Algorithms Supplement

Previously published algorithms

The following algorithms have recently appeared in the Algorithms Sections of the specified journals.

(*a*) **Communications of the ACM** (January–March 1969)

344 STUDENT'S *t*-DISTRIBUTION

Evaluates in single-precision the value of Student's t-distribution for argument T and degrees of freedom DF.

345 AN ALGOL CONVOLUTION PROCEDURE BASED ON THE FAST FOURIER TRANSFORM

Computes the convolution of two real vectors A and B

$$C_k = \frac{1}{n} \sum_{j=0}^{n-1} \alpha_j \beta_j^* \exp(i2\pi jk/n)$$

where α_j *and* β_j *are the Fourier transforms*

$$\alpha_j = \sum_{p=0}^{n-1} A_p \exp(i2\pi pj/n)$$

and

$$\beta_j = \sum_{q=0}^{n-1} B_q \exp(i2\pi qj/n)$$

and β_j^* *is the complex conjugate of* β_j.

346 F-TEST PROBABILITIES

Gives the probability that F will be greater than the value of f where

$$f = \sigma_1^2/\sigma_2^2,$$

σ_1^2 *is the variance of the sample with size* N_1, σ_2^2 *is the variance of the sample with size* N_2, $df1 = N_1 - 1$, $df2 = N_2 - 1$, *and F is the Snedecor-Fisher statistic.*

347 AN EFFICIENT ALGORITHM FOR SORTING WITH MINIMAL STORAGE

Sorts the elements of an array into ascending order.

(*b*) **Applied Statistics** (June 1969)

AS13 MINIMUM SPANNING TREE

Computes the minimum spanning tree of a distance matrix stored in lower triangular form without diagonal elements, using Prim's method. The procedure can be modified to handle matrices of similarities, and matrices stored in other ways.

AS14 PRINTING THE MINIMUM SPANNING TREE

Operating on the output of AS13, *this procedure prints the links of the minimum spanning tree in an order helpful in preparing it for display.*

AS15 SINGLE LINKAGE CLUSTER ANALYSIS

Uses the minimum spanning tree to compute a single linkage cluster analysis. Two forms of output are provided, (i) *a list of the members of each group at each level of clustering and* (ii) *a graphical form of* (i) *known as a dendrogram.*

AS16 MAXIMUM LIKELIHOOD ESTIMATION FROM GROUPED AND CENSORED NORMAL DATA

Computes the maximum likelihood estimates of the mean and standard deviation from a censored or grouped normal sample. Approximate values of their variances and covariance are also given.

AS17 THE RECIPROCAL OF MILLS' RATIO

For a given x the procedure calculates the corresponding ratio of the ordinate to the upper tail area for the standardised normal distribution, $Z(x)/Q(x)$. *The value of this function is required as a parameter in* AS16.

The following papers, containing useful algorithms, have recently appeared in the specified journals.

(*a*) **Numerische Mathematik** (December 1968)

SIMILARITY REDUCTION OF A GENERAL MATRIX TO HESSENBERG FORM (Band 12, Heft 5, pp. 349–368)

MODIFIED *LR* ALGORITHM FOR COMPLEX HESSENBERG MATRICES (Band 12, Heft 5, pp. 369–376)

IMPLICIT *QL* ALGORITHM (Band 12, Heft 5, pp. 377–383)

(*b*) **International Journal for Numerical Methods in Engineering** (January–March 1969)

A CONFORMING QUARTIC TRIANGULAR ELEMENT FOR PLATE BENDING (Vol. 1, No. 1, pp. 29–45)

New algorithms

Algorithm 41

A CURVE PLOTTING PROCEDURE

P. J. Le Riche
The Computation Laboratory
University of Southampton

Author's Note:

In order to plot the graph of a function it is often convenient to tailor a program for the purpose. Nevertheless, it is useful to have a procedure which will plot any function given to it, particularly if functions are likely to be encountered for which dynamic step length adjustment is essential. The procedure *CURVE*, given here, is an attempt at such a general procedure and will be found to handle practically all functions encountered, including any having discontinuities of themselves or their derivatives.

