Algorithms supplement

Algorithm 85

GENERATION OF CHISHOLM APPROXIMANTS

R. Hughes Jones, P. R. Graves-Morris and G. J. Makinson
University of Kent
Canterbury
Kent.

Authors' Notes

Chisholm (1973) has shown how rational approximants may be defined for functions expressed as a power series in two variables, and the ALGOL procedure presented here generates all the Chisholm approximants up to some given maximum order m. Common and Graves-Morris (1974) have proved some of the mathematical properties of such approximants.

Hughes Jones and Makinson (see ref.) have shown how to generate successively approximants of increasing order by the prong method, and Graves-Morris, Hughes Jones and Makinson (see ref.) have reported some computational experience in approximating a number of functions in this way.

Let $f(x, y)$ be defined by the formal power series

$$f(x, y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} c_{ij}x^iy^j \ . \tag{1}$$

Then the mth order rational approximant defined by

$$f_{m/m}(x, y) = \frac{\sum_{u=0}^{m}\sum_{v=0}^{m} a_{uv}x^uy^v}{\sum_{p=0}^{m}\sum_{q=0}^{m} b_{pq}x^py^q} = \frac{P(x, y)}{Q(x, y)} \ , \tag{2}$$

is found by first multiplying the difference between (1) and (2) by $Q(x, y)$ which gives formally

$$Q(x, y) \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} c_{ij}x^iy^j - P(x, y) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} d_{ij}x^iy^j \tag{3}$$

Linear equations for the parameters a_{uv} and b_{pq}, where $0 \leqslant u, v, p, q \leqslant m$, are obtained by requiring

$$d_{ij} = 0 \quad 0 \leqslant i + j \leqslant 2m \ , \tag{4}$$

$$d_{ij} + d_{ji} = 0 \quad 1 \leqslant i \leqslant m \text{ where } i + j = 2m + 1 \tag{5}$$

These equations together with a normalising equation for b_{00} are sufficient to determine the parameters uniquely provided the coefficient matrix is non-singular.

By taking the equations in a particular order the system can be partitioned into a form which enables the solution to be found directly block by block.

Assuming b_{00} is normalised to unity, then the equations which arise from matching terms x^{m+u}, $u = 1, 2, \ldots, m$, in Hughes Jones and Makinson, only involve the m parameters b_{i0}, $i = 1, 2, \ldots, m$, and so these parameters can be determined provided the coefficient matrix is non-singular. Similarly the m equations which arise from matching terms y^{m+v}, $v = 1, 2, \ldots, m$, in Hughes Jones and Makinson allow the m parameters b_{0i}, $i = 1, 2, \ldots, m$, to be determined. These are referred to in the program as the parameters involving index zero, of which altogether there are $2m + 1$.

The $m - 1$ equations which are obtained from matching terms $x^{m+u}y$, $u = 1, 2, \ldots, m - 1$, in Hughes Jones and Makinson may be grouped together with the $m - 1$ equations which are obtained by matching terms xy^{m+v}, $v = 1, 2, \ldots, m - 1$ in Hughes Jones and Makinson, and with the addition of the one symmetrised equation given by $i = 1$ in Reinsch and Wilkinson (1971), they form a linear system of $2m - 1$ equations relating the $2m - 1$ parameters b_{11}, b_{i1} and b_{1i}, $i = 2, 3, \ldots, m$, to the previously determined parameters. The new block is referred to in the program as the set of parameters involving index unity.

The process can be continued and by matching terms $x^{m+u}y^2$, $u = 1, 2, \ldots, m - 2$, and x^2y^{m+v}, $v = 1, 2, \ldots, m - 2$, $2m - 4$ equations are obtained, which, taken together with the symmetrised equation $i = 2$ in Reinsch and Wilkinson (1971), form a linear system for the $2m - 3$ parameters b_{22}, b_{2i} and b_{i2}, $i = 3, 4, \ldots, m$. These parameters are expressed only in terms of parameters which have already been evaluated and so the block of parameters involving index 2 can be computed.

Continuing in this way all the denominator parameters can be found using a block by block process. The largest block to be solved is $(2m - 1 \times 2m - 1)$. The b's are thus computed in blocks of increasing index order and within a block the ordering chosen is as follows: the b's of index i are ordered $b_{im}, b_{i,m-1}, \ldots, b_{ii}, \ldots, b_{mi}$. This is slightly different from that given in Hughes Jones and Makinson.

Once the b's have been determined, the a's are given explicitly by the equations which come from matching terms x^iy^j, $i = 0, 1, \ldots, m$, $j = 0, 1, \ldots, m$, in Hughes Jones and Makinson.

The full set of $(m + 1)^2$ linear equations for the b's, $P^{(m)}\mathbf{b} = \mathbf{r}$, has a lower triangular block structure which is simply extended when m is increased. Apart from one small additional alteration, the adding of a border partition along the left and top sides of $P^{(m)}$ gives $P^{(m+1)}$. The procedure builds up successive $P^{(j)}$ of order N, where $N = j(j + 2)$, in array $q1$ with the (N, N) element of $P^{(j)}$ in $q1[-1, 1]$ and the (1, 1) element in $q1[-N, N]$.

Since the coefficient matrix for the set of b's of $f_{m/m}$ involving index i is exactly the same as the coefficient matrix for the set of b's of $f_{m+1/m+1}$ involving index $i + 1$ where $i = 1, 2, \ldots, m$, its triangular factors, which are obtained in the solution process, need be found only once and stored for subsequent use. For each new approximant, however, the set of equations for the b's involving index zero and the set for those involving index unity, have to be solved completely each time.

The triangular factors are stored by overwriting the corresponding block diagonal matrix which is stored in the array $q1$. The triangularisation and solution calculation is performed using the algorithms 'unsymdet' and 'unsym acc solve' (Reinsch and Wilkinson (1971)).

The a's are given explicitly in terms of the b's by an equation

$$\mathbf{a} = W^{(m)}\mathbf{b}$$

where the matrix $W^{(m)}$ is upper triangular if the blocks of parameters are ordered in decreasing index order and if within each block the a's of index i are ordered $a_{i0}, a_{0i}, a_{i1}, a_{1i}, \ldots, a_{ii}$.

