Algorithms Supplement

Previously published algorithms

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

(*a*) **Communications of the ACM** (April 1970)

378 DISCRETIZED NEWTON-LIKE METHOD FOR SOLVING A SYSTEM OF SIMULTANEOUS NON-LINEAR EQUATIONS

Solves a system of simultaneous nonlinear algebraic or transcendental equations.

379 SQUANK (SIMPSON QUADRATURE USED ADAPTIVELY—NOISE KILLED)

Produces an expression of the form

$$\sum_{i=1}^{N} w_i f(x_i)$$

which is an approximation to the integral

$$\int_a^b f(x)\,dx.$$

(*b*) **BIT** (January 1970)

25 ANALYSIS OF THE OUTER PRODUCT OF SYMMETRIC GROUP REPRESENTATIONS

Uses the method of Littlewood and Richardson to analyse the outer product of two irreducible representations of the symmetric group.

(*c*) **Applied Statistics** (July–October 1970)

AS29 THE RUNS UP AND DOWN TEST

The procedure computes a χ^2 statistic based on the observed and expected number of runs (sequences of increasing and decreasing numbers) of different lengths in a set of n real numbers.

AS30 HALF NORMAL PLOTTING

Produces half normal plots on a line printer.

AS31 OPERATING CHARACTERISTICS AND AVERAGING SAMPLE SIZE FOR BINOMIAL SEQUENTIAL SAMPLING

Two procedures both giving the probability of rejection and average sample size for given values of handicap, penalty, target and population proportion of failures.

AS32 THE INCOMPLETE GAMMA RATIO

Evaluates $I(x, p) = \frac{1}{\Gamma(p)} \int_0^x e^{-t} t^{p-1} dt$ *for* $x \geqslant 0, p > 0$.

AS33 CALCULATION OF HYPERGEOMETRIC SAMPLE SIZES

Evaluates n given P, N and m where $P = \frac{(N-m)!\,(N-n)!}{N!\,(N-m-n)!}$

AS34 SEQUENTIAL INVERSION OF BAND MATRICES

The ith super and sub diagonal of a symmetric band matrix Σ_n *of order n have all elements equal to* σ_i *for* $i \leqslant k$, *zero otherwise. The program evaluates* Σ_{n+1}^{-1} *given* Σ_n^{-1}.

(*d*) **Computing** (January 1970)

12 A DISCRETE METHOD FOR THE SOLUTION OF FINITE-DIMENSIONAL SYSTEMS OF NON-LINEAR EQUATIONS

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

(*a*) **BIT** (January 1970)

ON NIELSEN'S GENERALIZED POLYLOGARITHMS AND THEIR NUMERICAL CALCULATION (Bind 10, Hefte Nr. 1, pp. 38–73)

A TRANSITIVE CLOSURE ALGORITHM (Bind 10, Hefte Nr. 1, pp. 76–94)

(*b*) **Computing** (February 1970)

DIRECT METHODS FOR EVALUATION OF ZEROES OF POLYNOMIALS (Vol. 5, Fasc. 2, pp. 97–118)

New algorithms

Algorithm 53

DECOMPOSITION OF POSITIVE DEFINITE SYMMETRIC BAND MATRICES

R. A. Zambardino
North Staffordshire Polytechnic
Stafford

Author's note:

These two procedures extend to positive definite symmetric band matrices the form of storage as a uni-dimensional array which is usually applied to symmetric matrices not of a band form.

If N is the order of the matrix and $2W + 1$ its band width, the array must be allocated $(W + 1)\ (2N - W) \div 2$ stores and should contain only the band elements of the upper triangle of the original matrix, in row sequence; i.e.:

$a(1, 1), a(1, 2), \ldots, a(1, W + 1), a(2, 2), \ldots, a(2, W + 2), a(3, 3), \ldots, a(N - 1, N), a(N, N)$.

Since this is, for any value of W, the minimum number of elements needed for the decomposition, these procedures can be used, without loss of efficiency, whatever the value of W, up to and including the value $W = N - 1$, i.e. when the matrix is not of a band form. The same procedures and therefore the same storage method can then be used for any positive definite symmetric matrix, whether of a band form or otherwise.

This form of storage is less restrictive than the alternative

form, as a two dimensional array of dimensions N and $W + 1$, used by Martin and Wilkinson (1965), which, as stated by the authors, becomes less efficient as W approaches N.

In the present version the two procedures over-write the original array A and the matrix of right-hand sides B; however only trivial modifications are required if A and/or B need to be preserved.

The decomposition of the original matrix is based on the Crout method. As compared to the Cholesky method, this avoids the calculation of N square roots at the cost of $W(N - (W + 1)/2)$ additional multiplications; a local array $PR[1:W]$ is used to minimise the number of additional multiplications.

Reference

MARTIN, R. S., and WILKINSON, J. H. (1965). Symmetric decomposition of positive definite band matrices, *Numerische Mathematik*, Vol. 7, pp. 355–361.

```
procedure symbandprp(n, w, a, d, FAIL); value n, w; integer
n, w; real d; array a; label FAIL;

comment The upper half of a positive definite symmetric band
matrix of order n, with a band width 2 × w + 1, is stored as
a uni-dimensional array a[1 : (w + 1) × (2 × n − w) ÷ 2].
If the matrix is not of a band type, it can still be considered
as such with w = n − 1. The Crout decomposition A = LU
is performed but, since for symmetric matrices and i ≠ j,
l(i, j) = u(j, i) × l(j, j), only the super-diagonal elements of U
and the diagonal elements of L need to be stored. These
elements are stored in array a, over-writing the elements of the
original matrix. The value of the determinant is also calculated
and stored in d. A jump to label FAIL will occur if any
diagonal element becomes ⩽0, the value left in d being, corre-
spondingly ⩽0;

  begin real s, m; array pr[1:w]; integer v, t, l, i, j, k, q, r, p;
  v := 1; d := 1·0;
  t := l := w + 1;
  for i := 1 step 1 until n do
    begin
    q := if w > i − 1 then i − 1 else w;
    if i > n − w then t := t − 1;
    r := t − 1; p := v;
    for j := 1 step 1 until q do
      begin
      if r < w then r := r + 1;
      p := p − r; pr[j] := a[p] × a[p − j]
      end j;
    for j := 1 step 1 until t do
      begin
      if j > l then q := q − 1;
      s := a[v]; r := t − 1;
      p := v;
      for k := 1 step 1 until q do
        begin
        if r < w then r := r + 1;
        p := p − r; s := s − pr[k] × a[p]
        end k;
      if j = 1 then
        begin
        m := a[v] := s; d := d × s;
        if s ⩽ 0 then goto FAIL
        end
      else a[v] := s / m;
      v := v + 1
      end j;
    if l > 1 then l := l − 1
    end i
  end symbandprp
```

```
procedure symbandsol(n, w, a, r, b); value n, w, r; integer n,
w, r; array a, b;
comment This procedure solves a system of n linear equations
when the matrix of the coefficients is a positive definite sym-
metric band matrix, with band width 2 × w + 1. When the
matrix is not of a band type, it can still be considered as such,
with w = n − 1.

Before entering this procedure the coefficient matrix must be
decomposed by using the procedure symbandprp, a being the
array of dimension 1 : (w + 1) × (2 × n − w) ÷ 2 obtained
from symbandprp. b is a n × r matrix of right-hand sides.
The procedure performs the operation LY = B and then the
back-substitution UX = Y. The solution is stored in matrix b,
over-writing the right-hand sides. If b is a diagonal or lower
triangular matrix of order n (e.g. the identity matrix), the
number of operations is reduced if the value −n is given to r
when entering the procedure;

  begin integer i, j, k, v, t, p, q; real s, m;
  v := 1; t := w;
  p := r;
  for i := 1 step 1 until n do
    begin
    if r = − n then p := i;
    if i > n − w then t := t − 1;
    m := a[v];
    for k := 1 step 1 until p do
      begin
      s := b[i, k];
      for j := 1 step 1 until t do
      b[i + j, k] := b[i + j, k] − s × a[v + j];
      b[i, k] := s / m
      end k;
    v := v + t + 1
    end i;
  v := v − 1; t := 1;
  for i := n − 1 step −1 until 1 do
    begin
    q := i + t + 1;
    for k := 1 step 1 until p do
      begin
      s := b[i, k];
      for j := 1 step 1 until t do
      s := s − a[v − j] × b[q − j, k];
      b[i, k] := s
      end k;
    v := v − t − 1;
    if t < w then t := t + 1
    end i
  end symbandsol
```

Algorithm 54

APPROXIMATION OF STRAIGHT LINES

A. H. J. Sale
Basser Computing Dept.
University of Sydney

Author's note:

This routine implements a method described by J. E. Bresenham (1965) for the control of an incremental digital plotter in USASI FORTRAN. A similar algorithm has been published earlier (Stockton, 1963) which is externally identical to that given here (it produces the same set of increments): this version is simpler and is written to achieve fast execution of the inner loop. No multiplications or divisions are needed.

Consider a Cartesian grid of lines of unit spacing, and a plotter which may plot points only at intersections of the grid lines. Then given two points (x_1, y_1) and (x_2, y_2) on the grid

the routine supplies a set of increments $(\Delta x_i, \Delta y_i)$ which when added successively to (x_1, y_1) produce an approximation to the straight line joining these two points, and such that no increment is greater than 1 in magnitude. The minimum number of increments is obviously

$$\max(\text{abs}(x_1 - x_2), \text{abs}(y_1 - y_2))$$

and in fact this number are supplied.

It can be seen from the above that of the two increments Δx_i and Δy_i, one will always be unity (in magnitude) after a call to the routine, and therefore more compact ways of returning the results than that chosen are possible. However since the purpose of the algorithm is to present a method, and since in most applications it will probably be rewritten in a suitable assembly language or recast in form, it was thought unwise to obscure the basic technique.

To illustrate the choice of points by the algorithm a 'print-plot' produced by a sample driver program is shown in Fig. 1. This consists of the numeral '2' (plotted as five straight-line segments) with various scales and in various orientations.