The recursive definition of the procedure makes it essentially simple and elegant and gives it the advantage that a minimum number of points is calculated, all of which are used.

procedure *curve* (*f, p, a, b, inc, phi, eps*); **value** *a, b, inc, phi, eps*; **real** *a, b, inc, phi, eps*; **procedure** *f*; **real array** *p*;
comment *draws the function f in the range* $a \leqslant t \leqslant b$ *where t is a parameter from which the x and y coordinates are found by f. p is an array of constants which may be used in the definition*

```
of f. The maximum increment of t is inc and this is subdivided
recursively until chords in adjacent subdivisions differ in
direction by less than phi radians or the chords are shorter than
eps. The maximum chord length satisfying these conditions is
found at all points and drawn. Finite discontinuities of the
function and its derivatives will be plotted correctly;
  begin real x1, x2, y1, y2, h, k, l; Boolean b1, b2, bb;
  integer m, n;

  procedure d(x1, y1, x2, y2, a, b);
  value x1, y1, x2, y2, a, b; real x1, y1, x2, y2, a, b;
  comment draws two chords in the interval supplied if they
  satisfy the conditions, otherwise calls itself recursively twice,
  in each case supplying half the interval;
    begin real x, y, c, l1, l2, theta;
    c := (a + b)/2; f(x, y, c, p);
    l1 := sqrt((x1 - x) ↑ 2 + (y1 - y) ↑ 2);
    l2 := sqrt((x - x2) ↑ 2 + (y - y2) ↑ 2);
    theta := (((x1 - x) × (x - x2) + (y1 - y) × (y - y2))/
    (l1 × l2));
    if theta ⩾ 1·0 then theta := 0 else theta := sqrt (1·0 -
    theta ↑ 2);
    b1 := b2 := false;
    if abs(l - l1) ⩽ eps ∧ l2 ⩽ eps then b1 := true;
    if abs(l - l2) ⩽ eps ∧ l1 ⩽ eps then b2 := true;
    if b1 ∧ b2 then b1 := b2 := false;
    comment b1 is true if there is a discontinuity in the first
    half of the interval and similarly b2;
    bb := theta ⩽ phi ∨ l1 ⩽ eps ∨ l2 ⩽ eps;
    bb := bb ∧ m ⩾ n - 1;
    if bb then
      begin
      if b1 then penup; draw(x, y);
      if b2 then penup; draw(x2, y2);
      n := m
      end
    else
      begin
      m := m + 1; l := l1;
      d(x1, y1, x, y, a, c); l := l2;
      d(x, y, x2, y2, c, b); m := m - 1
      end
    end of d;

  penup;
  n := if (b - a) / inc < 1 then 1 else entier(((b - a) / inc
  + 1) / 2);
  inc := (b - a) / n; m := n := 0;
  b := b + inc / 2; f(x2, y2, a, p);
  draw(x2, y2);
  for k := a + inc step inc until b do
    begin
    h := k - inc; x1 := x2;
    y1 := y2; f(x2, y2, k, p);
    l := sqrt((x2 - x1) ↑ 2 + (y2 - y1) ↑ 2);
    d(x1, y1, x2, y2, h, k)
    end
  end of curve;

procedure penup;
comment This procedure and the following one are machine
dependent user supplied procedures. penup causes the plotter
pen to be raised before it is next moved and automatically
lowered again after the move;
procedure draw(x, y); value x, y; real x, y;
comment The pen is moved to a position with coordinates x, y
with the pen down unless penup has been called since the last
call of draw;

procedure f(x, y, t, p); value t;
real x, y, t; real array p;
comment This procedure contains the function to be plotted and
must be written by the user. t specifies a point on the curve
and the procedure body must calculate values of x and y from
it, assigning them to the parameters. p may be used to hold
any constants in the function which need to be assigned by
program;
```

Algorithm 42

INTERPOLATION BY CERTAIN QUINTIC SPLINES

H. Späth
Institut für Neutronenphysik und Reaktortechnik
Kernforschungszentrum Karlsruhe
Germany

Author's Note:

We consider the problem of finding a function f such that for given data triples (x_k, y_k, y_k') $(k = 1, \ldots, n \geqslant 2)$ with $x_1 < x_2 < \ldots < x_n$ and values y_1'' and y_n'' the following relations hold:

$$(1) \qquad f(x_k) = y_k \quad (k = 1, \ldots, n)$$

$$(2) \qquad f'(x_k) = y_k' \quad (k = 1, \ldots, n)$$
$$f''(x_k) = y_k'' \quad (k = 1, n)$$
$$f \in C^3[x_1, x_n]$$

Such a function f is given by piecewise quintic polynomials

$$f(x) = f_k(x) \text{ for } x \in [x_k, x_{k+1}](k = 1, \ldots, n-1)$$

with $$f_k(x) = A_k z^5 + B_k z^4 + C_k z^3 + D_k z^2 + E_k z + F_k$$

where $$z = x - x_k$$

with $6(n-1)$ suitably determined constants.