The matrices W all have the property that $W^{(k+1)}$ is simply $W^{(k)}$ extended by a border partition along the top and left-hand sides. The procedure stores the triangular matrix $W^{(m)}$ with $W^{(m)}(i, j)$ in $q1[-N + i - 1, N - j + 1]$, in the array space left by the block triangular coefficient matrix $P^{(m)}$. In the problem of the overlap which occurs in the case of the diagonal blocks, the elements of $W^{(m)}$ are not written since the triangular factors of the diagonal submatrices of $P^{(m)}$ are already stored in these locations. All the elements of the diagonal block locations of $W^{(m)}$ are however repeated in the correct configuration in the leading triangular section of $W^{(m)}$. Hence by overwriting the upper triangular section of the leading diagonal block of array $q1$ with the elements of $W^{(m)}$, all the information is available in the store. This can be done because the leading diagonal block of $P^{(m)}$ does not occur again as a diagonal submatrix in $P^{(m+1)}$.

The triangular factors of the leading diagonal block of $P^{(m)}$ are consequently not stored for later use. The leading diagonal block elements of $P^{(m)}$ however, are stored to form $P^{(m+1)}$. Thus it is necessary to restore the overwritten elements before the procedure returns to begin calculating the parameters of the next order approximant.

The procedure computes the 1/1 approximant directly and generates successive i/i approximants up to m/m. It prepares for the construction of the coefficient matrix for the denominator parameters of unit index of the $i + 1$th order approximant whilst it is setting up the coefficient matrices for the denominator parameters of zero index of the ith order approximants.

Testing was carried out on an ICL 4130 computer by comparing the results obtained by the procedure with those obtained from a FORTRAN program which found the parameters by direct solution of the full set of equations.

The approximant was evaluated over various grids in the xy plane. A FORTRAN program has also been written by John and Lutterodt (1973) which might be adapted for this problem.

Procedure specification

comment The requisite power series coefficients c_{ij} must be supplied in array $c[0{:}2m+1, 0{:}2m+1]$ where c_{ij} is in $c[i, j]$. The elements of c which are not needed to store power series coefficients, namely $i + j > 2m + 1$, and $c[0{:}2m+1]$ and $c[2m+1, 0]$ can be assigned arbitrarily. On exit the numerator parameters are in *num*$[0{:}m, 0{:}m]$ and the denominator parameters are in *den*$[0{:}m, 0{:}m]$. The coefficient of $x^i y^j$ is assigned to array location $[i, j]$. The procedure exits to label *sing* if a singular block coefficient matrix is encountered during the solution process, and exits to label *ill* if any system of equations is too *ill* conditioned to be solved accurately. *eps* is the smallest number for which $1 + eps > 1$ on the computer;

References

CHISHOLM, J. S. R. (1973). Rational approximants defined from double power series, *Math. Comp.* 27, 124.

COMMON, A. K., and GRAVES-MORRIS, P. R. (1974). Some properties of Chisholm Approximants, *J. Inst. Maths. Applics.* 13.

HUGHES JONES, R., and MAKINSON, G. J. The Generation of Chisholm rational polynomial approximants to power series in two variables, to be published in *J. Inst. Maths. Applics.*

GRAVES-MORRIS, P. R., HUGHES JONES, R., and MAKINSON, G. J. The calculation of some rational approximants in two variables, to be published in *J. Inst. Maths. Applics.*

REINSCH, C., and WILKINSON, J. H. (1971). Linear Algebra *Handbook for Automatic Computation*, Vol. 2, Springer-Verlag.