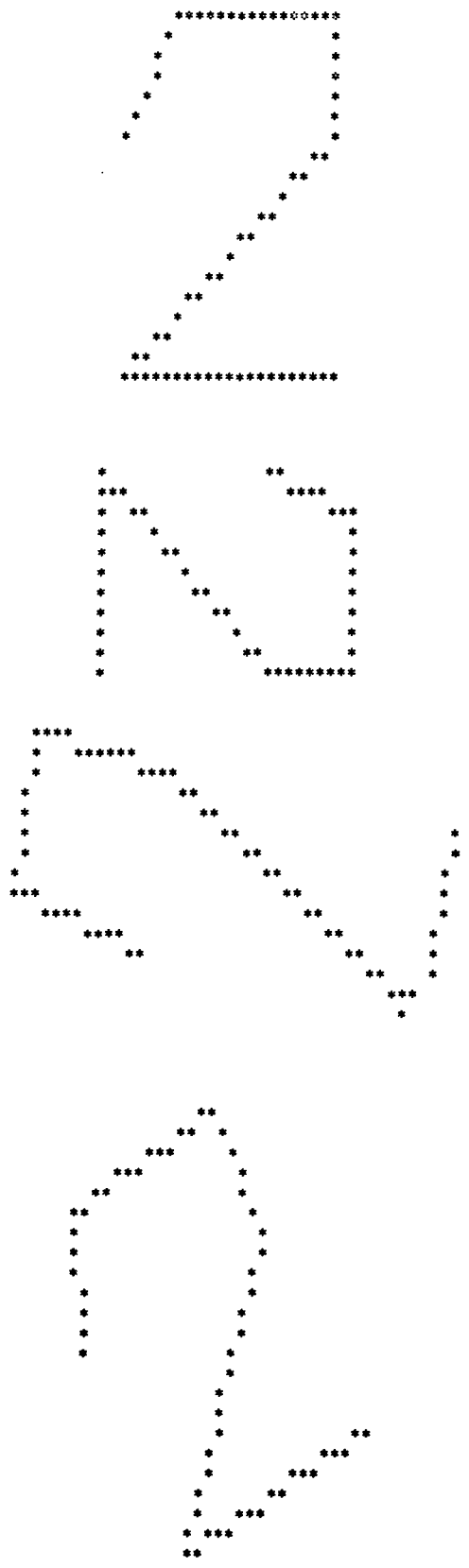


Fig. 1. Points chosen by LLINE

```
      SUBROUTINE LLINE(KX1,KY1,KX2,KY2,KXRAY,KYRAY,MAX,LENG)
C
C     LLINE PRODUCES AN APPROXIMATION TO THE STRAIGHT LINE BETWEEN THE
C     POINTS (KX1,KY1) AND (KX2,KY2), USING ONLY UNIT INCREMENTS
C     IN THE X-DIRECTION, IN THE Y-DIRECTION, OR BOTH TOGETHER.
C
C     THE (X,Y) INCREMENTS FOR EACH STEP ARE STORED IN THE ARRAYS
C     KXRAY AND KYRAY, WHICH MUST BE DECLARED IN THE CALLING
C     PROGRAM AS ONE-DIMENSIONAL AND AT LEAST OF LENGTH MAX.
C     IT IS SIMPLE TO MAKE THE ARRAYS STORE INSTEAD THE COORDINATES
C     WHICH CAN BE COMPUTED INSIDE THE DO LOOP.
C
C     THE VARIABLE LENG IS SET TO INDICATE THE LENGTH OF THE USED
C     PORTION OF THE ARRAYS. IF THIS IS EQUAL TO MAX THE ROUTINE
C     MAY HAVE RUN OUT OF ARRAY SPACE. TO PREVENT THIS, MAX MUST
C     BE AT LEAST EQUAL TO THE LARGEST (IN MAGNITUDE) OF
C     KX2-KX1 AND KY2-KY1.
C
C     SPECIFICATIONS
      INTEGER KX1,KY1,KX2,KY2,MAX,LENG
      INTEGER KXRAY(MAX),KYRAY(MAX)
C     DECLARATIONS
      INTEGER KDELA,KDELB,KDELI,KDELB2,KDELD2
      INTEGER INCX1,INCX2,INCY1,INCY2
      INTEGER J
C     EQUIV NOT NEEDED, JUST IN TO SAVE SPACE, AND FOR MNEMONICS
      EQUIVALENCE (KDELB,KDELB2)
C
C     SET UP THE INITIAL POINT COORDINATES, DELTA X AND DELTA Y,
C     AND COMPUTE THE 45 DEGREE INCREMENTS
      CALL SGNABS(KX2-KX1,INCX2,KDELA)
      CALL SGNABS(KY2-KY1,INCY2,KDELB)
C
C     NOW TEST THE MAGNITUDES OF DELTA X AND DELTA Y TO CHOOSE THE
C     APPROPRIATE COORDINATE SYSTEM A AND B. ALSO GET THE INCREMENTS
C     ALONG THE AXES AS REQUIRED
      IF (KDELA.LT.KDELB) GOTO 1
      INCX1=INCX2
      INCY1=0
      GOTO 2
    1 INCX1=KDELA
      KDELA=KDELB
      KDELB=INCX1
      INCX1=0
      INCY1=INCY2
C
C     NOW SET UP TWO TIMES DELTA A AND B, AND INITIAL DELTA I
    2 KDELB2=KDELB+KDELB
      KDELD2=KDELA+KDELA-KDELB2
      KDELI=KDELB2-KDELA
C
C     GO IN A LOOP FOR MAX TIMES OR UNTIL REACHING KDELA
      DO 4 J=1,MAX
C
C     TEST FOR DELTA A ZERO OR ONE, OR THE LINE FINISHED
      IF (J.GT.KDELA) GOTO 5
C     TEST THE SIGN OF DELTA I TO CHOOSE THE APPROPRIATE INCREMENTS
C     THEN COMPUTE NEXT POINT AND NEW DELTA I
      IF (KDELI.GE.0) GOTO 3
      KXRAY(J)=INCX1
      KYRAY(J)=INCY1
      KDELI=KDELI+KDELB2
      GOTO 4
    3 KXRAY(J)=INCX2
      KYRAY(J)=INCY2
      KDELI=KDELI-KDELD2
    4 CONTINUE
C
C     RUN OUT OF ARRAY SPACE
C     THE AVAILABLE MAX ELEMENTS ARE CORRECT
C     IF KDELA.GT.MAX THE SET IS TRUNCATED AFTER MAX POINTS
      LENG=MAX
      RETURN
C
C     FINISHED NORMALLY
C     LENG IS EITHER ZERO, OR LESS THAN MAX (AND POSITIVE)
    5 LENG=J-1
      RETURN
      END

C
C     SGNABS RETURNS THE SIGN OF I IN J AND THE ABS VALUE IN K
C
      SUBROUTINE SGNABS(I,J,K)
      INTEGER I,J,K
      IF (I.LT.0) GOTO 1
      J=+1
      K=I
      RETURN
    1 J=-1
      K=-I
      RETURN
      END
```

References:

BRESENHAM, J. E. (1965). Algorithm for computer control of a digital plotter, *IBM Systems Journal*, Vol. 4, No. 1, pp. 25–30.

STOCKTON, F. G. (1963). Algorithm 162, XYMove Plotting, *CACM*, Vol. 6, No. 4, p. 161.

Algorithm 55

AN INTERNAL MERGE SORT GIVING RANKS OF ITEMS

A. D. Woodall
North Staffordshire Polytechnic
Stafford

Author's note:

An internal sorting procedure may present its results in several ways, among them:

1. After calling the procedure the items may have been rearranged into order—e.g. Scowen (1965).
2. The procedure, while leaving the items untouched may provide the indices of the ordered items as successive entries of a rank vector—e.g. Boothroyd (1967).
3. The procedure may provide an ordered chain of items—e.g. Woodall (1969).

The first of these is the most useful form to have, if items need to be accessed other than in ordered sequence; it is also the only type which can be implemented with 'minimum storage'. The second is only a little less convenient for random access, but the third is of use only in giving the items in sequence.

The first type of procedure is generally the slowest, and the third is often fastest. The present algorithm uses chaining to accomplish the bulk of the sort, but presents the results in a rank vector.

The main part of the procedure is identical with *mergesort* (Woodall, 1970)—the array rank playing the part of *hds* in *mergesort*. When the sort has reached the commencement of the last pass, with two ordered lists remaining to be merged, the final ordering is recorded by putting the indices of the ordered items into the array rank.

On test on an ICL 4130 computer, *mergerank*, the present procedure, has been found to retain almost all of the advantages of *mergesort*. It is faster than *keysort* (Boothroyd, 1967) by about 20% with random data, and its speed improves when the data has some prior ordering (while *keysort* is becoming slower).

It is slower than *mergesort*, but the difference is very slight, and the results are in a more useful form.

References

BOOTHROYD, J. (1967). Algorithm 26, *The Computer Journal*, Vol. 10, p. 309.

SCOWEN, R. S. (1965). Algorithm 271, *CACM*, Vol. 8, p. 669.

WOODALL, A. D. (1969). Algorithm 43, *The Computer Journal*, Vol. 12, p. 406.

WOODALL, A. D. (1970). Algorithm 45, *The Computer Journal*, Vol. 13, p. 110.

```
procedure mergerank(n,a,rank); value n; integer n;
array a; integer array rank;
comment n is the number of keys to be sorted. The actual
parameter corresponding to rank should have bounds 1 to n.
a[1] to a[n] should contain the n keys to be sorted. After
mergesort has been called, a will be unchanged, but the order of
the keys in a will be given by rank, so that a[rank[i]] is a
non-decreasing function of i;
  begin real try, next, at 1, at 2; integer i, j, k, t, nol, try1, try2;
  integer array list[1:n];
  if n < 3 then goto SHORT;
  comment from here to the label MERGE the first pass links ad-
  jacent items into ordered lists, using existing runs in the data.
  A list may run either forward, if the run is in order or back-
  ward if it is in reverse order;
  j := t := 1; k := 2;
  try := a[1];
L2:next := a[k];
  if try > next then goto BACKWARD;
  rank [j] := t;
FORWARD:list[t] := k;
  if k = n then
    begin
    list[k] := 0; goto MERGE
    end;
  try := next; t := k;
  k := k + 1; next := a[k];
  if try ⩽ next then goto FORWARD;
  list[t] := 0;
L1: t := k; k := k + 1;
  j := j + 1;
  if t = n then
    begin
    rank[j] := n; list[n] := 0;
    goto MERGE
    end;
  try := next; goto L2;
BACKWARD:list[t] := 0;
BW:list[k] := t;
  if k = n then
    begin
    rank[j] := k; goto MERGE
    end;
  try := next; t := k;
  k := k + 1; next := a[k];
  if try ⩾ next then goto BW;
  rank[j] := t; goto L1;
MERGE: if j = 1 then goto SPECIAL;
  comment j = 1 means that the data was already in order, or
  reverse order;
  nol := j;
  comment nol is the number of lists after each pass;
  for t := 1, t + t while t < nol do k := t;
  j := t := k + k + 1 − nol;
  if j > 1 then goto LB;
  comment merging starts part-way through, at a point chosen
  to reduce the number of lists after the first merging pass to a
  power of 2;
LA: if nol = 2 then goto RANKPASS;
  t := 1; j := 1;
LB: try1 := rank[t]; try2 := rank[t + 1];
  at1 := a[try1]; at2 := a[try2];
  if at1 ⩽ at2 then
    begin
    rank[j] := try1; goto LL1
    end;
  rank[j] := try2;
LL2: k := list[try2];
  if k = 0 then
    begin
    list[try2] := try1; goto EXIT
    end;
  at2 := a[k];
  if at1 < at2 then
    begin
    list[try2] := try1; try2 := k
    end
  else
    begin
    try2 := k; goto LL2
    end;
LL1: k := list[try1];
```

```
    if k = 0 then
      begin
      list[try1] := try2; goto EXIT
      end;
    at1 := a[k];
    if at2 < at1 then
      begin
      list[try1] := try2; try1 := k;
      goto LL2
      end;
    try1 := k; goto LL1;
EXIT: j := j + 1; t := t + 2;
    if nol > t then goto LB;
    nol := j - 1; goto LA;
RANKPASS: try1 := rank[1]; try2 := rank[2];
    j := 0; at1 := a[try1];
    at2 := a[try2];
LOOP: j := j + 1;
    if at1 ⩽ at2 then
      begin
      rank[j] := try1; try1 := list[try1];
      if try1 = 0 then
        begin
        k := try2; goto RNX
        end;
      at1 := a[try1]; goto LOOP
      end
    else
      begin
      rank[j] := try2; try2 := list[try2];
      if try2 = 0 then
        begin
        k := try1; goto RNX
        end;
      at2 := a[try2]; goto LOOP
      end;
RNX: for j := j + 1 while j ⩽ n do
      begin
      rank[j] := k; k := list[k]
      end;
    goto RND;
SPECIAL: k := n + 1;
    if rank[1] = 1 then
      begin
      for i := 1 step 1 until n do rank[i] := i
      end
    else
    for i := 1 step 1 until n do rank[i] := k - i;
    goto RND;
SHORT: if n = 1 then rank[1] := 1
    else
      begin
      if a[1] > a[2] then
        begin
        rank[1] := 2; rank[2] := 1
        end
      else
        begin
        rank[1] := 1; rank[2] := 2
        end
      end;
RND: end mergerank
```

Algorithm 56

TO DISENTANGLE A CHAIN

A. D. Woodall
North Staffordshire Polytechnic
Stafford

Author's note:

Some recent sorting procedures (Woodall, 1969, 1970) have presented the results in the form of a chain. Since this is only useful if the results are to be accessed in sequence, the present procedure has been written to rearrange the sorted items into the order defined by the chain. The procedure, which uses little extra storage space, could clearly be used for reordering equal length records of any size. It operates by exchanging at most $n - 1$ pairs of items.

References

WOODALL, A. D. (1969). Algorithm 43, *The Computer Journal*, Vol. 12, p. 406.
WOODALL, A. D. (1970). Algorithm 45, *The Computer Journal*, Vol. 13, p. 110.

```
procedure untangle (n, a, link, start); value n, start; integer
n, start; array a; integer array link;
comment the array elements a[1] to a[n] contain the n items to
be reordered. After the call of untangle they will contain the
same items rearranged. The desired order is defined by start
and the integer array link. The first item (which will become
a[1]) is a[start], thereafter the successor of a[i] is a[link[i]] for
all i. If the final item is a[k], the value of link[k] is immaterial.
After calling untangle, start and n will be unchanged, but link
will contain rubbish;
  begin integer i, j, k, n1; real w;
  n1 := n - 1; j := start;
  for i := 1 step 1 until n1 do
    begin
LOOP: if j < i then
      begin
      j := link[j]; goto LOOP
      end;
    if j = i then j := link[j]
    else
      begin
      w := A[i]; A[i] := A[j];
      A[j] := w; k := link[j];
      link[j] := link[i]; link[i] := j;
      j := k
      end
    end
  end untangle
```

Algorithm 57

FIND

(The determination of the index value within some range for which a function assumes the minimal or maximal value subject to some condition; if no index value exists for which the condition is satisfied then go to a label.)*

T. O. M. Kronsjö
University of Birmingham

Author's note:

In mathematical programming it is often necessary to solve part problems as the following:

(*a*) Find the integer k for which the coefficient

$$c_k = \min_{\substack{j=1,\ldots,n \\ c_j<0}} c_j$$

which may be recognised as the problem of the primal simplex method of linear programming of finding the index k of the nonbasic variable with the minimal negative reduced

* The investigation leading to Algorithms 57 to 61 was supported by the Swedish Council for Social Science Research.

cost coefficient, or that of the dual simplex method of finding the index k of the row of the basic variable with minimal negative value.