From the conditions (1) and (2) we have at once

$$(3) \qquad E_k = y_k', F_k = y_k \ (k = 1, \ldots, n-1)$$

We can express the coefficients A_k, B_k, C_k in terms of the x_k, y_k, y_k' and D_k to give:

$$(4) \quad \begin{cases} A_k = \dfrac{1}{(\Delta x_k)^5}[6\Delta y_k - 3\Delta x_k(y_k' + y_{k+1}') \\ \qquad\qquad + (\Delta x_k)^2(D_{k+1} - D_k)] \\ B_k = \dfrac{1}{(\Delta x_k)^4}[-15\Delta y_k + \Delta x_k(8y_k' + 7y_{k+1}') \\ \qquad\qquad + (\Delta x_k)^2(3D_k - 2D_{k+1})] \\ C_k = \dfrac{1}{(\Delta x_k)^3}[10\Delta y_k - \Delta x_k(6y_k' + 4y_{k+1}') \\ \qquad\qquad + (\Delta x_k)^2(D_{k+1} - 3D_k)] \end{cases}$$

Matching the third derivative of f gives the following tridiagonal system of linear equations for the $D_k(k = 2, \ldots, n-1)$ where D_1 and D_n are given:

$$-\frac{1}{\Delta x_{k-1}} D_{k-1} + 3\left(\frac{1}{\Delta x_{k-1}} + \frac{1}{\Delta x_k}\right) D_k - \frac{1}{\Delta x_k} D_{k+1}$$
$$= 10\left[\frac{\Delta y_k}{(\Delta x_k)^3} - \frac{\Delta y_{k-1}}{(\Delta x_{k-1})^3}\right] + \left[\frac{4}{(\Delta x_{k-1})^2} y_{k-1}'\right.$$
$$\left. + 6\left(\frac{1}{(\Delta x_{k-1})^2} - \frac{1}{(\Delta x_k)^2}\right) y_k' - \frac{4}{(\Delta x_k)^2} y_{k+1}'\right]$$

As the matrix of coefficients is symmetric and strictly diagonally dominant it is also positive definite. Therefore, a unique solution exists and Gaussian elimination is numerically stable without pivoting. Having determined the D_k we obtain the remaining coefficients by (4).

The method can be extended to interpolate data points $(x_k, y_k, y_k', y_k'', \ldots, y_k^{(m-2)})(k = 1, \ldots, n; m > 3)$ with given values for $y_1^{(m-1)}$ and $y_n^{(m-1)}$ by piecewise polynomials of degree $2m - 1$ such that the interpolating function has m continuous derivatives in $[x_1, x_n]$. One would have to solve once for all a $m \times m$ linear system for the first m coefficients of f_k and for given data a tridiagonal system for the values $y_k^{(m-1)}(k = 2, \ldots, n - 1)$.