JOHN, G., and LUTTERODT, C. H. Private communication, 1973.

```
procedure chis2 (num, den, c, m, eps, sing, ill);
value m, eps; array num, den, c; integer m;
label sing, ill; real eps;
begin integer i, j, k, q, s, g, t, h, r, it, im, top, d2; real diag, d1;
  q := m × (m + 2) + 1; den[0, 0] := 1; diag := c[0, 0];
  num[0, 0] := diag;
begin integer tot, mm, spk, var, qp, u, run, sop, upp, und, dec, drop, inc,
  qt, cs, ds, itn;
  real w; real array q1[−q :−1, 1:q], int, store[1:q];
  integer array as, bs[1:q, 1:2], blas[1:2 × m + 1, 1:2];
  den[0, 1] := store[1] := −c[0, 2]/c[0, 1]; den[1, 0] := store[2]
  := −c[2, 0]/c[1, 0]; w := c[0, 1] + c[1, 0]; den[1, 1] := store
  [3] := −(c[2, 1] + c[1, 2] + (c[1, 1] + c[2, 0]) × store[1] −
  (c[1, 1] + c[0, 2]) × store[2])/w; num[1, 0] := diag × den[1, 0]
  + c[1, 0]; num[0, 1] := diag × den[0, 1] + c[0, 1]; num[1, 1]
  := diag × den[1, 1] + c[1, 1] + c[0, 1] × den[1, 0] + c[1, 0] ×
  den[0, 1];
  comment 1/1 approximant found;
  if m ≠ 1 then
  begin q1[−4, 2] := q1[−2, 4]: = 0; q1[−2, 2] := c[1, 0];
    q1[−2, 3] := q1[−3, 2] := c[2, 0]; q1[−4, 3] := q1[−3, 4]
    := c[0, 2]; q1[−4, 4] := c[0, 1]; q1[−3, 3] := c[0, 3] +
    c[3, 0]; q1[−4, 1] := c[1, 1] + c[2, 0]; q1[−2, 1] := c[1, 1]
    + c[0, 2]; q1[−3, 1] := c[2, 1] + c[1, 2]; as[1, 2] :=
    as[2, 1] := 0; as[1, 1] := as[3, 1] := as[2, 2] := as[3, 2]
    := 1; g := 3;
    for h := 2 step 1 until m do
    begin q := h × (h + 2) + 1; qp := q + 1; tot := h + 1;
      sop := 2 × h + 1; s := q − tot;
      for i := 1 step 1 until tot do
      begin cs := qp − i;
        for j := 1 step 1 until i do
        begin comment setting up matrices for coeffts with a zero
          index but anticipating the next order unit index matrix;
          ds := qp − j; q1[−ds, cs − h] := q1[h − cs, ds] :=
          q1[−cs, ds − h] := q1[h − ds, cs] := 0; if j ≠ m + 1
          then q1[−ds, cs] := q1[−cs, ds] := c[0, i + j − 1];
          if i ÷ j ≠ m + 1
          then q1[−i − s + h, q − 2 × h − j + i] :=
          q1[−q + 2 × h + j − i, s + i − h] := c[2 × i − j, 0]
        end
      end i;
      if h ≠ m then q1[−q + h, q − h] := c[sop, 0] + c[0, sop];
      store[tot] := 1;
      begin array aa, d[1:h, 1:h], z, sol, bb[1:h, 1:1], piv[1:h];
        comment solving for the coeffts with zero index;
        for k := 0, 1 do
        begin upp := k × tot; dec := qp − upp;
          for i := 1 step 1 until h do
          begin z[i, 1] := −q1[−q + h, dec − i];
            for j := 1 step 1 until h do
            aa[i, j] := d[i, j] := q1[−dec + j, dec − i]
          end;
          unsymdet(h, eps, aa, d1, d2, piv, sing);
          unsymaccsolve(h, 1, d, aa, piv, z, eps, sol, bb, itn, ill);
        for i := 1 step 1 until h do store[i + upp] := sol[i, 1]
        end
      end decl d;
      mm := h − 1; r := q − 2 × h; u := q;
      begin integer array se[1:q, 1:2];
        for k := 0 step 1 until h do
        begin dec := mm − k;
          comment increasing index ordering vector bs extended;
          for i := 0 step 1 until dec do
          begin s := u − i; t := r + i; bs[s, 2] := bs[t, 1] :=
            h − i;
            bs[s, 1] := se[s, 1] := se[t, 2] := bs[t, 2] := k;
            se[t, 1] := se[s, 2] := tot + i
          end i;
          s := u − h + k; bs[s, 1] := bs[s, 2] := se[s, 1] := k;
          se[s, 2] := 2 × h − k + 1; spk := 2 × (h − k) − 1;
          r := r − spk; u := u − spk − 2
        end k;
        t := 2 × h + 1; s := q − 2 × h;
        for k := 0 step 1 until mm do
        begin comment setting up new lhs partition;
          spk := 2 × (h − k) − 1;
          for j := 1 step 1 until spk do
          begin var := s − j;
            for i := 1 step 1 until t do
            begin run := qp − i; it := se[var, 1] − bs[run, 1];
              im := se[var, 2] − bs[run, 2];
              if it < 0∨ im < 0 then q1[−run, var] := 0
              else q1[−run, var]: = c[it, im];
              if j = h − k then
              begin it := se[var, 2] − bs[run, 1];
                im := se[var, 1] − bs[run, 2];
                if it ≥ 0 ∧ im ≥ 0 then
                q1[−run, var] := q1[−run, var] + c[it, im]
              end
            end i
          end j;
          s := s − spk
        end k
      end decl se;
      t := 2 × h − 1; r := q − 2 × h;
      begin array aa, d[1:t, 1:t], z, sol, bb[1:t, 1:1], piv[1:t];
        integer f;
        comment solving for coeffts with one index unity;
        for i := 1 step 1 until t do
        begin z[i, 1] := 0;
          for j := 1 step 1 until sop do
          z[i, 1] := −q1[j − qp, r − i] × store[j] + z[i, 1]
        end i;
        comment coefft matrix to array d. Its factors to array aa;
        for i := 1 step 1 until t do for j := 1 step 1 until t do
        aa[i, j] := d[i, j] := q1[j − r, r − i];
        unsymdet(t, eps, aa, d1, d2, piv, sing);
        unsymaccsolve(t, 1, d, aa, piv, z, eps, sol, bb, itn, ill);
        for i := 1 step 1 until t do
        begin store[2 × h + 1 + i] := sol[i, 1];
        int[r − i] := piv[i];
          for j := 1 step 1 until t do q1[j − r, r − i] := aa[i, j]
        end factors overwrite coeffts in diagonal blocks of q1;
    comment remaining sets of equations solved using stored
    triangular factors. coeffts found in increasing index order;
    t := q - 4 × h; tot := h - 2;
    for k := 1 step 1 until tot do
    begin f := 2 × (h - k) - 1; qt := q - t; u := h - k - 1;
      for i := 1 step 1 until u do
      begin r := f + 1 - i;
        d[r, u + 1] := d[u + 1, r] := c[u + i, 0];
        d[u + 1, i] := d[i, u + 1] := c[0, u + i];
        for j := 1 step 1 until i do
        begin inc := f + 1 - j; d[i, j] := d[j, i] :=
          c[0, i + j - 1];
          d[r, inc] := d[inc, r] := c[i + j - 1, 0];
          d[r, j] := d[j, r] := d[inc, i] := d[i, inc] := 0
        end j
      end i;
      d[u + 1, u + 1] := c[0, f] + c[f, 0];
      for i := 1 step 1 until f do
      begin z[i, 1] := 0; s := t - i + 1; piv[i] := int[s];
        for j := 1 step 1 until f do
        aa[i, j] := q1[j - t - 1, s];
        for j := 1 step 1 until qt do
        z[i, 1] := -q1[j - qp, s] × store[j] + z[i, 1]
      end;
      unsymaccsolve (f, 1, d, aa, piv, z, eps, sol, bb, itn, ill);
      for i := 1 step 1 until f do store[qt + i] := sol[i, 1];
      t := t - f
    end k;
    sol[1, 1] := 0;
    for i := q - 1 step -1 until 1 do
    sol[1, 1] := sol[1, 1] - q1[-i - 1, 1] × store[q - i];
    store[q] := sol[1, 1]/w;
    comment list of denominator coeffts output to array den;
    for j := 1 step 1 until q do
    begin t := q - j + 1;
      den[bs[t, 1], bs[t, 2]] := store[j]
    end
  end denominator coeffts for h/h approximant now found;
  t := 2 × h - 1;
  for i := 1 step 2 until t do
  begin blas[i, 1] := blas[i + 1, 2] := h;
    blas[i + 1, 1] := blas[i, 2] := (i - 1)/2
  end;
  t := t + 2; blas[t, 1] := blas[t, 2] := h; dec := g + t;
  for j := 1 step 1 until t do
  begin as[j + g, 1] := blas[j, 1]; as[j + g, 2] := blas[j, 2]
  end decreasing index ordering vector as is extended;
  for i := q - 1 step -1 until 1 do store[i + 1] :=
  den[as[i, 1], as[i, 2]];
  comment the denominator coeffts are now in the array store in
  decreasing index order but array store will subsequently be
  overwritten by the numerator coeffts;
  for i := g + 1 step 1 until dec do
  begin q1 [-i, i + 1] := 0; tot := i - 2;
    for j  := 1 step 1 until tot do
    begin it := as[i, 1] - as[j, 1]; im := as[i, 2] - as[j, 2];
      if it < 0 ∨ im < 0 then q1[-j - 1, i + 1] := 0;
      else q1[-j - 1, i + 1] := c[it, im]
    end
  end coeffts of new eqns written to next row partition of q1;
  q1[-dec, dec + 1] := c[1, 0]; g := dec; upp := q;
  und := q - t + 1; dec := t; drop := 0;
  for k := 1 step 1 until h do
  begin inc := q; tot := und - 1;
    for i := upp step -1 until und do
    begin s := i - 1; store[i] := diag × store[i] +
      c[as[s, 1], as[s, 2]];
      for j := 2 step 1 until tot do
      store[i] := store[i] + q1[-j, i] × store[j];
      for j := und step 1 until s do
      store[i] := store[i] + q1[-j - drop, inc] × store[j];
      inc := inc - 1
    end;
    drop := drop + dec; upp := upp - dec; dec := dec - 2;
    und := und - dec
  end numerator coeffts in array store in the order as in array as;
  for i := q - 1 step -1 until 1 do num[as[i, 1] + as[i, 2]] :=
  store[i + 1];
  if h ≠ m then
  begin tot := h + 1; s := q - tot; qp := q + 1;
    for i := 1 step 1 until tot do
    for j := 1 step 1 until i do
    begin comment resetting the first diagonal block but
      only the upper triangular section;
      q1[-qp + i + h, qp - j] := q1[-qp + j + h, qp - i]
      := 0;
      q1[i - qp, qp - j] := c[0, i + j - 1];
      q1[-q + 2 × h + j - i, s + i - h] := c[2 × i - j, 0]
    end;
    q1[-q + h, q - h] := c[t, 0] + c[0, t]
  end
  All h/h coeffts found. A test to determine whether continuation
  to h + 1/h + 1 is required can be inserted here
 end h
end
end decl q1
end chis2;
```