(*b*) Find the integer l for which the ratio

$$\frac{b_l}{a_{lk}} = \min_{\substack{i=1,\ldots,m \\ a_{ik}>0}} \frac{b_i}{a_{ik}}$$

which again may be recognised as the problem of the primal simplex method of finding the row l of the basic variable that first will become zero when the nonbasic variable k is increased, or that of the dual simplex method of finding the nonbasic variable l that should be increased in order to enable the negative basic variable in the kth row to become equal to zero at minimal cost.

This extremely common operation in mathematical programming may be solved by the following procedure, which closely follows the procedure *TROUVER* by P. Broise (1965).

```
procedure FIND(r) the integer which corresponds to the
minimal value of the expression: (f) when the index: (i)
assumes successive integer values from: (s) to: (t) subject to the
condition: (b) if no such integer r exists then go to: (e);
value s, t; integer r, i, s, t; real f; Boolean b; label e;
  begin real min;
  min := 10^20;
  for i := s step 1 until t do
  if b then
    begin
    if f < min then
      begin
      min := f; r := i
      end
    end;
  if min = 10^20 then goto e
  end procedure FIND;
```

Note that if the formulation

if *b* **and** $f <$ *min* **then**

had been used above this would have led to the complete condition being evaluated, in spite of the fact that f should not be evaluated if b is not satisfied. This may lead to odd types of errors, e.g. if the actual Boolean expression is $a[i, k] > 0$ and the actual function f is $b[i]/a[i, k]$ and the particular $a[i, k]$ coefficient is equal to 0, this would lead to a failure indication of 'division by zero' or 'overflow'.

The above procedure was tested on the ICL–KDF9 computer and the modification of the note made by A. M. Irving, National Economic Planning Unit, Faculty of Commerce and Social Science, University of Birmingham.

The above problems may then be solved by the procedure calls:

FIND(k) the integer which gives the minimal coefficient: (c[j]) when the index: (j) assumes successive integer values from: (1) to: (n) subject to the condition: (c[j] < 0) if no such integer k exists then goto: (finite solution);

FIND(l) the integer which corresponds to the minimal ratio: (b[i]/a[i, k]) when the index: (i) assumes successive integer values from: (1) to: (m) subject to the condition: (a[i, k] > 0) if no such integer l exists then goto: (infinite solution);

To facilitate the elaboration of experimental programs it is useful to modify the procedure so that it may search for the minimum and the maximum of the expression depending upon whether a parameter *opt* is given the value -1 or $+1$, respectively.

It is also valuable to let the procedure identifier assume the extremal value of the expression, if it exists.

The essential difference if the procedure is to search for the minimum or maximum can be summarised by the following table:

MINIMUM (*opt* = − 1)	MAXIMUM (*opt* = + 1)
min: = large positive value;	*max:* = large negative value;
if $f <$ *min* **then**	**if** $f >$ *max* **then**

These conditions can be reformulated as:

$f - min < 0$	$f - max > 0$
or	or
$-1(f - min) > 0$	$1(f - max) > 0$

The procedure *FIND* can thus be reformulated to search for the maximum or minimum using the parameter *opt* by combining the above statements and conditions into the single statement:

extremum := large negative value × *opt*;

and the condition

if *opt* × (*f* − *extremum*) > 0 **then**

The corresponding procedure declaration is given below.

The procedure was tested on the ICL–KDF9 computer and an improvement suggested by V. Kolarov, Deputy Chief of the Mathematical Simulation of Economic Processes Department of the State Committee of Planning, Sofia, Bulgaria.

Reference

BROISE, P. (1965). *Le langage ALGOL, Applications à des problèmes de Recherche Opérationelle*, Dunod, Paris, p. 79, procedure TROUVER.

```
real procedure FIND (r) the integer which for the parameter:
(opt) equal to either minus one or plus one corresponds to the
minimum or the maximum value respectively of the expression:
(f) when the index: (i) assumes successive integer values from:
(s) to: (t) subject to the condition: (b) if no such integer r exists
then goto: (e);
value opt, s, t; integer r, opt, i, s, t; real f; Boolean b; label e;
  begin real extremum;
  extremum:= − 10 20 × opt;
  for i := s step 1 until t do
  if b then
    begin
    if opt × (f − extremum) > 0 then
      begin
      extremum := f;
      r := i
      end
    end;
    if extremum = − 10^20 × opt then goto e
    else FIND := extremum
  end real procedure FIND;
```

Algorithm 58

AN ILLUSTRATIVE PRIMAL SIMPLEX LINEAR PROGRAM

T. O. M. Kronsjö
University of Birmingham

Author's note:

An extremely simplified linear program procedure based on the primal simplex method may be formulated using not more than nine statements (semicolons) in the procedure body.

The linear programming problem is assumed to be

$$\text{Extremum } \{cx + d \mid Ax \geqslant b,\ x \geqslant 0\} \qquad (1)$$

where Extremum represents either minimisation or maximisation and $b \leqslant 0$.

In order to obtain the desired brief formulation it is suitable to define (1) as the problem

$$\text{Extremum}\ \{\sum_{j=1}^{n-m} c_j x_j + d \mid \sum_{j=1}^{n-m} a_{ij} x_j \geqslant b_i \quad (i = 1, \ldots, m) \quad x_j \geqslant 0 \quad (j = 1, \ldots, n-m)\} \quad (2)$$

A variable x_0 representing the value of the objective function and slack variables $x_j (j = n - m + 1, \ldots, n)$ may be introduced in order to obtain the canonical problem

Extremum x_0 subject to

$$\begin{aligned} x_0 - \sum_{j=1}^{n-m} c_j x_j &= d \\ -\sum_{j=1}^{n-m} a_{ij} x_j + x_{n-m+i} &= -b_i \quad (i = 1, \ldots, m) \\ x_j &\geqslant 0 \quad (j = 1, \ldots, n) \end{aligned} \quad (3)$$

A starting basis $x_0 = d$ and $x_{n-m+i} = -b_i$ is immediately available, which is also a feasible solution because of the assumption made in (1) that $b \leqslant 0$.

To simplify the formulation of these iterations it is useful to introduce an array $A[0{:}m, 0{:}n]$, the elements of which contain all the coefficients and constants of problem (3) except for the coefficients of the x_0 variable which are never affected by the pivoting operations and therefore need not be explicitly remembered. The detailed content of the array A is as follows:

$$\begin{aligned} &A[0, 0] = d \\ &A[0, 1{:}n-m] = -c[1{:}n-m] \\ &A[0, n-m+1{:}n] = 0 \\ &A[1{:}m, 0] = -b[1{:}m] \\ &A[1{:}m, 1{:}n-m] = -a[1{:}m, 1{:}n-m] \\ &A[1{:}m, n-m+1{:}n] = E \end{aligned}$$

where 0 denotes a row vector of zeros and E an identity matrix of appropriate dimensions.

The names of the variables in the current basis may be stored in the integer array $I[0{:}m]$ with initial values

$$\begin{aligned} &I[0] = 0 \\ &I[i] = n - m + i \quad (i = 1, \ldots, m) \end{aligned}$$

The problem (3) may then be solved by iterations of the primal simplex algorithm upon the problem (4), i.e. the simplex tableau

List of basic variables	Extremum x_0
$I[i]$	$x_0 + \sum_{j=1}^{n} A_{0j} x_j = A_{00}$
$I[i]$	$\sum_{j=1}^{n} A_{ij} x_j = A_{i0} \quad (i = 1, \ldots, m)$
	$x_j \geqslant 0$

(4)

which from iteration to iteration is kept in canonical form with respect to the basic variables

$$x_j \quad j \in I[0{:}m]$$

The procedure finds the minimum or maximum of x_0 depending on whether a parameter *opt* is equal to -1 or $+1$, respectively. In solving the problem the procedure calls on the previously mentioned procedure *FIND* and gives on exit

either $k = 0$

in which case the linear program has a finite optimal solution given by

$$x_{I[i]} = A[i, 0] \quad (5)$$

or $1 \leqslant k \leqslant n$

in which case the linear program has an infinite solution given by

$$x_{I[i]} = A[i, 0] - A[i, k] \times \theta \quad (i = 0, \ldots, m) \quad (6)$$

$\frac{z}{k} = \theta$, $\theta \to \infty$

Rounding errors

In linear programming tableaus the pivoting operations aim at producing a column of zeros except for a unit coefficient in one position. Due to inevitable rounding errors, the resulting elements will often be absolutely small numbers instead of zeros. If they in due course are multiplied by larger numbers, they are likely to make a considerable contribution to the cumulative effect of the rounding errors of each iteration. To alleviate this tendency, it is normal to set a number resulting from a pivot operation equal to zero if it is less than a given 'drop' coefficient.

This may be done by the following straightforward procedure

```
real procedure SIGNIFICANT (coefficient, criteria);
value coefficient, criteria; real coefficient, criteria;
SIGNIFICANT := if abs(coefficient) > criteria then
    coefficient else 0;
```

To prevent the computer program from continuing the iterations indefinitely or obtaining grossly inaccurate solutions due to it not recognising a finite or infinite optimal solution, or the existence or nonexistence of a feasible solution, as a consequence of rounding errors, a number of conventional criterias for defining what is a true 'zero' or not, are used as suggested in an investigation by P. Wolfe (1965).

The procedure was tested on ICL–KDF9 and some shortcomings eliminated by R. Z. Meron, National Economic Planning Unit, Faculty of Commerce and Social Science, University of Birmingham.

References

DANTZIG, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press.

BROISE, P. (1965). *Le Langage ALGOL, Applications à des problèmes de Recherche Opérationelle*, Dunod, Paris, pp. 63–81.

WOLFE, P. (1965). Error in the Solution of Linear Programming Problems, in Louis B. Rall (Ed.), *Error in Digital Computation*, Vol. 2, United States Army Mathematics Research Center, The University of Wisconsin, Publication No. 15, John Wiley and Sons, Inc., New York, pp. 271–284.


```
procedure PRIMAL SIMPLEX LP (opt, m, n, A, I, k, e0, e1,
e2, FIND, SIGNIFICANT);
value opt, m, n, e0, e1, e2; integer opt, m,n,k;
array A; integer array I;
real e0, e1, e2; real procedure FIND, SIGNIFICANT;
  begin integer i, j, l;
  comment determination of the index k of the variable that
  should enter the basis. e1 (e.g. 10 - 3) is used instead of a
  zero in order to reduce the possibility of perpetual basis
  changes arising from cumulating rounding errors;
NEXT ITERATION:
  FIND (k, -opt, A[0, j], j, 1, n, -opt × A[0, j] > e1, FINITE
  SOLUTION);
  comment determination of the row l of the basic variable
  that should leave the basis. e2 (e.g. 10 - 4) is used instead
  of a zero in order to reduce the effect of rounding errors;
  FIND (l, -1, A[i, 0] / A[i, k], i, 1, m, A[i, k] > e2,
```

INFINITE SOLUTION);
comment *change of basis, transformation of the equations into canonical form with respect to the new basis variables, a coefficient which arises from a subtraction and is absolutely less than e0 (e.g.* $_{10} - 7$*) is set equal to zero*;
$I[l] := k$;
for $j := 0$ **step** 1 **until** $k - 1, k + 1$ **step** 1 **until** n, k **do**
$A[l, j] := A[l, j] / A[l, k]$;
for $i := 0$ **step** 1 **until** $l - 1, l + 1$ **step** 1 **until** m **do**
for $j := 0$ **step** 1 **until** $k - 1, k + 1$ **step** 1 **until** n, k **do**
$A[i, j] := SIGNIFICANT\ (A[i, j] - A[l, j] \times A[i, k], e0)$;
goto *NEXT ITERATION*;
FINITE SOLUTION: $k := 0$;
INFINITE SOLUTION:
end *of procedure PRIMAL SIMPLEX LP*;

Algorithm 59

AN ILLUSTRATIVE PRIMAL SIMPLEX LINEAR PROGRAM USING MULTIPLIERS

T. O. M. Kronsjö
University of Birmingham

Author's note:

The preceding algorithm may be modified so that instead of explicitly transforming the equations into canonical form with respect to the current basis variables, it would record in a mnemonic table (the inverse) what fractions of each equation that it has to add to obtain the transformed equations. (Compare Dantzig, 1963.) Using the information of the mnemonic table (the inverse), the procedure could then calculate the reduced cost coefficients of the non-basic variables, the transformed coefficients of the entering non-basic variable, and the transformed right-hand constants.