The algorithm is given as **Table 1.**

Table 1

```
      SUBROUTINE QUINT(N, X, Y, Y1, Y21, Y2N, A, B, C, D)
      DIMENSION X(1), Y(1), Y1(1), A(1), B(1), C(1), D(1)
C
C FOR GIVEN DATA TRIPLES (X(K), Y(K), Y1(K), K = 1, N), N
C GREATER THAN OR EQUAL TO TWO, AND VALUES Y21 AND Y2N
C FOR THE SECOND DERIVATIVE AT THE POINTS X(1) AND X(N)
C THIS SUBROUTINE CALCULATES THE COEFFICIENTS (A(K), B(K),
C C(K), D(K), E(K), F(K), K = 1, N1), (N1 = N - 1), OF
C QUINTIC POLYNOMIALS DEFINED IN THE INTERVALS (X(K), X(K + 1))
C THAT JOIN TO A THREE TIMES CONTINUOUSLY DIFFERENTIABLE
C FUNCTION F DEFINED IN (X(1), X(N)).
C THE ACTUAL ARRAY D MUST HAVE THE DIMENSION N BECAUSE IT IS
C ALSO USED AS INTERMEDIATE STORAGE WORKING FIELD.
C THE VALUES OF E(K) AND F(K), (K = 1, N1), ARE RETURNED IN THE
C ARRAYS Y1 AND Y RESPECTIVELY.
C
      N1 = N - 1
      D(1) = 0.5 * Y21
      D(N) = 0.5 * Y2N
      C(1) =0.0
      A(1) =0.0
      IF (N - 2) 13, 1, 2
1     R2 = 1.0 / (X(2) - X(1))
      B(1) = R2 * R2 * R2 * (Y(2) - Y(1))
      GOTO 11
2     DO 9 K = 1, N1
      R2 = 1.0 / (X(K + 1) - X(K))
      G2 = R2 * R2
      G = G2 * R2
      R = (Y(K + 1) - Y(K)) * G
      B(K) = R
      F = Y1(K)
      F2 = 10.0 * R - 4.0 * G2 * (F + Y1(K + 1))
      IF (K - 1) 3, 8, 3
3     R = F1 + F2 + 2.0 * (G1 - G2) * F
      G = 3.0 * (R1 + R2)
      IF (K - N1) 5, 4, 5
4     R = R + R2 * D(N)
5     IF (K - 2) 7, 6, 7
6     R = R + R1 * D(1)
7     F = 1.0 / (G - R1 * C(K - 1))
      C(K) = R2 * F
      A(K) = (R + R1 * A(K - 1)) * F
8     R1 = R2
      F1 = -F2
      G1 = G2
9     CONTINUE
      D(N1) = A(N1)
      N2 = N - 2
      DO 10 I = 2, N2
      K = N - I
10    D(K) = C(K) * D(K + 1) + A(K)
11    F1 = Y1(1)
      DO 12 K = 1, N1
      R = 1.0 / (X(K + 1) - X(K))
      R1 = R * R
      R2 = R1 * R
      F2 = Y1(K + 1)
      F = B(K)
      C(K) = 10.0 * F - R1 * (6.0 * F1 + 4.0 * F2) +
     CR * (D(K + 1) - 3.0 * D(K))
      F = F * R
      B(K) = -15.0 * F + R2 * (8.0 * F1 + 7.0 * F2) +
     CR1 * (3.0 * D(K) - 2.0 * D(K + 1))
      A(K) = 6.0 * F * R - 3.0 * R1 * R1 * (F1 + F2) +
     CR2 * (D(K + 1) - D(K))
      F1 = F2
12    CONTINUE
13    RETURN
      END
```

Note on Algorithm 35

BEST RATIONAL APPROXIMATION TO A REAL NUMBER

This problem has been investigated as a practical example by Computer Science M.Tech. students at Brunel University. Algorithm 35 appears to be inefficient for large values of *bound* except in special cases. For example, if *real* equals π, x is just under one-seventh; it would be approximately seven times faster to find a rational approximation p'/q to $\pi - 3$ by searching p' from 1 until the associated q reaches *bound*, instead of the q search suggested. Admittedly, $q := p'/x$ may not be the best q for some values of p', but it will be for good values of p', and it is with these that the process is concerned.

A better technique altogether is well known to students of continued fractions, e.g. H. S. Hall and S. R. Knight, *Higher Algebra*, Macmillan 1929 (4th edition—1st edition 1887), Chapter XXV. 3·14159, for example, can be written $3 + \frac{1}{7+}\,\frac{1}{15+}\,\frac{1}{1+}\,\frac{1}{25+}\,\frac{1}{1+}\,\frac{1}{7+}\,\frac{1}{4}$. If we write $\frac{p_i}{q_i} = n_0 + \frac{1}{n_1+}\,\frac{1}{n_2+}\cdots\frac{1}{n_i}$ as an approximation to some *real*, then it is known that each successive convergent is better than any other rational approximation with smaller q. Successive p_i and q_i are computed from the recurrence relations $p_i = n_i p_{i-1} + p_{i-2}$ and $q_i = n_i q_{i-1} + q_{i-2}$, and the successive n_i are obtained from the reciprocal of the previous residues, e.g. 7 is the integer part of $\frac{1}{0{\cdot}14159}$, 15 is the integer part of the reciprocal of $\frac{1}{0{\cdot}14159} - 7$, and so on.

This sequence may miss a value near the terminal *bound*, e.g. $3 + \frac{1}{7+}\,\frac{1}{14}$ is a better approximation than $3 + \frac{1}{7}$, but this can be accommodated when a q_i is found which exceeds *bound* by testing whether the largest possible n_i' such that $n_i' q_{i-1} + q_{i-2} \leqslant$ *bound* gives a better approximation than p_{i-1}/q_{i-1}.