Algorithm 86

COMPLEX INTERVAL ARITHMETIC

J. Rokne and P. Lancaster
Department of Computer Science
University of Calgary
Calgary, Alberta
Canada T2N 1N4

Author's Note

For the algebra we will refer to the note by Rokne and Lancaster (1971). In these subroutines a complex interval A is an ordered pair of complex numbers $(A1, A2)$. If $A = (A1, A2)$ then $\text{Re}(A1) + i\,\text{Im}(A1)$ is the lower left hand corner of our complex interval and $\text{Re}(A2) + i\,\text{Im}(A2)$ is the upper right hand corner.

The algorithm presupposes the existence of four procedures, *FIADD*, *FISUB*, *FIMUL* and *FIDIV* that performs interval arithmetic operations on real numbers, and a procedure *FXTEND* that adds 1 to the right hand digit of the mantissa of a floating point number. Such procedures were published earlier in ALGOL 60 by Gibb (1961) under the names *RANGESUM*, *RANGESUB*, *RANGEMPY*, *RANGEDVD* and *CORRECTION* with the obvious translation. Let *XYZ* denote any one of *ADD*, *SUB*, *MUL* and *DIV* and let $X1$, $X2$, $Y1$, $Y2$, $Z1$, $Z2$, be single precision storage locations containing real floating point quantities. Then the call

CALL FIXYZ $(X1, X2, Y1, Y2, Z1, Z2, IXYZER)$

performs an interval addition, subtraction, multiplication or division according to the 'value' of *XYZ* on the intervals $(X1, X2)$ and $(Y1, Y2)$ and stores the result in $(Z1, Z2)$. The contents of $X1$, $X2$, $Y1$, $Y2$ remain unaltered. Let Q, R be a single precision storage location containing real floating point quantities. Then the statement

$$R = FXTEND\ (Q)$$

assigns to R the number Q with a 1 added to the last digit of the mantissa.

If either $X1 > X2$ or $Y1 > Y2$ in the call of these routines the errorflag *IXYZER* is set to 1. In the case of *FIDIV*, the errorflag is set to 2 if $Y1 * Y2 \leqslant 0$. Normal return is indicated by the errorflag being set to zero.

Let A, B, C be complex intervals $(A1, A2)$, $(B1, B2)$ and $(C1, C2)$ respectively. The four routines *ADDX*, *SUBX*, *MULX* and *DIVX* perform complex interval arithmetic operations on complex intervals A, B and return the result in C. Let *XYZX* be any one of the above four complex interval arithmetic routines. Then the routines are used according to the call sequence

CALL XYZX $(A1, A2, B1, B2, C1, C2, IXZXER)$

If in the call of the routine *XYZX* any of the conditions $\text{Re}(A1) \leqslant \text{Re}(A2)$, $\text{Im}(A1) \leqslant \text{Im}(A2)$, $\text{Re}(B1) \leqslant \text{Re}(B2)$ or $\text{Im}(B1) \leqslant \text{Im}(B2)$ are violated, the errorflag *IXZXER* is set to 1. In case of *DIVX*, the errorflag is set to 2 if zero is a member of the complex interval B. Normal return from the routines is indicated by the errorflag being set to zero.

We define the area S of a complex interval $(A1, A2)$ to be

$$S(A1, A2) = [\mathrm{Re}\,(A2) - \mathrm{Re}\,(A1)] * [\mathrm{Im}\,(A2) - \mathrm{Im}\,(A1)]\ ,$$

and use the size of this quantity as a measure of how good our algorithm is.