(To save operations, it is, however, useful to maintain the explicit form of the transformed right-hand constants as in the previous algorithm.)

The following procedure solves a linear programming problem of the form minimise or maximise x_0 depending on the value of a parameter *opt* being either -1 or $+1$, respectively, and subject to:

$$x_0 + \sum_{j=1}^{n} A_{0j}x_j = A_{00}$$

$$\sum_{j=1}^{n} A_{ij}x_j = A_{i0} \qquad (i = 1, \ldots, m)$$

$$x_j \geqslant 0 \qquad (j = 1, \ldots, n)$$

for which a feasible basic solution is known consisting of the variables with index

$$j \in I[0{:}m]$$

with the inverse

$$U[0{:}m, 0{:}m]$$

and with the transformed form of the right-hand side, i.e. the current values of the basic variables in

$$U[0{:}m, -1]$$

The transformed form of an entering variable is obtained in

$$U[0{:}m, -2]$$

The procedure gives upon exit

either $k = 0$

in which case the linear program has a finite optimal solution given by

$$x_{I[i]} = U[i, -1] \qquad (i = 0, \ldots, m)$$

or $1 \leqslant k \leqslant n$

in which case the linear program has an infinite solution given by

$$x_{I[i]} = U[i, -1] - U[i, -2] \times t \qquad (i = 0, \ldots, m)$$
$$\underset{\theta \to \infty}{x_k} = \theta$$

Assignment of the result of a summation

In order to achieve a compact formulation of the linear program in dealing with expressions like

if $\sum_{j=0}^{m} U[i, j] \times A[j, k] > 0$ **then**

$$theta := U[i, -1] / \sum_{j=1}^{m} U[i, j] \times A[j, k]$$

where the same sum is required at different places, it is extremely useful to define a **real procedure** *SUM*(r, i, s, t, f); which sets the identifier *SUM* and the formal variable r equal to $\sum_{i=s}^{t} f$ where f is an unspecified function of i, which will be specified when calling upon the procedure.
The procedure is straightforward

real procedure *SUM*(r, i, s, t, f);
value s, t; **real** r, f; **integer** i, s, t;
begin
$r := 0$;
for $i := s$ **step** 1 **until** t **do** $r := r + f$;
$SUM := r$
end *real procedure SUM*;

To obtain a faster execution of the procedure it may be useful to use a local working variable w. The corresponding procedure body would then be

begin real w;
$w := 0$;
for $i := s$ **step** 1 **until** t **do** $w := w + f$;
$r := w$;
$SUM := w$
end *real procedure SUM*;

Using this summation procedure, the modification of the preceding procedure is readily undertaken and yields the following brief algorithm.

The procedure was tested on ICL-KDF9 and some shortcomings eliminated by M. Walmsley, Ministry of Planning and Economic Development, Entebbe, Uganda.

Reference

DANTZIG, G. B. (1963). *Linear programming and extensions*, Princeton University Press, Chapter 9.

procedure *PRIMAL SIMPLEX LP USING MULTIPLIERS* (*opt, m, n, I, A, U, k, e0, e1, e2, FIND, SIGNIFICANT, SUM*);
value *opt, m, n, e0, e1, e2*; **integer** *opt, m, n, k*; **integer array** *I*;
array *A, U*; **real** *e0, e1, e2*;
real procedure *FIND, SIGNIFICANT, SUM*;
begin integer *i, j, l*; **real** *w*;
comment *the determination of the entering variable k, using e1 (e.g.* $_{10} - 3$*) instead of zero to reduce the possibility of perpetual basis changes arising from cumulating rounding errors*;
NEXT ITERATION:
$U[0,-2] := FIND\ (k, -opt, w, j, 1, n, -opt \times SUM\ (w, i, 0, m, U[0,i] \times A[i,j]) > e1, FINITE\ SOLUTION)$;
comment *determination of row l of the basic variable that should leave the basis, at the same time the transformed column* A_k *is calculated, e2 (e.g.* $_{10} - 4$*) is again used instead of zero to reduce the effect of rounding errors*;
$FIND\ (l, -1, U[i, -1] / U[i, -2], i, 1, m, SUM(U[i, -2], j, 0, m, U[i, j] \times A[j, k]) > e2, INFINITE\ SOLUTION)$;
comment *change of basis, transforming the constants and*

updating the inverse, a coefficient which arises from a subtraction and is absolutely less than e0 (e.g. $_{10} - 7$*) is set equal to zero*;
$I[l] := k$;
for $j := -1$ **step** 1 **until** m **do**
$U[l, j] := U[l, j] / U[l, -2]$;
for $i := 0$ **step** 1 **until** $l - 1, l + 1$ **step** 1 **until** m **do**
for $j := -1$ **step** 1 **until** m **do**
$U[i, j] := SIGNIFICANT\ (U[i, j] - U[l, j] \times U[i, -2], e0)$;
goto *NEXT ITERATION*;
FINITE SOLUTION: $k := 0$;
INFINITE SOLUTION:
end *procedure PRIMAL SIMPLEX LP USING MULTIPLIERS*;

The linear program

$$\underset{x}{\text{Min}}\ \{cx + d \mid Ax \geqslant b,\ x \geqslant 0\}$$

with m constraints, $n - m$ variables, elements of A being a_{ij} and $b \leqslant 0$ may be solved, setting *opt* equal to -1, by choosing the slack variables as basic variables and defining an array A as follows

$$\begin{aligned}
&A[0, 0] = d\\
&A[0, 1{:}n - m] = -c[1{:}n - m]\\
&A[0, n - m + 1{:}n] = 0\\
&A[1{:}m, 0] = -b[1{:}m]\\
&A[1{:}m, 1{:}n - m] = -a[1{:}m, 1{:}n - m]\\
&A[1{:}m, n - m + 1{:}n] = E
\end{aligned}$$

where 0 denotes a row vector of zeros and E an identity matrix of appropriate dimensions,

$$\begin{aligned}
&I[0] = 0\\
&I[i] = n - m + i \qquad (i = 1, \ldots, m)\\
&U[0{:}m, -2] = \text{any value}\\
&U[0, -1] = d\\
&U[0{:}m, 0{:}m] = E\\
&U[1{:}m, -1] = -b[1{:}m]
\end{aligned}$$

where E denotes an identity matrix of appropriate dimensions, and finally $k =$ any value.

Algorithm 60

AN ILLUSTRATIVE PRIMAL SIMPLEX LINEAR PROGRAM USING MULTIPLIERS AND SINGLE DIMENSIONAL ARRAYS

T. O. M. Kronsjö
University of Birmingham

Author's note:

In computer problems involving similar operations to be performed upon a variable number r of coefficient matrices C_i of different dimensions $m[i], n[i] (i = 1, \ldots, n)$ it would be extremely useful to be able to declare these matrices in the form of a single matrix

$$C[1{:}r, 1{:}m[i], 1{:}n[i]]$$

where i refers to the particular value of the first index. This is, however, not possible in either ALGOL 60 or FORTRAN IV. This limitation of these computer languages may however be overcome by declaring multidimensional arrays as single dimensional ones, and introducing corresponding changes in the programs.

The approach may be illustrated upon the basis of the previous algorithm *PRIMAL SIMPLEX LP USING MULTIPLIERS*.

The columns of its matrix $A[0{:}m, 0{:}n]$ may be stored after each other in the form of a column vector $A[0{:}m + (m + 1) \times n]$. The earlier element $A[i, j]$ will then be contained in the present element $A[i + (m + 1) \times j]$, as evident from the following illustrations and derivation.

	0	1	j	n
0	0, 0	0, 1		
i	i, 0		i, j	
m	m, 0			m, n

Fig. 1. The indices of the earlier array $A[0{:}m, 0{:}n]$.

0	i	m	$m + 1$	k		
0, 0	i, 0	m, 0	0, 1	i, j		m, n

Fig. 2. The relationship between the indices of the present array $A[0{:}m + (m + 1) \times n]$ and those of the earlier array $A[0{:}m, 0{:}n]$.

To the index i, j of the two dimensional array A there will correspond an index k of the single dimensional array A, which may be expressed as a linear function of i and j as follows:

$$k = a + b \times i + c \times j \tag{1}$$

It is required that the earlier elements $A[0, 0]$, $A[i, 0]$ and $A[0, 1]$ should be contained in the present elements $A[0]$, $A[i]$ and $A[m + 1]$, respectively, which together with (1) gives the conditions

$$0 = a + b \times 0 + c \times 0 \quad \text{i.e. } a = 0 \tag{2}$$

$$i = 0 + b \times i + c \times 0 \quad \text{i.e. } b = 1 \tag{3}$$

$$m + 1 = 0 + 1 \times 0 + c \times 1 \quad \text{i.e. } c = m + 1 \tag{4}$$

and hence (1) may be expressed as

$$k = i + (m + 1) \times j \tag{5}$$

Similarly the rows of the matrix $U[0{:}m, -2{:}m]$ may be stored after each other as a row vector $U[0{:}2 + (m + 3) \times m + m]$, where the earlier element $U[i, j]$ will be contained in the present element $U[2 + (m + 3) \times i + j]$.

The relationships between the index k and the index i, j may similarly be obtained by noting that the earlier elements $U[0, -2]$, $U[0, -1]$ and $U[1, -2]$ should now be contained in the present elements $U[0]$, $U[1]$ and $U[m + 3]$ which together with (1) give the conditions

$$0 = a + b \times 0 + c \times (-2) \quad \text{i.e. } a = 2c \tag{6}$$

$$1 = a + b \times 0 + c \times (-1) \quad \text{i.e. } a = c + 1 \tag{7}$$

It follows from (6) and (7) that $2c = c + 1$ and hence that $c = 1$ and $a = 2$

$$m + 3 = 2 + b \times 1 + 1 \times (-2) \quad \text{i.e. } b = m + 3 \tag{8}$$

and hence

$$k = 2 + (m + 3) \times i + j \tag{9}$$

The above index relations (5) and (9) may be used to define a procedure *PRIMAL SIMPLEX LP USING MULTIPLIERS AND SINGLE DIMENSIONAL ARRAYS* based upon the statements of the earlier elaborated procedure *PRIMAL SIMPLEX LP USING MULTIPLIERS*.

The procedure was tested on the ICL–KDF9 computer and some shortcomings eliminated by V. Kolarov, Deputy Chief of the Mathematical Simulation of Economic Processes Department of the State Committee of Planning, Sofia, Bulgaria.

```
procedure PRIMAL SIMPLEX LP USING MULTIPLIERS
AND SINGLE DIMENSIONAL ARRAYS (opt, m, n, I, A,
U, k, e0, e1, e2, FIND, SIGNIFICANT, SUM);
value opt, m, n, e0, e1, e2; integer opt, m, n, k;
integer array I; array A, U; real e0, e1, e2;
real procedure FIND, SIGNIFICANT, SUM;
  begin integer i, j, l; real w;
  comment The determination of entering variable k, using e1
  (e.g. 10 - 3) instead of zero to reduce the possibility of
  perpetual basis changes arising from cumulating rounding
  errors;
NEXT ITERATION: U[0] := FIND (k, -opt, w, j, 1, n,
  -opt × SUM(w, i, 0, m, U[i + 2] × A[i + j × (m + 1)])
  > e1, FINITE SOLUTION);
  comment The determination of row l of the basic variable
  that should leave the basis, at the same time the transformed
  column A_k is calculated, e2 (e.g. 10 - 4) is again used instead
  of zero to reduce the effect of rounding errors;
  FIND (l, -1, U[i × (m + 3) + 1] / U[i × (m + 3)],i,1,m,
  SUM(U[i × (m + 3)], j,0,m, U[i × (m + 3) + j + 2] ×
  A[j + k × (m + 1)]) > e2, INFINITE SOLUTION);
  comment Change of basis, transforming the constants and
  updating the inverse, a coefficient which arises from a sub-
  traction and is less than e0 (e.g. 10 - 7) is set equal to zero;
  I[l] := k;
  for j := - 1 step 1 until m do
  U[l × (m + 3) + j + 2] := U[l × (m + 3) + j + 2] /
  U[l × (m + 3)];
  for i := 0 step 1 until l - 1, l + 1 step 1 until m do
  for j := - 1 step 1 until m do
  U[i × (m + 3) + j + 2] := SIGNIFICANT(U[i × (m + 3)
  + j + 2] - U[l × (m + 3) + j + 2] × U[i × (m + 3)],
  e0);
  goto NEXT ITERATION;
FINITE SOLUTION: k := 0;
INFINITE SOLUTION:
  end procedure PRIMAL SIMPLEX USING MULTIPLIERS
  AND SINGLE DIMENSIONAL ARRAYS;
```