The process can be made a little faster by using nearest integers rather than integer parts in the division, e.g. the sequence for 3·14159 can be written $3 + \frac{1}{7+}\,\frac{1}{16-}\,\frac{1}{27-}\,\frac{1}{8+}\,\frac{1}{4}$. This is conveniently programmed by allowing negative n_i, accepting negative p, q combinations.

M. L. V. Pitteway
Brunel University

R. W. Parry
Reading College of Technology

Note on Algorithm 35

Algorithm 35, though logically correct, is a very inefficient means of finding the best rational approximation p/q to a real number, since it involves increasing q in unit steps and checking at each stage.

A much faster method is based on the continued fraction

$$real = b_o + \frac{1}{b_1+}\,\frac{1}{b_2+}\,\frac{1}{b_3+}\cdots\frac{1}{b_r+}\cdots$$

Terminating the continued fraction at the rth stage yields the approximation

$$real \simeq p_r/q_r$$

where

$$p_r = b_r p_{r-1} + p_{r-2},\ q_r = b_r q_{r-1} + q_{r-2},\ r \geqslant 2.$$

There are alternative ways of finding the b_r.

When the object is to find the maximum accuracy for a given bound on q the fastest method is as follows:

(i) Put b_o equal to the nearest integer to *real*,

(ii) b_1 = nearest integer to $\frac{1}{real - b_o}$,

(iii) b_2 = nearest integer to $\frac{1}{\frac{1}{real - b_o} - b_1}$,

and so on. In it the b_r can be positive or negative.

Such an expansion has been used for many years in finding rational approximants to transcendental functions for computing purposes. For example, one expansion for *tan x* is

$$\tan x = \frac{x}{1+}\ \frac{-x^2/3}{1+}\ \frac{-x^2/15}{1+}\ \frac{-x^2/35}{1+}\ \cdots$$

This is the last method suggested by Pitteway and Parry in the comments published above. An algorithm based on it is given as Appendix 1.

However, in many practical problems the important criterion is a fixed accuracy using the smallest possible q. In these circumstances it is necessary to increase q_r in the smallest sensible steps, i.e. in steps of q_{r-1}, equivalent to increasing b_r in unit steps. Testing for convergence is then carried out at each stage. It is also desirable to keep the $b_r > 0$, which requires the error to be tested for a change in sign. The same method can be used when the object is to find, as in Algorithm 35, the maximum accuracy for a given bound on q, but it is slower than continued reciprocation. An algorithm to do this is given as Appendix 2. To convert this algorithm to the fixed accuracy form it is only necessary to move the convergence test to follow the calculation of x. The real variable accuracy is, of course, read in rather than calculated.

It is perhaps worth mentioning that in the equipment design problem for which the original version of this algorithm was written the accuracy was a function of q, so that in a very few cases convergence was not possible even by stepping b_r, so that other methods had to be used.

To test the various methods a set of 401 cases was run on the Atlas computer with bound = 2500. The execution time measured in instruction interrupts was 83071 for Algorithm 35, but only 967 using Appendix 1, making a speed-up factor of 86. An even larger factor could be expected with a larger bound. Using Appendix 2 the time taken was 2281, whilst with positive b_r formed by reciprocation it was 1093. Thus reciprocation is over twice as fast as increasing b_r in unit steps.

References

PRESCOTT, R. J. (1968). Algorithm 35. Best Rational Approximation to a Real Number, *The Computer Journal*, Vol. 11, No. 3, pp. 347–350.

PITTEWAY, M. L. V., and PARRY, R. W. (1969). Note on Algorithm 35, *The Computer Journal*, Vol. 12, No. 3, pp. 293