We present an example of the division routine. At the input of

$$(X1, X2) = \{(0{\cdot}10000000E + 01, 0{\cdot}10000000E + 01) + i(0{\cdot}0, 0{\cdot}0)\}$$
$$(Y1, Y2) = \{(0{\cdot}50000000E + 01, 0{\cdot}60000000E + 01) + i(-0{\cdot}30000000E + 01, -0{\cdot}10000000E + 01)\}$$

the computer (IBM 360/50) produced the result

$$(Z1, Z2) = (X1, X2) \div (Y1, Y2) = \{(0{\cdot}13333321E + 00, 0{\cdot}19230783E + 00) + i(0{\cdot}27027003E - 01, 0{\cdot}88235438E - 01)\}.$$

The resulting area is

$$S_1(Z1, Z2) = [0{\cdot}19230783E + 00 - 0{\cdot}13333321E + 00] * [0{\cdot}88235438E - 01 - 0{\cdot}27027003E - 01] = 0{\cdot}36097441E - 02$$

If, however, we use the simpleminded approach on these intervals the result is (see also Rokne-Lancaster 1971)

$$(Z1, Z2) = (X1, X2) \div (Y1, Y2) = \{(0{\cdot}11111099E + 00, 0{\cdot}23076934E + 00) + i(0{\cdot}22222206E - 01, 0{\cdot}11538470E + 00)\}$$

The resulting area is in this case

$$S_2(Z1, Z2) = [0{\cdot}23076934E + 00 - 0{\cdot}11111099E + 00] * [0{\cdot}11538470E + 00 - 0{\cdot}22222206E - 01] = 0{\cdot}11147670E - 01$$

Comparing S_1 and S_2 we see that we have a considerable improvement.

In some cases our division routine will produce a result, whereas the simpleminded approach fails. The reason for this is that the growth of the intervals is slower for our routine than for the simpleminded approach.

For example if we have

$$(X1, X2) = \{(0{\cdot}10000000E + 01, 0{\cdot}10000000E + 01) + i(0{\cdot}0, 0{\cdot}0)\}$$
$$(Y1, Y2) = \{(0{\cdot}10000000E + 01, 0{\cdot}20000000E + 01) + i(-0{\cdot}20000000E - 01, 0{\cdot}30000000E + 01)\}$$

the computer produced the result

$$(Z1, Z2) = (X1, X2) \div (Y1, Y2) = \{(0{\cdot}99999964E - 01, 0{\cdot}10000029E + 01) + i(-0{\cdot}500000066E + 00, 0{\cdot}500000066E + 00)\}$$

using our routine. The simpleminded approach, however, could not handle this division.

References

Gibb, A. (1961). Algorithm 61, Procedures for Range Arithmetic, *Communications of the ACM*, Vol. 4, p. 319.

Rokne, J., and Lancaster, P. (1971). Complex Interval Arithmetic, *Communications of the ACM*, Vol. 14, p. 111.