The above linear program procedure may be reformulated in a form involving fewer index multiplications by redefining the procedures *SUM* and *FIND* to include a formal step parameter *u* between the lower and upper bound parameters *s* and *t*. This procedure was tested on ICL-KDF9 and some shortcomings eliminated by V. Kolarov, Deputy Chief of the Mathematical Simulation of Economic Processes Department of the State Committee of Planning, Sofia, Bulgaria.

```
procedure PRIMAL SIMPLEX LP USING MULTIPLIERS
AND SINGLE DIMENSIONAL ARRAYS (opt, m, n, I, A,
U, k, e0, e1, e2, FIND, SIGNIFICANT, SUM);
value opt, m, n, e0, e1, e2; integer opt, m, n, k;
integer array I; array A, U; real e0, e1, e2;
real procedure FIND, SIGNIFICANT, SUM;
  begin integer i, j, l; real w;
NEXT ITERATION:
  U[0] := FIND(k, -opt, w, j, m + 1, m + 1, n × (m + 1),
  -opt × SUM(w, i, 0, m, U[i + 2] × A[i + j]) > e1,
  FINITE SOLUTION);
  FIND(l, -1, U[i+1]/U[i], i, m+3, m+3, m×(m+3),
  SUM(U[i], j, 0, m, U[i + j + 2] × A[j + k]) > e2,
  INFINITE SOLUTION);
  I[l ÷ (m + 3)] := k ÷ (m + 1);
  for j := l + 1 step 1 until l + m + 2 do
  U[j] := U[j] / U[l];
  for i := 0 step m + 3 until (l - 1) × (m + 3),
  (l + 1) × (m + 3) step m + 3 until m × (m + 3) do
  for j := 1 step 1 until m + 2 do
  U[i + j] := SIGNIFICANT(U[i + j] - U[l + j] ×U[i],
  e0);
  goto NEXT ITERATION;
FINITE SOLUTION: k := 0;
INFINITE SOLUTION:
  end procedure PRIMAL SIMPLEX USING MULTIPLIERS
  AND SINGLE DIMENSIONAL ARRAYS;
```

Algorithm 61

AN ILLUSTRATIVE SELF-DUAL PARAMETRIC SIMPLEX LINEAR PROGRAM USING MULTIPLIERS

T. O. M. Kronsjö
University of Birmingham

Author's note:

An ALGOL procedure for the solution of a linear program using the self-dual parametric algorithm (Dantzig, 1963, section 11.3) may by assiduous use of variants of the above procedures *FIND*, *SIGNIFICANT* and *SUM* be formulated using only about 15 statements.

A formulation of the linear programming problem solved by the self-dual parametric simplex method

The self-dual parametric simplex algorithm solves the problem

$$\begin{aligned} \operatorname*{Min}_x \{cx + d \mid & \qquad \text{Dual variables} \qquad (1)\\ Ax &\geqq b \qquad u\\ x &\geqq 0\} \end{aligned}$$

by considering the parametric problem

$$\begin{aligned} \operatorname*{Min}_x \{(c + c^*p)x + d \quad \mid & \qquad \text{Dual variables} \qquad (2)\\ Ax &\geqq b + b^*q \qquad u\\ x &\geqq 0 \quad \} \end{aligned}$$

where $b^* < 0$ and $c^* > 0$.

It is evident that if p, q are chosen sufficiently large then an optimal and feasible solution may easily be obtained to the parametric problem, e.g. if $b + b^*q \leqq 0$ and $c + c^*p \geqq 0$ then an optimal solution is $x = 0$, $u = 0$.

The above problem may be formulated as Min x_0 subject to

$$\begin{aligned} & x_0 - \sum_{j=1}^{n-m} (c_j + c_j^*p)x_j = d\\ & -\sum_{j=1}^{n-m} a_{ij}\, x_j + x_{n-m+i} = -\, b_i - b_i^*q \quad (i = 1, \ldots, m)\\ & x_j \geqq 0 \qquad\qquad (j = 1, \ldots, n) \qquad (3) \end{aligned}$$

This problem may be reformulated as Min $x_0 + px_{-1}$ subject to

$$p\begin{pmatrix} x_{-1} & -\sum_{j=1}^{n-m} c_j^* x_j & = & 0 + \begin{bmatrix} 0 \\ \\ 0 \\ \\ -b_i^* \end{bmatrix} \end{pmatrix}^{-1} \begin{matrix} \text{Row No.} \\ \\ q \\ \\ \end{matrix} \begin{matrix} 0 \\ 1 \\ \vdots \\ m \end{matrix}$$

$$\begin{aligned} x_0 - \sum_{j=1}^{n-m} c_j x_j &= d \\ -\sum_{j=1}^{n-m} a_{ij}x_j + x_{n-m+i} &= -\, b_i \end{aligned}$$

$$x_j \geqq 0 \qquad (4)$$

Column No.	1	...	n	0	-1

This reformulation yields a possible way of recording the problem for computation, the values of the parameters p and q being separately remembered. The actual cost coefficients

will then be given by the entries in the 0th row plus p times the corresponding entries in the -1st row, similarly the actual right-hand side constants will then be given by the entries in the -2nd column plus q times the corresponding entries in the -3rd column.

The problem (4) is in canonical form with respect to the variables x_{-1}, x_0, $x_{n-m+i} (i = 1, \ldots, m)$. Therefore, an initial solution may easily be obtained by setting the (nonbasic) variables $x_j = 0 (j = 1, \ldots, n-m)$ and obtaining the corresponding values for the basic variables $x_{-1} = 0$, $x_0 = d$, $x_{n-m+i} = -b_i - b_i^* q (i = 1, \ldots, m)$. This solution will be a feasible solution if q is chosen so that each $b_i^* q$ is smaller than or equal to the negative of the corresponding b_i constant $(i = 1, \ldots, m)$. The basic solution will also be optimal if all the cost coefficients $-(c_j + c_j^* p)$ of the nonbasic variables are negative, which will be the case if p is chosen so that each $c_j^* p$ is greater than or equal to the negative of the corresponding c_j coefficient $(j = 1, \ldots, n-m)$.

As the solution process proceeds the original equations (3) or (4) will be transformed, and the corresponding equations become Min $x_0 + px_{-1}$ subject to

$$\begin{aligned} p\Big(x_{-1} \quad + \sum_{j\in J} \bar{c}_j^* x_j \quad &= \bar{b}_{-1} + \begin{bmatrix} \bar{b}_{-1}^* \end{bmatrix} \Big) \\ x_0 + \sum_{j\in J} \bar{c}_j x_j \quad &= \bar{b}_0 \quad + \begin{bmatrix} \bar{b}_0^* \end{bmatrix} \quad q \\ \sum_{j\in J} \bar{a}_{ij} x_j + x_k \quad &= \bar{b}_i \quad + \begin{bmatrix} \bar{b}_i^* \end{bmatrix} \quad (i=1,\ldots,m) \\ x_j = 0, j \in J & \\ x_k \geqslant 0, k \in I & \end{aligned} \tag{5}$$

where the set of indices of the basic variables is denoted by I and that of the nonbasic variables by J.

At any iteration of the simplex algorithm the reduced cost coefficient for a basic variable is zero, except if it itself expresses the value of the objective function in which case it is equal to unity. The reduced cost coefficient for any variable $x_j (j = 1, \ldots, n)$ consists of the sum $\bar{c}_j + \bar{c}_j^* p$. As $p \gtrsim 0$ this means that both terms $\bar{c}_j$ and $\bar{c}_j^*$ must be equal to zero for the basic variables $(j \in I)$. This may easily be achieved by using the above formulation (5) of the equation system, always retaining x_{-1} and x_0 in the basis, and for each change of basis transforming the equation system into canonical form with respect to the current basic variables, using the standard rules of linear programming.

The linear programming matrices required by the simplex method using multipliers (Kronsjö, 1970) may be set up having separate rows and columns for the coefficients associated with the p and q parameters, respectively.

	-1	0	1	j	$n-m$	$n-m+1$	n	
-1	0	0	$-c_j^*$			0		
0	0	d	$-c_j$			0		
1								
i	$-b_i^*$	$-b_i$	$-a_{ij}$			E		$= A$
m								

	-4	-3	-2	-1	0	1	m				
-1		0	0	1	0	0		1	-1		
0		0	d	0	1			0	0		
1								1	$n-m+1$		
i	any	$-b_i^*$	$-b_i$	0	0	E	$= U$	i	$n-m+i$	$= I$	
m								m	$n-m+m$		

where E denotes an identity matrix of appropriate dimensions.

Primal simplex iterations

It is desirable that the final procedure should function for both minimisation and maximisation according to a parameter *opt* being given the value -1 and $+1$, respectively, without requiring any further changes in the data given. The two cases will be considered in turn and then summarised by using the values of the parameter *opt*.

(a) Minimisation

Assume that at the current iteration $p = p^* > 0$ and that all the reduced cost coefficients are nonpositive

$$\bar{c}_j + \bar{c}_j^* p^* \leqq 0 \qquad (j = 1, \ldots, n) \tag{6'}$$

Decreasing the parameter $p = p^* > 0$ may affect the sign of the reduced cost coefficient $\bar{c}_j$ and $\bar{c}_j^* p$ depending upon the sign of the coefficients $\bar{c}_j$ and $\bar{c}_j^*$. The effects may be summarised in the form of the following table:

sign $(\bar{c}_j + \bar{c}_j^* p)$ for $0 \leqslant p < p^*$

	$\bar{c}_j^* > 0$	$\bar{c}_j^* = 0$	$\bar{c}_j^* < 0$
$\bar{c}_j > 0$	(+)	(+)	$- \rightarrow +$
$\bar{c}_j = 0$	(+)	0	$-$
$\bar{c}_j < 0$	$-$	$-$	$-$

(7')

The cases indicated by (+) are not applicable as they contradict the assumption of (6') that all the reduced cost coefficients are nonpositive for $p = p^*$. Therefore only reduced cost coefficients satisfying

$$\bar{c}_j > 0 \text{ and } \bar{c}_j^* < 0 \tag{8'}$$

may become positive when $p \gtrsim 0$ is decreased. The values of p for which the reduced cost coefficients satisfying (8') change sign may be obtained by solving (6') for p remembering that $\bar{c}_j^* < 0$

$$p \geqslant p_j = \frac{-\bar{c}_j}{\bar{c}_j^*} \tag{9'}$$

For all p satisfying (9') the inequality (6') will be true. The index of the first variable for which the sign of its reduced cost coefficient will change from negative to positive when the parameter p is decreased may be denoted by k. The value of p for which this occurs may be denoted by p_k and obtained from

$$p_k = \frac{-\bar{c}_k}{\bar{c}_k^*} = \operatorname*{Max}_j \left\{ \frac{-\bar{c}_j}{\bar{c}_j^*} \,\middle|\, \bar{c}_j > 0 \text{ and } \bar{c}_j^* < 0 \right\} \tag{10'}$$

This definition of p_k will in this case of minimisation either give a positive value or no value at all.

(b) Maximisation

Assume that at the current iteration $p = p^* < 0$ and that all the reduced cost coefficients are nonnegative

$$\bar{c}_j + \bar{c}_j^* p^* \gtrsim 0 \qquad (j = 1, \ldots, n) \tag{6''}$$

Increasing the parameter $p = p^* < 0$ may only affect the sign of the expression if

$$\bar{c}_j < 0 \text{ and } \bar{c}_j^* < 0 \tag{8''}$$

as evident from the table following.

sign $(\bar{c}_j + \bar{c}_j^* p)$ for $p^* < p \leqslant 0$

	$\bar{c}_j^* > 0$	$\bar{c}_j^* = 0$	$\bar{c}_j^* < 0$
$\bar{c}_j > 0$	+	+	+
$\bar{c}_j = 0$	(−)	0	+
$\bar{c}_j < 0$	(−)	(−)	+ → −

(7″)

The values of p for which the reduced cost coefficients satisfying (8″) change sign may be obtained by solving (6″) for p remembering that $\bar{c}_j^* < 0$, to obtain

$$p \leqslant p_j = \frac{-\bar{c}_j}{\bar{c}_j^*} \tag{9″}$$

For all p satisfying (9″) the inequality (6″) will be true. The index of the first variable for which the sign of its reduced cost coefficient will change from positive to negative when the parameter is increased may be denoted by k. The value of p for which this occurs may be denoted by p_k and obtained from

$$p_k = \frac{-\bar{c}_k}{\bar{c}_k^*} = \underset{j}{\text{Min}} \left\{ \frac{-\bar{c}_j}{\bar{c}_j^*} \,\middle|\, \bar{c}_j < 0 \text{ and } \bar{c}_j^* < 0 \right\} \tag{10″}$$

This definition of p_k will in this case either give a negative value or no value at all.

