go to  $pos$ ;
if  $x = 0$  then begin  $psi := -0.5772156649$ ; go to  $exit$  end;
  begin integer  $x1$ ;  $x1 := x$ ;
  if  $x = x1$  then go to  $errexit$  end
  comment  $psi$  is infinite;
   $pei := 3.141592654$ ;  $x := -x - 1$ ;
   $psi := pei \times \tan(peix(x+0.5))$ ;
 $pos$ : if  $x \geq a$  then go to  $large$ ;
   $x := x + 1$ ;  $psi := psi - 1/x$ ; go to  $pos$ ;
 $large$ : begin real  $y$ ;  $y := 1/x$ ;
   $psi := psi + \ln(x) + y/2 - y \uparrow 2/12 + y \uparrow 4/120 - y \uparrow 6/252$ ;
 $exit$ :  $psif := psi$ ;
end psif

```

ALGORITHM 148

TERM OF MAGIC SQUARE

D. M. COLLISON

Elliott Brothers (London) Ltd., Borehamwood, Herts.

```

integer procedure  $magicterm(x, y, n)$ ; value  $x, y, n$ ; integer
   $x, y, n$ ;
comment for the magic square  $s[1:n, 1:n]$ ,  $magicterm$  generates
  the element  $s[x, y]$ , where  $n > 2$  and  $n$  is odd. De la Loubère's
  method is used;
begin integer  $b, c$ ;
   $b := y - x + (n-1) \div 2$ ;  $c := y + y - x$ ;
  if  $b \geq n$  then  $b := b - n$  else if  $b < 0$  then  $b := b + n$ ;
  if  $c > n$  then  $c := c - n$  else if  $c \leq 0$  then  $c := c + n$ ;
   $magicterm := b \times n + c$ 
end magicterm

```

ALGORITHM 149

COMPLETE ELLIPTIC INTEGRAL

J. N. MERNER

Burroughs Corp., Pasadena, Calif.

comment The following two procedures, along with a test program were compiled and run by Peter Naur on the DISADEC computer. Compilation time for the 9 pass compiler was less than 10 seconds. The elliptic integral of the form

$$\int_0^{\pi/2} \frac{dt}{\sqrt{a^2 \cos^2 t + b^2 \sin^2 t}}$$

is evaluated by replacing a and b by their arithmetic and geometric means, respectively. *ELIP 2* is a nonrecursive procedure to accomplish the same thing;

real procedure *ELIP 1* (a, b); **value** a, b ; **real** a, b ;

ELIP 1 := **if** $abs(a-b) <_{10} -8 \times a$

then $3.14159265/2/a$

else *ELIP 1* ($(a+b)/2, \text{sqrt}(a \times b)$);

real procedure *ELIP 2* (a, b); **value** a, b ; **real** a, b ;

begin real C ;

L : $C := (a+b)/2$; $b := \text{sqrt}(a \times b)$; $a := c$;

if $abs(a-b) <_{10} -8 \times a$ **then** *ELIP 2* := $3.14159265/2/a$

else go to L **end**

CERTIFICATION OF ALGORITHM 31

GAMMA FUNCTION [R. M. COLLINGE, *Comm.*

ACM, Feb. 61]

PETER G. BEHREZ

Mathematikmaskinnämnden, Stockholm, Sweden

GAMMA was successfully run on FACIT EDB using FACIT-ALGOL 1, which is a realization of ALGOL 60 for FACIT EDB. No changes in the program were necessary. The relative error was as stated in the comment of GAMMA about 10^{-8} .

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

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

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

REMARK ON ALGORITHM 58
MATRIX INVERSION [D. Cohen, *Comm. ACM*,
May 61]

PETER G. BEHREZ

Matematikmaskinnmänden, Box 6131, Stockholm 6,
Sweden

invert was run on FACIT EDB using FACIT-ALGOL 1. Some changes in the procedure had to be made:

1. y and w had to be declared in the procedure-body as **real** y, w ;

2. The last part of the procedure starting with $l := 0$; which should interchange the matrix rows did not work correctly, even with the corrections proposed by R. A. Conger [*Comm. ACM*, June 62]. We propose the following code:

```
for l := 1 step 1 until n do begin
  k := z[l]; for j := l while k ≠ j do begin
    for i := 1 step 1 until n do begin
      w := a[j, i]; a[j, i] := a[k, i]; a[k, i] := w end;
      i := z[k]; z[k] := z[j]; k := z[j] := i end end end invert
```

If the matrix a is singular, the value of the pivot element y will once be zero or very nearly zero and division by zero would occur in the course of the calculation. It would therefore be advantageous to introduce an empirical tolerance parameter ϵ into the procedure.

To calculate the determinant of the matrix a it is only necessary to put three more statements into the code. With these augmentations *invert* should read:

```
procedure invert (n, a, epsilon, determinant);
value n, epsilon; real epsilon, determinant;
array a; integer n;
begin real y, w; integer i, j, k, l, p;
array b, c[1:n]; integer array z[1:n];
determinant := 1;
```

followed by the same code as before until:

```
y := w end end;
determinant := y × determinant;
if k ≠ i then determinant := -determinant;
if abs (y) < epsilon then go to singular;
```

followed by the same code as before with the changes mentioned in the certification by R. A. Conger [*Comm. ACM*, June 62] and the changes given above. *singular* should be a nonlocal label in the main program.

CERTIFICATION OF ALGORITHM 94
COMBINATION [J. Kurtzburg, *Comm. ACM*, June,
1962]

R. E. GRENCH*

Reactor Eng. Div., Argonne National Laboratory,
Argonne, Ill.

* Work supported by U. S. Atomic Energy Commission

Four changes were required in the algorithm.

1. The last sentence in the comment should read: That initial combination is also produced after 0, 1, \dots , $K-1$, the last value in that cycle;
2. The integer A was declared;
3. Parentheses were replaced by brackets in the subscript expressions;
4. A semicolon was inserted at the end of the initiate statement.

After the above changes were made the body of Algorithm 94 was tested on an LGP-30 computer using the Dartmouth College ALGOL-30 translator. The body tested satisfactorily and the time required to generate one J when $K = 5$ and $N = 15$ was 30 seconds.

Various tests should be included if this algorithm is to be used as a procedure. These tests might include a statement to check if $K > N$ and if the initial value of J is correct. These two possibilities were investigated and it was found that improper J 's are generated.

CERTIFICATION OF ALGORITHM 112
POSITION OF POINT RELATIVE TO POLYGON
[M. Shimrat, *Comm. ACM*, Aug. 1962]

RICHARD HACKER

The Boeing Co., Seattle Wash.

The Boolean procedure *POINT IN POLYGON* was programmed in FORTRAN for the IBM 7090. The algorithm gave satisfactory results except for a case such as the following:

Let the polygon points be: (0, 0), (1, 0), (2, 1), (1, 2), (0, 2).

In this case the procedure would not detect that the point (1, 1) is in the polygon. However, the correct result was obtained by changing:

if ($y < y[i] = y > y[i+1]$) \wedge

to read:

if ($y0 \leq y[i] = y0 > y[i+1]$) \wedge

CERTIFICATION OF ALGORITHM 115
PERM [H. F. Trotter, *Comm. ACM*, Aug. 1962]
E. S. PHILLIPS
Michigan State University, East Lansing, Mich.

PERM was translated into FORTRAN for the CDC 160-A, and it performed correctly. For $n = 8$, this method requires 2822 seconds. For comparison, Algorithm 86, *PERMUTE*, was translated and run on the same machine, requiring 3710 seconds as opposed to 1316 when run on an IBM 1620.

CERTIFICATION OF ALGORITHM 118
MAGIC SQUARE (ODD ORDER) [D. M. Collison,
Comm. ACM, Aug. 1962]

HENRY C. THACHER, JR.*

Reactor Engineering Div., Argonne National Lab.,
Argonne, Ill.

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

The body of the procedure *magicodd* was tested on the LGP-30 using the Dartmouth ALGOL 60 translator. No syntactical errors were found. The procedure generated odd-order magic squares satisfactorily. For orders up to 9, times were as follows (including output on the Flexowriter):

Order	Time(sec)
3	171
5	422
7	804
9	1285

The 3×3 square was:

4	3	8
9	5	1
2	7	6

REMARK ON ALGORITHM 133
RANDOM (P. G. Behrenz, *Comm. ACM*, Nov. 1962)
PETER G. BEHREZ

Matematikmaskinnmänden, Box 6131, Stockholm 6,
Sweden

Replace the declarations in the body of the procedure,
integer M35, M36, M37; own integer X;

by:

own integer X, M35, M36, M37;

The sequence of 2^{33} random numbers contains about 15 numbers which are not really random numbers. For details, see R. W. Hamming, *Numerical Methods for Scientists and Engineers*, p. 384 [McGraw-Hill, 1962].