Acknowledgements

The authors are grateful for financial support from the National Research Council of Canada and Defence Research Board of Canada.

```
      SUBROUTINE ADDX(A1, A2, B1, B2, C1, C2, IADXER)
C     THE SUBROUTINE ADDX PERFORMS A COMPLEX INTERVAL
C     ADDITION ON THE COMPLEX INTERVALS (A1, A2) AND (B1, B2)
C     AND STORES THE RESULT IN (C1, C2). THE COMPLEX INTERVALS
C     (A1, A2) AND (B1, B2) ARE UNALTERED. IADXER IS AN ERROR-
C     FLAG. IT IS SET TO ZERO FOR NORMAL RETURN AND TO 1 TO
C     INDICATE AN ERRORCONDITION.
      COMPLEX A1, A2, B1, B2, C1, C2, CMPLX
      IADXER = 0
      IF (REAL(A1) .LE. REAL(A2) .AND. AIMAG(A1) .LE. AIMAG
     1(A2) .AND.
     2REAL(B1) .LE. REAL(B2) .AND. AIMAG(B1) .LE. AIMAG(B2))
     3GO TO 1
      IADXER = 1
      RETURN
    1 CALL FIADD(REAL(A1), REAL(A2), REAL(B1), REAL(B2), CR, CI,
     1IDUM)
      CALL FIADD(AIMAG(A1), AIMAG(A2), AIMAG(B1), AIMAG(B2),
     1DR, DI, IDUM)
      C1 = CMPLX(CR, DR)
      C2 = CMPLX(CI, DI)
      RETURN
      END

      SUBROUTINE SUBX(A1, A2, B1, B2, C1, C2, ISBXER)
C     THE SUBROUTINE SUBX PERFORMS A COMPLEX INTERVAL
C     SUBTRACTION ON THE COMPLEX INTERVALS (A1, A2) AND
C     (B1, B2) AND STORES THE RESULT IN (C1, C2). THE COMPLEX
C     INTERVALS (A1, A2) AND (B1, B2) ARE UNALTERED. ISBXER IS
C     AN ERRORFLAG. IT IS SET TO ZERO FOR NORMAL RETURN AND
C     TO 1 TO INDICATE AN ERRORCONDITION.
      COMPLEX A1, A2, B1, B2, C1, C2, CMPLX
      ISBXER = 0
      IF (REAL(A1) .LE. REAL(A2) .AND. AIMAG(A1) .LE. AIMAG
     1(A2) . AND .
     2REAL(B1) .LE. REAL(B2) .AND. AIMAG(B1) .LE. AIMAG(B2))
     3GO TO 1
      ISBXER = 1
      RETURN
    1 CALL FISUB(REAL(A1), REAL(A2), REAL(B1), REAL(B2), CR, CI,
     1IDUM)
      CALL FISUB(AIMAG(A1), AIMAG(A2), AIMAG(B1), AIMAG(B2),
     1DR, DI, IDUM)
      C1 = CMPLX(CR, DR)
      C2 = CMPLX(CI, DI)
      RETURN
      END

      SUBROUTINE MULX(A1, A2, B1, B2, C1, C2, IMLXER)
C     THE SUBROUTINE MULX PERFORMS A COMPLEX INTERVAL
C     MULTIPLICATION ON THE COMPLEX INTERVALS (A1, A2) AND
C     (B1, B2) AND STORES THE RESULT IN (C1, C2). THE COMPLEX
C     INTERVALS (A1, A2) AND (B1, B2) ARE UNALTERED. IMLXER IS AN
C     ERRORFLAG. IT IS SET TO ZERO FOR NORMAL RETURN AND TO
C     1 TO INDICATE AN ERRORCONDITION.
      COMPLEX A1, A2, B1, B2, C1, C2, CMPLX
      IMLXER = 0
      IF(REAL(A1) .LE. REAL(A2) .AND. AIMAG(A1) .LE. AIMAG(A2)
     1. AND .
     2REAL(B1) .LE. REAL(B2) .AND. AIMAG(B1) .LE. AIMAG(B2))
     3GO TO 1
      IMLXER = 1
      RETURN
    1 CALL FIMUL(REAL(A1), REAL(A2), REAL(B1), REAL(B2), Q11,
     1Q21, IDUM)
      CALL FIMUL(AIMAG(A1), AIMAG(A2), AIMAG(B1), AIMAG(B2),
     1Q12, Q22, IDUM)
      CALL FISUB(Q11, Q21, Q12, Q22, CR, CI, IDUM)
      CALL FIMUL(REAL(A1), REAL(A2), AIMAG(B1), AIMAG(B2), Q11,
     1Q21, IDUM)
      CALL FIMUL(AIMAG(A1), AIMAG(A2), REAL(B1), REAL(B2),
     1Q12, Q22, IDUM)
      CALL FIADD(Q11, Q21, Q12, Q22, DR, DI, IDUM)
      C1 = CMPLX(CR, DR)
      C2 = CMPLX(CI, DI)
      RETURN
      END

      SUBROUTINE DIVX(A1, A2, B1, B2, C1, C2, IDVXER)
C     THE SUBROUTINE DIVX PERFORMS A COMPLEX INTERVAL
C     DIVISION ON THE COMPLEX INTERVALS (A1, A2) AND (B1, B2)
C     AND STORES THE RESULT IN (C1, C2). THE COMPLEX INTERVALS
C     (A1, A2) AND (B1, B2) ARE UNALTERED. DIVX SORTS OUT AND
C     MAPS THE DIFFERENT INTERVALS INTO THE FIRST AND SECOND
C     QUADRANT: AFTER DIVISION (DIVIQ) THE INTERVAL IS MAPPED
C     BACK INTO THE PROPER POSITION
C     IDVXER IS AN ERRORFLAG. IT IS SET TO ZERO FOR NORMAL
C     RETURN AND TO 1 TO INDICATE NONSTANDARD INTERVALS
C     ON INPUT. IT IS SET TO 2 IF THE DENOMINATOR CONTAINS
C     ZERO.
      COMPLEX A1, A2, B1, B2, C1, C2, D1, D2, CMPLX
      REAL L, M, N, LP, MP, NP
      IDVXER = 0
      IF (REAL(A1) .LE. REAL(A2) .AND. AIMAG(A1) .LE. AIMAG
     1(A2) .AND.
     2REAL(B1) .LE. REAL(B2) .AND. AIMAG(B1) .LE. AIMAG(B2))
     3GO TO 10
      IDVXER = 1
      RETURN
   10 L = REAL(B1)
      M = AIMAG(B1)
      N = REAL(B2)
      P = AIMAG(B2)
      IF(.NOT.(((M.GT.0.0).AND.(L.GE.0.0)).OR.((P.GT.0.0).
     1AND.(L.GT.0.0)
     2))) GO TO 20
      CALL DIV1Q(L, N, M, P, LP, NP, MP, PP)
      GO TO 60
   20 IF(.NOT.(((M.GT.0.0).AND.(L.LT.0.0)).OR.((M.GE.0.0).
     1AND.(N.LT.0.0)
     2))) GO TO 30
      CALL DIV1Q(M, P, -N, -L, PP, MP, LP, NP)
      MP = -MP
      PP = -PP
      GO TO 60
   30 IF(.NOT.(((N.LT.0.0).AND.(M.LT.0.0)).OR.((N.LE.0.0).
     1AND.(P.LT.0.0)
     2))) GO TO 40
      CALL DIV1Q(-N, -L, -P, -M, NP, LP, PP, MP)
      LP = -LP
      MP = -MP
      NP = -NP
      PP = -PP
      GO TO 60
   40 IF(.NOT.(((N.GT.0.0).AND.(P.LT.0.0)).OR.((L.GT.0.0).
     1AND.(P.LE.0.0)
     2))) GO TO 50
      CALL DIV1Q(-P, -M, L, N, MP, PP, NP, LP)
      NP = -NP
      LP = -LP
      GO TO 60
   50 IDVXER = 2
      RETURN
   60 D1 = CMPLX (LP, MP)
      D2 = CMPLX (NP, PP)
      CALL MULX (A1, A2, D1, D2, C1, C2, IDUM)
      RETURN
      END

      SUBROUTINE DIV1Q(X1, X2, Y1, Y2, X1D, X2D, Y1D, Y2D)
C     THE SUBROUTINE DIV1Q TAKES CARE OF THE DIVISION IN THE
C     FIRST AND IN THE FIRST AND SECOND QUADRANT
      CALL FIMUL(X1, X1, X1, X1, AU1, AU2, IDUM)