(*c*) *Optimisation*

The above two rules for determining the variable entering the basis may be summarised using the parameter *opt* assumed equal to −1 in the case of minimisation and +1 in the case of maximisation, and defining Extremum (−1) and Extremum (+1) to mean Min and Max, respectively.

$$p_k = \frac{-\bar{c}_k}{\bar{c}_k^*} = \text{Extremum}\,(-opt) \left\{ \frac{-\bar{c}_j}{\bar{c}_j^*} \,\middle|\, opt \times \bar{c}_j < 0 \text{ and } \bar{c}_j^* < 0 \right\} \tag{10}$$

For $p < p_k (p > p_k)$ at minimisation (maximisation) the reduced cost coefficient of the variable x_k will be positive (negative) and according to the rules of the primal simplex method it will be a candidate for introduction into the basis. Therefore if a p_k value exists as defined by (10), then a corresponding primal simplex iteration is undertaken, else a feasible dual solution has been found.

If the increase of x_k does not lead to that some basis variable begins to become negative, then there exists an infinite homogeneous solution to the primal problem and hence no feasible dual solution to the problem.

The determination of the row of the basic variable which has to leave the basis is analogous to the primal simplex method and given by

$$\frac{\bar{b}_1 + \bar{b}_1^* q^*}{\bar{a}_{lk}} = \underset{i}{\text{Min}} \left\{ \frac{\bar{b}_i + \bar{b}_i^* q^*}{\bar{a}_{ik}} \,\middle|\, \bar{a}_{ik} > 0 \right\} \tag{11}$$

Dual simplex iterations

Similarly, it may be assumed that at the current iteration $q = q^* > 0$ and that all the reduced constants are non-negative.

$$\bar{b}_i + \bar{b}_i^* q^* \geqslant 0 \qquad (i = 1, \ldots, m) \tag{12}$$

Decreasing the parameter of $q \geqslant 0$ similarly affects the sign of (10) if

$$\bar{b}_i < 0 \text{ and } \bar{b}_i^* > 0 \tag{13}$$

as evident from the table below.

sign $(\bar{b}_i + \bar{b}_i^* q^*)$ for $0 \leqslant q < q^*$

	$\bar{b}_i^* > 0$	$\bar{b}_i^* = 0$	$\bar{b}_i^* < 0$
$\bar{b}_i > 0$	+	+	+
$\bar{b}_i = 0$	+	0	(−)
$\bar{b}_i < 0$	+ → −	(−)	(−)

(14)

The expressions (12) for which (13) holds may be expressed as

$$q \geqslant q_i = \frac{-\bar{b}_i}{\bar{b}_i^*}$$

The critical value of q and the corresponding row l may be obtained from

$$q_l = \frac{-\bar{b}_l}{\bar{b}_l^*} = \underset{i}{\text{Max}} \left\{ \frac{-\bar{b}_i}{\bar{b}_i^*} \,\middle|\, \bar{b}_i < 0 \text{ and } \bar{b}_i^* > 0 \right\} \tag{15}$$

This definition of q_l will either give a positive value or no value at all.

Again for $q < q_l$ the value of the basic variable in the lth row will become negative and hence according to the rules of the dual simplex algorithm a candidate for leaving the basis. Therefore, if a q_l value exists as defined by (15) then a corresponding dual simplex iteration is undertaken, else a feasible primal solution has been found.

In the case that a dual simplex iteration is undertaken, then any entering variable has to be chosen so that its introduction decreases the infeasibility of the primal solution corresponding to $q < q^*$. This can only occur if its transformed coefficient $\bar{a}_{lj} < 0$.

If there is no such variable then it is impossible to decrease the infeasibility of the corresponding basic variable, in which case no feasible primal solution exists to the problem, and hence there exists an infinite homogeneous solution to the dual problem. (See Kronsjö, 1968, section 1.)

If there are several variables with $\bar{a}_{lj} < 0$, then the one is selected which would least unfavourably affect the optimisation, as follows.

All nonbasic reduced cost coefficients are in the case of minimisation (maximisation) nonpositive (nonnegative) and the variable x_k associated with the smallest increase (decrease) in the objective function per unit of decreasing the infeasibility is selected, i.e.

$$\frac{\bar{c}_k + \bar{c}_k^* p^*}{-\bar{a}_{lk}} = \underset{j}{\text{Extremum}}\,(-opt) \left\{ \frac{\bar{c}_j + \bar{c}_j^* p^*}{-\bar{a}_{lj}} \,\middle|\, \bar{a}_{lj} < 0 \right\} \tag{16}$$

The four possible types of solution of a linear programming problem

In order to establish to which one of the four possible cases any given linear program belongs, it is near to hand to start off using the working hypothesis that a feasible primal and a feasible dual solution may exist, i.e. by assuming that *primal solution* := *dual solution* := 1. The values of the parameters p and q may then systematically be forced to zero. The iterations are only continued if a primal solution is not ruled out and primal feasibility improvement possible $q \neq 0$, or a dual solution is not ruled out and further dual feasibility improvement possible $p \neq 0$. If the absolute value of p is greater than or equal to q or no further improvement is

possible of q (which will be the case if it has been established that no feasible primal solution exists) then we shall try to decrease the absolute value of $p(-opt \times p)$ else an attempt should be made to decrease q. If $p(q) \neq 0$ and no improvement of $p(q)$ is possible, it is then recorded that the problem has no feasible dual (primal) solution, by setting the variable *dual (primal) solution* equal to zero.

Upon exit the nonexistence or existence of a primal (dual) solution to the unparameterised problem may be indicated by a variable $k(l)$ assuming the values 0 and >0, respectively.

The solution of the unparameterised problem must belong to one of the following four categories:

(i) A primal feasible ($k > 0$) and a dual feasible ($l > 0$) solution:

$$x_{I[i]} = U[i, -2] \qquad (i = 0, \ldots, m)$$
$$u_j = U[0, j] \qquad (j = 1, \ldots, m)$$

(ii) An infinite primal solution ($k > 0$) and no dual feasible ($l = 0$) solution:

$$x_{I[i]} = U[i, -2] - U[i, -4] \times \theta \qquad (i = 0, \ldots, m)$$
$$\lim_{\theta \to \infty} x_k = \theta$$

(iii) No primal feasible ($k = 0$) and an infinite dual ($l > 0$) solution:

$$u_j = U[0, j] + U[l, j] \times \theta \qquad (j = 1, \ldots, m)$$
$$\lim_{\theta \to \infty} u_l = \theta$$

(iv) No primal feasible ($k = 0$) and no dual feasible ($l = 0$) solution.

Examples for the systematic testing of the computer program

The computer program should be able to deal with problems having:

1. No feasible primal and no feasible dual solution;
2. Feasible primal and no feasible dual solution, i.e. infinite primal solution;
3. No feasible primal and feasible dual solution, i.e. infinite dual solution;
4. Feasible primal and feasible dual solution;

and involving:

(i) Minimisation;
(ii) Maximisation.

The following five test examples were therefore prepared. In solving them, the equations were at each stage completely transformed into canonical form, as this facilitated the checking of the solution at each stage. The computer procedure will only calculate the contents of I, U and the elements of the -1st, 0th (and lth) rows in order to determine the index k of the entering variable in performing a primal (dual) simplex iteration.

A column of checksums indicated by Σ is formed by adding the coefficients and the constant of the original equations. The same operations are performed upon this column as upon the other columns of the equations. It follows that in any iteration the sum of an equation's transformed coefficients and constants must equal the transformed checksum, which is used to unravel the practically unavoidable mistakes performed in the manual calculations. Assuming that any discovered mistake is immediately corrected, the transformations may be continued without greater cause for worry that any earlier mistakes will make nonsense out of the subsequent calculations.

An entering variable is indicated by ↑ and a leaving one by ↓.

In comparing the ultimate contents of the arrays produced by the computer with the tables above, it is natural to find that the computer array $U[-1{:}m, -4]$ of the entering variable is part of an ultimate table in those cases in which the problem has an infinite homogeneous primal solution and that it is part of a penultimate table in all those other cases in which no variable is a candidate for entry in the ultimate table.

The solution of the test problems

$I[-1{:}m]$ $U[-1{:}m, -1{:}m]$ $U[-1{:}m, -4]$ $U[-1{:}m, -2{:}-3]$ enclosed by boxes.

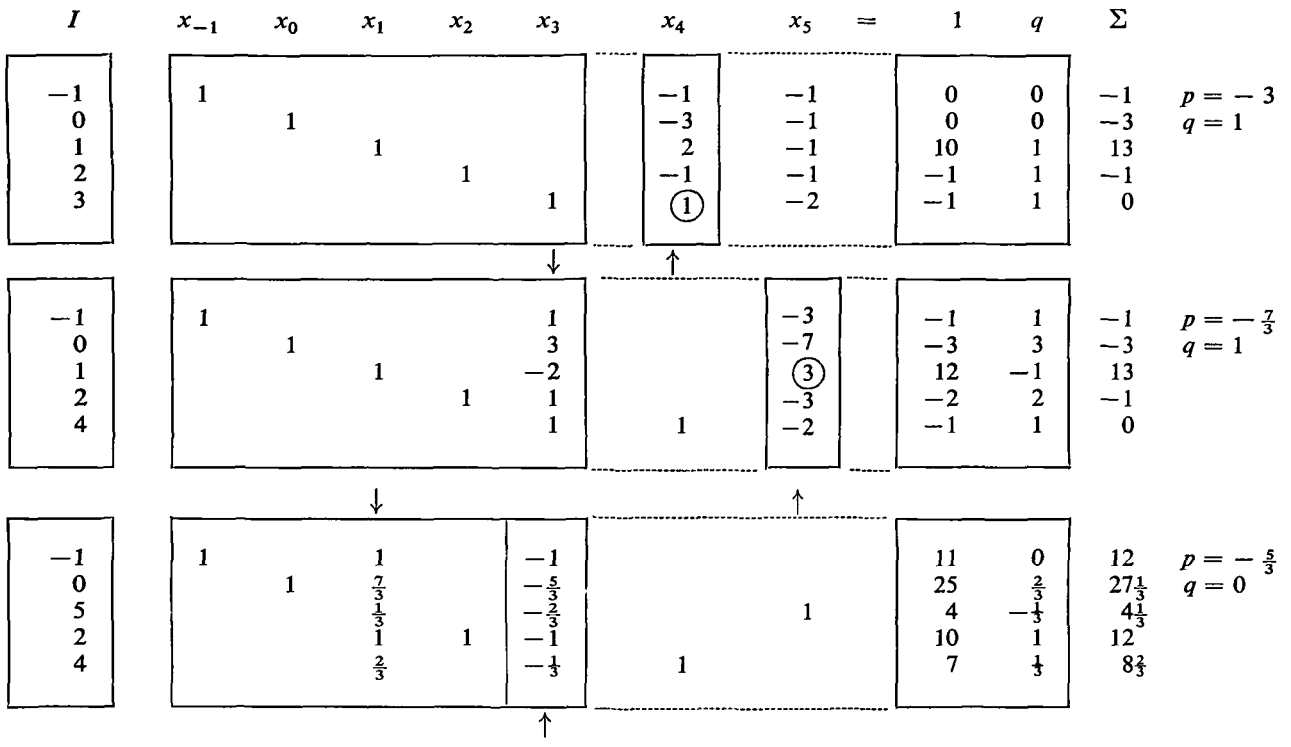
No primal and no dual solution, Min x_0

I	x_{-1}	x_0	x_1	x_2	x_3	x_4	x_5	=	1	q	Σ	
−1	1					−1	−1		0	0	−1	$p = 4$
0		1				−3	4		0	0	2	$q = 2$
1			1			2	−2		10	1	12	
2				1		−1	(1)		−1	1	1	
3					1	1	−1		−2	1	0	
				↓			↑					
−1	1			1		−2	0		−1	1	0	$p = \frac{1}{2}$
0		1		−4		1	0		4	−4	−2	$q = \frac{3}{2}$
1			1	2		0	0		8	3	14	
5				1		−1	1		−1	1	1	
3				1	1	0	0		−3	2	1	
						↑	↓					

$q = \frac{3}{2}$ $l = 3$ no variable with a negative coefficient in that constraint, hence *no feasible primal solution (infinite homogeneous solution).*

$p = \frac{1}{2}$ $k = 4$ no constraint in which that variable has a positive coefficient, hence *infinite homogeneous primal solution (no feasible dual solution).*