APPENDIX 1

```
procedure continued fraction(real, p, q, bound); value real,
bound; real real; integer p, q, bound;
  begin real x, residue; integer b, p1, p2, q1, q2;
  comment This procedure finds the best rational approximation
  p / q to a real number real, in the sense that abs(p / q − real)
  is a minimum, subject to the condition 0 < q ⩽ bound.
  The approximation is by a continued fraction expansion
  for which the recurrence relations are p = b × p2 + p1,
  q = b × q2 + q1, where p2 / q2 and p1 / q1 are the best
  approximations obtained with the two next lower order
  truncations;
  x := abs(real − entier(real + 0·5));
  comment x is the distance of real from its nearest integer;
  if x < 1·0 / bound then
    begin
    if x < 0·5 / bound then
      begin
      comment solution is an integer;
      p := real; q := 1;
      goto DONE
      end
    else
      begin
      comment solution is of form integer / bound;
      q := bound; p := real × bound;
      goto DONE
      end
    end;
  p1 := real; q1 := 1;
  residue := real − p1; x := 1 / residue;
  q2 := x; p2 := p1 × q2 + q1;
  residue := x − q2;
  comment These are the initial values of the iteration;
NEXT: x := 1 / residue; b := x;
  residue := x − b; p := b × p2 + p1;
  q := b × q2 + q1;
  if abs(q) > bound then goto REDUCEb else
    begin
    p1 := p2; q1 := q2;
    p2 := p; q2 := q;
    goto NEXT
    end;
REDUCEb: b := (sign(q) × bound − q1)÷q2;
  p := b × p2 + p1; q := b × q2 + q1;
  if abs(p/q − real) < abs(p2/q2 − real) then goto DONE
  else p := p2; q := q2;
DONE: p := abs(p); q := abs(q)
  end of continued fraction
```

APPENDIX 2

```
procedure continued fraction(real, p, q, bound); value real
bound; real real; integer p, q, bound;
  begin real x, r, accuracy; integer p2, q2;
  comment This procedure finds the best rational approximation
  p / q to a real number real, in the sense that abs(p / q − real)
  is a minimum, subject to the condition 0 < q ⩽ bound.
  The approximation is by a continued fraction expansion
  for which the recurrence relations are p = b × p2 + p1,
  q = b × q2 + q1, where p2 / q2 and p1 / q1 are the best
  approximations obtained with the two next lower order
  truncations;
  x := abs(real − entier(real + 0·5));
  comment x is the distance of real from its nearest integer;
  if x ⩽ 1·0 / bound then
    begin
    if x ⩽ 0·5 / bound then
      begin
      comment solution is an integer;
      p := real; q := 1;
      goto DONE
      end
    else
      begin
      comment solution is of form integer / bound;
      q := bound; p := real × bound;
      goto DONE
      end
    end;
  accuracy := 0·5 / (bound × (bound − 1));
```

```
  comment If abs(p / q − real) < accuracy, p and q will
  necessarily give the best approximation, since for any pair
  of rational numbers p1 / q1 and p2 / q2, not equal in value,
  satisfying 0 < q1 ⩽ q2 ⩽ bound,
  abs(p1 / q1 − p2 / q2) ⩾ 1 / (bound × (bound − 1));
  p := entier(real); q2 := entier(1 / (real − p));
  p2 := p × q2 + 1; p := p + p2;
  q := 1 + q2;
  comment These are the initial values of the iteration;
  r := −1;
  comment Sign of r is that of p / q − real;
NEXT: if q > bound − q2 then
    begin
    if abs(x) < abs(p2 / q2 − real) then goto DONE else
      begin
      p := p2; q := q2;
      goto DONE
      end
    end;
  p := p + p2; q := q + q2;
  x := p / q − real;
  if r × x > 0 then goto NEXT;
  p2 := p − p2; q2 := q − q2;
  if abs(p2 / q2 − real) < accuracy then
    begin
    p := p2; q := q2;
    goto DONE
    end
  else
    begin
    r := —r; goto NEXT
    end
DONE: end of continued fraction
```

S. M. Cobb
The Plessey Company Ltd.
Roke Manor
Romsey

Contributions for the Algorithms Supplement should be sent to

Mrs. M. O. Mutch
University Mathematical Laboratory
Corn Exchange Street
Cambridge

Future papers

The following papers have been accepted for publication but, owing to pressure of space, have had to be held over to the next issue:

R. P. Tewarson. A least squares iterative method for singular equations.

An iterative method for the solution of a system of simultaneous linear equations, having a singular coefficient matrix A, is described. The method is obtained by minimising (in the least squares sense) the image under A^T of a given residual vector. (Received December 1968)

D. Wood. The theory of left factored languages: Part 1

Left factored grammars and languages are introduced and their relevance to syntax-directed top-down analysers is discussed. A number of results concerning these languages are proved, including the decidability of left factored grammars. Finally a number of open problems are posed. (Received February 1969)

W. A. Zaremba. A syntax for ALGOL input/output formats.

The desirability of simple and yet comprehensive input/output formats is postulated. A possible grammar for ALGOL formats is given and shown to be of a simple precedence type. A reduced precedence matrix has been included and the meaning of major constructs explained and illustrated using the regular expression language. (Received January 1969)