      CALL FIMUL(Y1, Y1, Y1, Y1, AU3, AU4, IDUM)
      CALL FIADD(AU1, AU2, AU3, AU4, AU5, AU6, IDUM)
      CALL FIDIV(X1, X1, AU5, AU6, AK11, AK12, IDUM)
      CALL FIDIV(Y1, Y1, AU5, AU6, AL11, AL12, IDUM)
      CALL FIMUL(X2, X2, X2, X2, AU7, AU8, IDUM)
      CALL FIMUL(Y2, Y2, Y2, Y2, AU9, AU10, IDUM)
      CALL FIADD(AU7, AU8, AU3, AU4, AU11, AU12, IDUM)
      CALL FIDIV(X2, X2, AU11, AU12, AK21, AK22, IDUM)
      CALL FIDIV(Y1, Y1, AU11, AU12, AL21, AL22, IDUM)
      CALL FIADD(AU7, AU8, AU9, AU10, AU5, AU6, IDUM)
      CALL FIDIV(X2, X2, AU5, AU6, AK31, AK32, IDUM)
      CALL FIDIV(Y2, Y2, AU5, AU6, AL31, AL32, IDUM)
      CALL FIADD(AU1, AU2, AU9, AU10, AU5, AU6, IDUM)
      CALL FIDIV(X1, X1, AU5, AU6, AK41, AK42, IDUM)
      CALL FIDIV(Y2, Y2, AU5, AU6, AL41, AL42, IDUM)
      A1 = AMIN1(AK11, AK21, AK31, AK41)
      A2 = AMAX1(AK12, AK22, AK32, AK42)
      B1 = AMIN1(AL11, AL21, AL31, AL41)
      B2 = AMAX1(AL12, AL22, AL32, AL42)
      X1D = A1
      IF(.NOT.((Y1.GT.X2).OR.
     1((X1.GT.Y2).AND.(Y1.GT.0.0)))) GO TO 10
      X2D = A2
      Y1D = -B2
      Y2D = -B1
      RETURN
   10 CONTINUE
      IF(.NOT.(((Y2.GE.X2).AND.(X2.GE.Y1).AND.(Y1.GE.X1))
     1.OR.
     1((X2.GT.Y2).AND.(Y1.GE.X1)))) GO TO 20
      X2D = AMAX1(A2, FPDIV(1., Y1)/2.)
      Y1D = -B2
      Y2D = -B1
      RETURN
   20 P1 = FPDIV(1., X1)
      Q1 = FPDIV(P1, 2.)
      IF(Y1.LE.0.0) GO TO 30
      X2D = A2
      Y1D = AMIN1(-B2, -Q1)
      Y2D = -B1
      RETURN
   30 X2D = P1
      IF (ABS(Y1).GE.X1) GO TO 40
      Y2D = -B1
      GO TO 50
   40 Y2D = Q1
   50 IF (ABS(Y2).GE.X1) GO TO 60
      Y1D = -B2
      GO TO 70
   60 Y1D = -Q1
   70 CONTINUE
      RETURN
      END

      SUBROUTINE FIADD(X1, X2, Y1, Y2, Z1, Z2, IADDER)
C     FIADD PERFORMS AN INTERVAL ADDITION ON THE INTERVALS
C     (X1, X2) AND (Y1, Y2) AND STORES THE RESULT IN (Z1, Z2).
C     THE CONTENTS OF (X1, X2) AND (Y1, Y2) REMAIN
C     UNCHANGED.
C     IADDER IS AN ERRORFLAG. IT IS SET TO ZERO FOR NORMAL
C     RETURN AND TO 1 TO INDICATE AN ERRORCONDITION.
      IADDER = 0
      IF(X1 .LE. X2 .AND. Y1 .LE. Y2) GO TO 10
      IADDER = 1
      RETURN
   10 Z1 = X1 + Y1
      Z2 = X2 + Y2
      IF (Z1) 1, 2, 3
    1 CALL FXTEND(Z1)
    2 IF (Z2) 4, 4, 3
    3 CALL FXTEND (Z2)
    4 RETURN
      END

      SUBROUTINE FISUB(X1, X2, Y1, Y2, Z1, Z2, ISUBER)
C     FISUB PERFORMS AN INTERVAL SUBTRACTION ON THE INTER-
C     VALS (X1, X2) AND (Y1, Y2) AND STORES THE RESULT IN (Z1, Z2).
C     THE CONTENTS OF (X1, X2) AND (Y1, Y2) REMAIN UNCHANGED.
C     ISUBER IS AN ERRORFLAG. IT IS SET TO ZERO FOR NORMAL
C     RETURN AND TO 1 TO INDICATE AN ERRORCONDITION.
      ISUBER = 0
      IF (X1 .LE. X2 .AND. Y1 .LE. Y2) GO TO 10
      ISUBER = 1
      RETURN
   10 Z1 = X1 - Y2
      Z2 = X2 - Y1
      IF (Z1) 1, 2, 3
    1 CALL FXTEND (Z1)
    2 IF (Z2) 4, 4, 3
    3 CALL FXTEND (Z2)
    4 RETURN
      END

      SUBROUTINE FIMUL(X1, X2, Y1, Y2, Z1, Z2, IMULER)
C     FIMUL PERFORMS AN INTERVAL MULTIPLICATION ON THE
C     INTERVALS (X1, X2) AND (Y1, Y2) AND STORES THE RESULT IN
C     (Z1, Z2)
C     THE CONTENTS OF (X1, X2) AND (Y1, Y2) REMAIN UNCHANGED.
C     IMULER IS AN ERRORFLAG. IT IS SET TO ZERO FOR NORMAL
C     RETURN AND TO 1 TO INDICATE AN ERRORCONDITION.
      IMULER = 0
      IF (X1 .LE. X2 .AND. Y1 .LE. Y2) GO TO 10
      IMULER = 1
      RETURN
   10 Z1 = AMIN1(X1*Y1, X1*Y2, X2*Y1, X2*Y2)
      Z2 = AMAX1(X1*Y1, X1*Y2, X2*Y1, X2*Y2)
      IF (Z1) 1, 2, 3
    1 CALL FXTEND (Z1)
    2 IF (Z2) 4, 4, 3
    3 CALL FXTEND (Z2)
    4 RETURN
      END

      SUBROUTINE FIDIV(X1, X2, Y1, Y2, Z1, Z2, IDIVER)
C     FIDIV PERFORMS AN INTERVAL DIVISION ON THE INTERVALS
C     (X1, X2) AND (Y1, Y2) AND STORES THE RESULT IN (Z1, Z2)
C     THE CONTENTS OF (X1, X2) AND (Y1, Y2) REMAIN UNCHANGED.
C     IDIVER IS AN ERRORFLAG. IT IS SET TO ZERO FOR NORMAL
C     RETURN AND TO 1 TO INDICATE NONSTANDARD INTERVALS
C     ON INPUT. IT IS SET TO 2 IF THE DENOMINATOR CONTAINS
C     ZERO.
      IDIVER = 0
      IF (X1 .LE .X2 .AND. Y1 .LE. Y2) GO TO 10
      IDIVER = 1
      RETURN
   10 IF (Y1*Y2 .GT. 0.0) GO TO 20
      IDIVER = 2
      RETURN
   20 A2 = 1./Y1
      A1 = 1./Y2
      IF (A1) 1, 2, 3
    1 CALL FXTEND(A1)
    2 IF (A2) 4, 4, 3
    3 CALL FXTEND(A2)
    4 CONTINUE
      RETURN END CALL FIMUL (X1, X2, A1, A2, Z1, Z2, IDUM)

      FUNCTION FPDIV(A, B)
      C = A/B
      IF (C) 1, 1, 2
    2 CALL FXTEND(C)
    1 FPDIV = C
      RETURN
      END

      SUBROUTINE FXTEND(X)
C     FXTEND SHIFTS X TO THE NEXT MACHINE REPRESENTABLE
C     VALUE. THE VALUE OF N, M, B AND L MUST BE SUPPLIED BY THE
C     USER. THE VALUES BELOW ARE FOR IBM/360 FORTRAN.
C     L IS THE LENGTH OF THE MANTISSA.
C     B IS THE BASE.
C     M IS THE LOWER AND N IS THE UPPER BOUND FOR THE
C     EXPONENT. IT IS RECOMMENDED THAT THIS ROUTINE BE
C     REWRITTEN IN MACHINE CODE WHEN THE PACKAGE IS
C     IMPLEMENTED.
      IF (X) 2, 3, 2
    3 RETURN
    2 M = -62
      B = 16.
      N = 61
      L = 6
      J = N - M
      Y = ABS(X)
      DO 1 I = 1, J
      K = I + M - 1
      A = B**K
      IF (.NOT.(A.LT. Y .AND. Y .LT. A*B)) GO TO 1
      X = SIGN(Y + B**(K + 1 - L), X)
      RETURN
    1 CONTINUE
      END
```