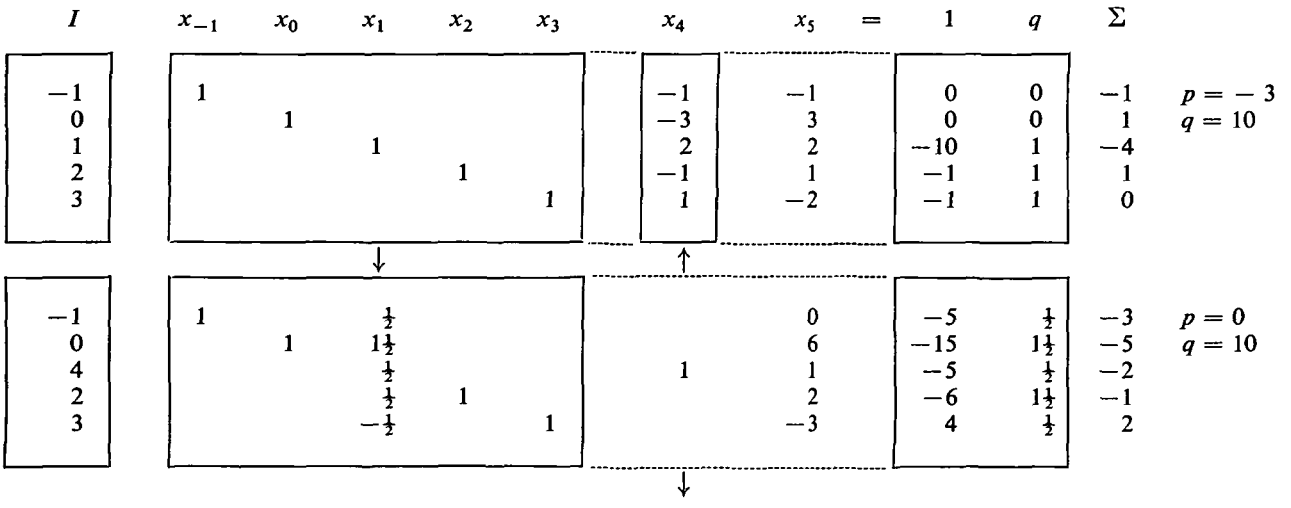
Infinite primal solution, Max x_0



I	x_{-1}	x_0	x_1	x_2	x_3	x_4	x_5	$=$	1	q	Σ	
−1	1					−1	−1		0	0	−1	$p = -3$
0		1				−3	−1		0	0	−3	$q = 1$
1			1			2	−1		10	1	13	
2				1		−1	−1		−1	1	−1	
3					1	(1)	−2		−1	1	0	
					↓	↑						
−1	1				1		−3		−1	1	−1	$p = -\frac{7}{3}$
0		1			3		−7		−3	3	−3	$q = 1$
1			1		−2		(3)		12	−1	13	
2				1	1		−3		−2	2	−1	
4					1	1	−2		−1	1	0	
			↓				↑					
−1	1		1		−1				11	0	12	$p = -\frac{5}{3}$
0		1	$\frac{7}{3}$		$-\frac{5}{3}$				25	$\frac{2}{3}$	$27\frac{1}{3}$	$q = 0$
5			$\frac{1}{3}$		$-\frac{2}{3}$		1		4	$-\frac{1}{3}$	$4\frac{1}{3}$	
2			1	1	−1				10	1	12	
4			$\frac{2}{3}$		$-\frac{1}{3}$	1			7	$\frac{1}{3}$	$8\frac{2}{3}$	
					↑							

$q = 0$ hence *feasible primal solution*
$p = -\frac{5}{3}$ $k = 3$ no constraint in which the variable has a positive coefficient, hence *infinite homogeneous primal solution* (*no feasible dual solution*)

Infinite dual solution, Max x_0



I	x_{-1}	x_0	x_1	x_2	x_3	x_4	x_5	$=$	1	q	Σ	
−1	1					−1	−1		0	0	−1	$p = -3$
0		1				−3	3		0	0	1	$q = 10$
1			1			2	2		−10	1	−4	
2				1		−1	1		−1	1	1	
3					1	1	−2		−1	1	0	
			↓			↑						
−1	1		$\frac{1}{2}$				0		−5	$\frac{1}{2}$	−3	$p = 0$
0		1	$1\frac{1}{2}$				6		−15	$1\frac{1}{2}$	−5	$q = 10$
4			$\frac{1}{2}$			1	1		−5	$\frac{1}{2}$	−2	
2			$\frac{1}{2}$	1			2		−6	$1\frac{1}{2}$	−1	
3			$-\frac{1}{2}$		1		−3		4	$\frac{1}{2}$	2	
							↓					

$q = 10$ $l = 1$ no variable with a negative coefficient in that constraint, hence *no feasible primal solution* (*infinite homogeneous dual solution*)
$p = 0$ hence *a feasible dual solution*

A feasible primal and dual solution, Max x_0

I	x_{-1}	x_0	x_1	x_2	x_3	x_4	x_5	=	1	q	Σ	
−1	1					−1	−1		0	0	−1	$p = -3$
0		1				−3	3		0	0	1	$q = 1$
1			1			2	2		10	1	16	
2				1		−1	1		−1	1	1	
3					1	(1)	−2		−1	1	0	
					↓	↑						
−1	1				1		−3		−1	1	−1	$p = -1$
0		1			3		−3		−3	3	1	$q = 1$
1			1		−2		(6)		12	−1	16	
2				1	1		−1		−2	2	1	
4					1	1	−2		−1	1	0	
			↓				↑					
−1	1		$\frac{1}{2}$		0				5	$\frac{1}{2}$	7	$p = -\frac{1}{2}$
0		1	$\frac{1}{2}$		2				3	$2\frac{1}{2}$	9	$q = 1$
5			$\frac{1}{6}$		$-\frac{1}{3}$		1		2	$-\frac{1}{6}$	$2\frac{2}{3}$	
2			$\frac{1}{6}$	1	$\frac{2}{3}$				0	$1\frac{5}{6}$	$3\frac{2}{3}$	
4			$\frac{1}{3}$		$\frac{1}{3}$	1			3	$\frac{2}{3}$	$5\frac{1}{3}$	

$q = 0$ hence *a feasible primal solution*
$p = 0$ hence *a feasible dual solution*

A feasible primal and dual solution, Min x_0

I	x_{-1}	x_0	x_1	x_2	x_3	x_4	x_5	=	1	q	Σ	
−1	1					−1	−1		0	0	−1	$p = 3$
0		1				−3	3		0	0	1	$q = 1$
1			1			2	2		10	1	16	
2				1		−1	(1)		−1	1	1	
3					1	1	−2		−1	1	0	
				↓			↑					
−1	1			1		−2			−1	1	0	$p = 0$
0		1		−3		0			3	−3	−2	$q = 1$
1			1	−2		4			12	−1	14	
5				1		−1	1		−1	1	1	
3				2	1	−1			−3	3	2	
						↑	↓					
−1	1			−1			−2		1	−1	−2	$p = 0$
0		1		−3			0		3	−3	−2	$q = 1$
1			1	2			4		8	3	18	
4				−1		1	−1		1	−1	−1	
3				1	1		−(1)		−2	2	1	
				↓			↑					
−1	1			−3	−2				5	−5	−4	$p = 0$
0		1		−3	0				3	−3	−2	$q = 0$
1			1	6	4				0	11	22	
4				−2	−1	1			3	−3	−2	
5				−1	−1		1		2	−2	−1	

$q = 0$ hence *a feasible primal solution*
$p = 0$ hence *a feasible dual solution*

A numerical example prepared by A. C. McKay (1968) was selected to provide a test problem involving moreiterations.

The following criteria for zero have been proposed by P. Wolfe (1965)

$$e_0 = 10^{-7},\ e_1 = 10^{-3},\ e_2 = 10^{-5}\ (e_3 =)\ e_4 = 10^{-4}.$$

The procedure was tested on the ICL-KDF9 Computer and some shortcomings eliminated by V. Kolarov, Deputy Chief of the Mathematical Simulation of Economic Processes Department of the State Committee of Planning, Sofia, Bulgaria.

References

DANTZIG, G. B. (1963). *Linear Programming and Extensions*, Princeton University Press.

KRONSJÖ, T. O. M. (1970). Primal Simplex LP using Multipliers, Algorithm 59, *The Computer Journal*, Vol. 13, No. 4, pp. 428–429.

KRONSJÖ, T. O. M. (1968). Centralization and Decentralization of Decision Making, the Decomposition of any Linear Programme in Primal and Dual Directions—to obtain a Primal and a Dual Master Solved in Parallel and one or more Common Subproblems, *Revue Française d' Informatique et de Recherche Opérationelle*, 2e année, No. 10, pp. 73–104.

MCKAY, A. C. (1968). A Numerical Example with One Common Subproblem and Finite Optimal Solution, *Revue Française d'Informatique et de Recherche Opérationelle*, 2e année, No. 10, pp. 105–113.

WOLFE, P. (1965). Error in the Solution of Linear Programming Problems, in Louis B. Rall (Ed.), *Error in Digital Computation*, Vol. 2, United States Army Mathematics Research Center, The University of Wisconsin, Publication No. 15, John Wiley and Sons, Inc., New York, pp. 271–284.


```
comment The procedure FIND is redefined to be a real procedure FIND OR ASSIGN, the identifier of which is assigned the minimal value of the function evaluated, and if no index is found with the required properties then a formal real variable p is assigned the value of another formal real variable q before the jump to the formal label;
real procedure FIND OR ASSIGN (r, opt, f, i, s, t, b, p, q, e);
value opt, s, t, q; integer r, opt, i, s, t; real f, p, q; Boolean b;
label E;
  begin real extremum;
  extremum := - 10↑20 × opt;
  for i := s step 1 until t do
  if b then
    begin
    if opt × (f = extremum) > 0 then
      begin
      extremum := f;
      r := i
      end
    end;
  if extremum = - 10↑20 × opt then
    begin
    p := q;
    goto E
    end
  else FIND OR ASSIGN := extremum
  end procedure FIND OR ASSIGN;

comment A real procedure SIGMA is introduced which is identical with SUM except that the result of the summation is not assigned to a result variable r;
real procedure SIGMA (i, s, t, f);
value s, t; real f; integer i, s, t;
  begin real w;
  w := 0;
    for i := s step 1 until t do w := w + f;
    SIGMA := w
    end real procedure SIGMA;

procedure SELF DUAL PARAMETRIC SIMPLEX LP USING MULTIPLIERS (opt, m, n, I, A, U, k, l, e0, e1, e2, e4, FIND OR ASSIGN, SIGNIFICANT, SUM, SIGMA);
value opt, m, n, e0, e1, e2, e4; integer opt, m, n, k, l;
integer array I; array A, U; real e0, e1, e2, e4;
real procedure FIND OR ASSIGN, SIGNIFICANT, SUM, SIGMA;
  begin integer i, j; real primal solution, dual solution, p, q, v, w;
  comment primal (dual) solution equals if no primal (dual) feasible solution to the unparameterised problem then 0 else 1
  v, w working variables;
  primal solution := dual solution := q := 1; p := - opt;
  comment The determination of the minimal value of the parameters p and q consistent with that the initial basic solution would represent a feasible and optimal solution of the parameterised problem;
  p := FIND OR ASSIGN (k, -opt, -v/w, j, 1, n, opt × SUM (v, i 0, m, U[0, i] × A[i, j]) < -e1 ∧ SUM (w, i, −1, m, U[−1, i] × A[i, j]) < -e1, p, 0, NEXT ITERATION);
  q := FIND OR ASSIGN (l, 1, −U[i, −2] / U[i, −3], i, 1, m, U[i, −2] < -e4 ∧ U[i, −3] > e4, q, 0, NEXT ITERATION);
NEXT ITERATION: if primal solution = 1 ∧ q > e4 ∨ dual solution = 1 ∧ -opt × p > e1 then
    begin
    if primal solution = 0 ∨ dual solution = 1 ∧ -opt × p ⩾ q then
      begin
      comment The variable x_k is found which will become a candidate for introduction into the basis when the parameter p is decreased, if no candidate exists then a dual feasible solution has been found to the unparameterised problem;
      p := FIND OR ASSIGN (k, -opt, -v/w, j, 1, n, opt × SUM(v, i, 0, m, U[0, i] × A[i, j]) < -e1 ∧ SUM (w, i, −1, m, U[−1, i] × A[i, j] < -e1, p, 0, NEXT ITERATION);
      comment The leaving variable is determined according to the rules of the primal simplex algorithm, if no basic variable is decreased in value at the increase of the entering variable, then there exists an infinite homogenous solution to the primal problem and hence no feasible dual solution to the unparameterised problem;
      FIND OR ASSIGN (l, −1,(U[i, −2] + q × U[i, −3] / U[i, −4], i, −1, m, SUM(U[i, −4], j, −1, m, U[i, j] × A[j, k]) > e2 ∧ i > 0, dual solution, 0, NEXT ITERATION)
      end
    else
      begin
      comment The basic variable in the lth equation is found which will be a candidate for leaving the basis when the parameter q is decreased, if no candidate exists then a primal feasible solution has been found to the unparameterised problem;
      q := FIND OR ASSIGN (l, 1, −U[i, −2] / U[i, −3], i, 1, m, U[i, −2] < − e4 ∧ U[i, −3] > e4, q, 0, NEXT ITERATION);
      comment The entering variable x_k is determined according to the rules of the dual simplex algorithm, if there exists no nonbasic variable the increase of which may decrease the infeasibility of the infeasible basic variable in the lth equation then there exists no feasible primal solution to the unparameterised problem;
```

```
    FIND OR ASSIGN (k, -opt, (SIGMA(i, 0, m, U[0, i]
    × A[i, j]) + p × SIGMA(i, −1, m, U[−1, i]
    × A[i, j])) / (−w), j, 1, n, SUM(w, i, −1, m, U[l, i]
    × A[i, j]) < -e2, primal solution, 0, NEXT
    ITERATION);
    comment The determination of the transformed form of
    the entering variable x_k;
    for i := − 1 step 1 until m do
    U[i, −4] := SIGNIFICANT(SIGMA(j, −1, m,
    U[i, j] × A[j, k]), e0)
    end;
  comment The change of the basis, transforming the con-
  stants and updating the inverse;
  I[l] := k;
  for j := −3 step 1 until m do
  U[l, j] := U[l, j] / U[l, −4];
  for i := − 1 step 1 until l − 1, l + 1 step 1 until m do
  for j := − 3 step 1 until m do
  U[i, j] := SIGNIFICANT(U[ i, j] −U[l, j] × U[i, −4],
  e0);
  goto NEXT ITERATION
  end;
if primal solution = 0 then k := 0;
if dual solution = 0 then l := 0
end procedure SELF DUAL PARAMETRIC SIMPLEX
LP USING MULTIPLIERS;
```

Algorithm 62

INTERPOLATING QUINTIC SPLINES ON EQUI-DISTANT KNOTS

W. D. Hoskins
Brunel University

Author's note:

For a full quintic spline $f(x)$ interpolating to the points (x_i, y_i) (where $x_i = x_0 + ih$, $i = 0(1)n$) and the boundary values y_0', y_0'', y_n', y_n'' it can be demonstrated that if the continuity of third and fourth derivatives is required then the following two relationships between the first and second derivative values of the spline apply,

$$\delta^2 y_j - \frac{2h}{5}(f'_{j+1} - f'_{j-1}) + \frac{h^2}{20}\delta^2 f_j'' = \frac{h^2}{5} f_j'' \qquad (1)$$

$$y_{j+1} - y_{j-1} - \frac{7h}{15}\delta^2 f_j' + \frac{h^2}{15}(f''_{j+1} - f''_{j-1}) = 2hf_j' \qquad (2)$$
$$j = 1(1)n - 1$$

equation (1) being first suggested in the literature by Späth (1969).

The above equations can be linearly combined to yield a further equation,

$$y_{i+2} + 10y_{i+1} - 10y_{i-1} - y_{i-2} = \frac{h}{5}(f'_{i+2} + 26f'_{i+1} + 66f'_i + 26f'_{i-1} + f'_{i-2}) \qquad (3)$$
$$i = 2(1)n - 2$$

called the consistency condition by Loscalzo (1969).

However, corresponding to $i = 1$, $n - 1$ two further equations are required and may be obtained from equations (1) and (2) by eliminating the second derivatives of $f(x)$ to obtain

$$16h^2 f_0'' = -235y_0 + 65y_1 + 155y_2 + 15y_3 - 111hf_0' - 227hf_1' - 79hf_2' - 3hf_3' \qquad (4)$$

and

$$16h^2 f_n'' = -235y_n + 65y_{n-1} + 155y_{n-2} + 15y_{n-3} + 111hf_n' + 227hf_{n-1}' + 79hf_{n-2}' + 3hf_{n-3}' \qquad (5)$$

Equations (4), (3) and (5) now constitute a matrix equation of order $n - 1$ and band width 5 (not symmetric, but diagonally dominant) for the $n - 1$ unknowns f_j'. This matrix equation can be solved directly by a stable LU decomposition (Wilkinson, 1965).

A constrasting analysis is that given by Ahlberg, Nilson and Walsh (1967) which uses a quintic defined in terms of fourth derivatives to establish a matrix equation of similar structure, but of slightly larger order and calculation of the first derivatives then has to be done with the equivalent of an integration formula.

Quintic spline interpolation is obtained on using the equations essentially given by Späth, viz.

$$f(x) = A_k z^5 + B_k z^4 + C_k z^3 + D_k z^2 + E_k z + F_k$$
$$k = j - 1 = 0(1)n - 1$$

where $z = \dfrac{(x - x_{j-1})}{h}$

$$F_k = y_{j-1}$$
$$E_k = hf'_{j-1}$$
$$2D_k = h^2 f''_{j-1}$$
$$C_k = 10(y_j - y_{j-1}) - h(6f'_{j-1} + 4f'_j) + \frac{h^2}{2}(f''_j - 3f''_{j-1})$$
$$B_k = -15(y_j - y_{j-1}) + h(7f'_j + 8f'_{j-1}) + \frac{h^2}{2}(3f''_{j-1} - 2f''_j)$$
$$A_k = 6(y_j - y_{j-1}) - 3h(f'_j + f'_{j-1}) + \frac{h^2}{2}(f''_j - f''_{j-1})$$

and $x_k \leqslant x \leqslant x_{k+1}$, in conjunction with equations (4) and (5) (with the appropriate subscripts changed) to compute the quantities f_j''.

References

AHLBERG, J. H., NILSON, E. N., and WALSH, J. L. (1967). *The theory of splines and their applications*, Academic Press.

LOSCALZO, F. R. (1969) (Ed. Greville, T. N. E.). *Theory and application of spline functions*, Academic Press.

SPÄTH, H. (1969). Interpolation by certain quintic splines, *The Computer Journal*, Vol. 12, p. 292.

WILKINSON, J. H. (1965). *The Algebraic eigenvalue problem*, Oxford University Press.

```
procedure Quintic(y, y1, y2, d, c, aa, n, h);
value n, h; integer n; real h; array y, y1, y2, d, c, aa;
comment This procedure calculates from the n + 1 values y[i],
the first derivative conditions y1[0], y1[n] and the second deriva-
tive conditions y2[0], y2[n] the unknowns y1[i] (i = 1(1)n − 1)
needed to perform full quintic spline interpolation. The deriva-
tives y2[i] and coefficients, C[k], B[k], A[k] (k = 0(1)n − 1)
are further computed and stored respectively in the arrays
y2, d, c, aa;
  begin integer i, j, k; real n2, w, z;
  for i := 2 step 1 until n − 2 do
    begin
    y2[i − 1] := 1; aa[i − 1] := c[i] := 26;
    y1[i] := 66;
    d[i] := 5 × y[i + 2] + 50 × y[i + 1] − 50 ×
    y[i − 1] − 5 × y[i− 2]
    end;
  y2[n − 3] := 3; c[1] := aa[n − 2] := 79;
  y1[1] := y1[n − 1] := 227;
  d[1] := −235 × y[0] − 111 × h × y1[0] − 16 × h ×
  h × y2[0] + 65 × y[1] + 155 × y[2] + 15 × y[3];
  d[2] := d[2] − h × y1[0]; d[n − 2] := d[n − 2] − h ×
  y1[n];
  d[n − 1] := 235 × y[n] − 111 × h × y1[n] + 16 × h ×
  h × y2[n] − 65 × y[n − 1] − 155 × y[n − 2] − 15 ×
  y[n − 3];
```

```
for i := n − 1 step −1 until 3 do
  begin
  j := i − 1; k := i − 2;
  z := c[j]/y1[i]; y1[j] := y1[j] − aa[j] × z;
  d[j] := d[j] − z × d[i]; aa[k] := aa[k] − y2[k] × z;
  z := (if k > 1 then 1 else 3)/y1[i];
  c[k] := c[k] − aa [j] × z;
  y1[k] := y1[k] − y2[k] × z; d[k] := d[k] − z × d[i]
  end;
y1[1] := y1[1] − (c[1]/y1[2]) × aa[1];
d[1] := d[1] − (c[1]/y1[2]) × d[2];
z := 1/h; y1[1] := z × d[1]/y1[1];
y1[2] := (−aa[1] × y1[1] + z × d[2])/y1[2];
n2 := z × z × ·0625; w := y2[1];
d[1] := y2[2];
for i := 3 step 1 until n − 1 do
  begin
  y1[i] := (z × d[i] − aa[i − 1] × y1[i − 1] − w ×
  y1[i − 2])/y1[i];
  w := d[1]; d[1] := y2[i];
  y2[i] := n2 × (−235 × y[i] + 65 × y[i − 1] + 155 ×
  y[ i − 2]+15 × y[i − 3] + h × (111 × y1[i] + 227 ×
  y1[i − 1] + 79 × y1[i − 2] + 3 × y1[i − 3]))
  end;
for i := 1 step 1 until 2 do
y2[i] := n2 × (−235 × y[i] + 65 × y[i + 1] + 155 ×
y[i + 2] + 15 × y[i + 3] −h × (111 × y1[i] + 227 ×
y1[i + 1] + 79 × y1[i + 2] + 3 × y1[i+ 3]));
z := h × h × ·5;
for i := 1 step 1 until n do
  begin
  k := i − 1; w := y[i] − y[k];
  d[k] := 10 × w − h × (6 × y1[k] + 4 × y1[i])
  + z × (y2[i] − 3 × y2[k]);
  c[k] := − 15 × w + h × (7 × y1[i] + 8 × y1[k])
  + 2 × z × (1·5 × y2[k] − y2[i]);
  aa[k] := 6 × w − 3 × h × (y1[i] + y1[k])
  + z × (y2[i] − y2[k])
  end
end;
```

Contributions for the Algorithms Supplement should be sent to
Mrs. M. O. Mutch
University Engineering Department
Control Engineering Group
Mill Lane, Cambridge

Book review continued

equations are exhibited, for the deterministic as well as for the stochastic case. The relationship with Markov processes is also illustrated. The last chapter deals, very briefly, with the use of computers, storage requirements, and computing times. Nine references are given.

Vol. 16. This volume contains short descriptions of non-linear programming methods, due respectively to J. E. Kelley, H. O. Hartley and R. R. Hocking, K. Kleibohm, A. F. Veinott, jr., G. Zoutendijk, R. E. Griffith and R. A. Stewart, P. Huard. There follows a description of the reduced gradient method, of methods with penalty functions and unconstrained minimization techniques, and finally of a method of feasible directions, apparently due to S. I. Zukhovitsky, R. A. Poljak and M. E. Primak.

The collection does not claim to be exhaustive, and the most valuable part of the book is a bibliography of more than 100 pages, referring also to control theory, economics, operational research and games theory, while dynamic, stochastic and integer programming are not considered.

Vol. 17. The authors say that it is their aim to set side by side the two principal possibilities of solving control problems —the method of dynamic programming, and the procedure based on the maximum principle of Pontrjagin. They do this in Chapters II and III, but the relationship between the two methods is not explored, and the preface states explicitly that occasional cross-references are not essential to the understanding. In fact, the first-named author was mainly responsible for Chapter III, and the other author for the first two chapters.

Chapter I starts with a few examples and leads up to the general formulation of a control problem:

Minimize

$$F = \int_{t_0}^{t_1} f_0(x(t), y(t), t)dt$$

subject to

$$\dot{x} = f(x(t), y(t), t),\ x(t_0) = x_0,\ x(t_1) \in Z(t_1) \subset R^m.$$

where the unknown control vector $y(t)$ is assumed to be Lebesgue-integrable and $x(t)$ might be subject to further restrictions in the range $t_0 \leqslant t \leqslant t_1$.

Conditions are given which ensure the uniqueness of the solution of the differential equations, and certain continuity properties of F and x are deduced, dependent on y. The existence and uniqueness of an optimal control vector is then proved.

Chapter II deals with the fundamental concepts of dynamic programming, such as the principle of optimality, and with formulations in terms of functional equations. The (misleading) remark is made that dynamic programming is only applicable to problems with known horizon. Existence and uniqueness of optimal solutions is discussed, the functional equations are solved approximately by two different methods of iteration, and questions of computation are mentioned.

In Chapter III a slightly simplified problem is attacked. The classical maximum principle and transversality conditions are then introduced and the theory of linear control (i. e. where f^0 and f are linear in x and in y) is formulated. Extremal control vectors (i. e. those where the final point of the trajectory is on the boundary of the set of attainable points) are studied. Finally, methods are mentioned for the solution of non-linear control problems.

An Appendix defines relevant concepts, such as convexity and Lebesgue integrability, and gives a brief glimpse of functional analysis in Banach space. The bibliography contains 26 items with German and English titles.

S. VAJDA (Birmingham)