Algorithm 87

MINIMUM OF A NON-LINEAR FUNCTION BY THE APPLICATION OF THE GEOMETRIC PROGRAMMING TECHNIQUE

Robert Fleck and John Bailey
Department of Management Science
University of South Carolina
Columbia, South Carolina 29208 USA

Author's note

Introduction

Geometric programming can be used to find the minimum of a specific type of a non-linear function. For this specific subset of non-linear functions, the minimum value can be determined by geometric programming *without* calculating the values of the variables. This method does not require the solution of non-linear equations or differentiation. The types of functions for which this technique applies are found in industrial engineering (Duffin, Petersen and Zener, 1967; Federowicz and Mazumdar, 1968; Passy, 1970; Wilde and Beightler, 1967, pp. 27-30), inventory management (Kochenberger, 1971), and computer science (Chow, 1974).

Methodology

The methodology locates the minimum value by 'inspection' of the exponents in the terms of the non-linear function. The theoretical basis and initial applications were made by Zener, Petersen and Duffin (1967). The theoretical foundation relies on the geometric-arithmetic mean inequality. The function to be minimised, which is a series of terms involving products of variables, is converted to a weighted geometric mean and the weights determined which cause the geometric mean to be maximised and equal to a weighted arithmetic mean.

Example

Assume we have a function to be minimised:

$$y = \sum_{j=1}^{N} \left(C_j \prod_{i=1}^{M} X_i^{a_{ij}} \right) \qquad (1)$$

where C_j is the coefficient of each of N terms, a_{ij} is the exponent of variable i in term j. An explicit example (Wilde and Beightler, 1967, pp. 27-30) is

$$y = 1000X_1 + 4 \times 10^9 X_1^{-1} X_2^{-1} + 2\cdot5 \times 10^5 X_2 \qquad (2)$$

If we took derivatives, the first partials of (1) must be zero at the optimal solution:

$$\begin{aligned} \left(\frac{\partial y}{\partial X_K}\right) &= \sum_{j=1}^{N} C_j a_{Kj} (X_K)^{a_{Kj}-1} \prod_{i \neq K} X_i^{a_{ij}} \\ &= \frac{1}{X_K} \sum_{j=1}^{N} a_{Kj} C_j X_K^{a_{Kj}} \prod_{i \neq k} X_i^{a_{ij}} \\ &= \frac{1}{X_K} \sum_{j=1}^{N} a_{Kj} C_j \prod_{i=1}^{M} X_i^{a_{ij}} = 0 \qquad (3) \\ \therefore \quad &= \sum_{j=1}^{N} a_{Kj} C_j \prod_{i=1}^{M} X_i^{a_{ij}} = 0 \end{aligned}$$

for $K = 1, \ldots, M$.

Define a weight, w_j, as:

$$w_j = \frac{C_j \prod_{i=1}^{M} X_i^{a_{ij}}}{y^*} \qquad (4)$$

for each term j where y^* is the optimal functional value and the X_i are the optimal values determined from the solution of (3).

Transforming (4) to

$$w_j y^* = C_j \prod_{i=1}^{M} X_i^{a_{ij}} \qquad (5)$$

allows substitution of (5) into (3) yielding

$$y^* \sum a_{Kj} w_j = 0$$

or

$$\sum_{=1}^{N} a_{Kj} wj = 0 \qquad (6)$$

for $K = 1, \ldots, M$.

From equation (1) with optimal values of each X_i, the optimal function value:

$$y^* = \sum_{j=1}^{N} \left(C_j \prod_{i=1}^{M} X_i^{a} \right) \qquad (7)$$

Dividing equation (7) by y^*,

$$\frac{y^*}{y^*} = 1 = \sum_{j=1}^{N} \left(C_j \prod_{i=1}^{M} X_i^{a_{ij}} \right) / y^* \qquad (8)$$

Substituting for each term j in equation (8), w_j as defined in equation (4), we have

$$\sum_{j=1}^{N} w_j = 1 \ . \qquad (9)$$

If we form the N linear equations in N unknowns based on (6) and (9) for the example (2),

$$\begin{aligned} w_1 + w_2 + w_3 &= 1 \\ w_1 - w_2 \quad &= 0 \qquad (10) \\ - w_2 + w_3 &= 0 \end{aligned}$$

The solution to (10) yields $w_1 = w_2 = w_3 = 1/3$.

From (9), (4):

$$y^* = \prod_{j=1}^{N} (y^*)^{w_j} = \prod_{j=1}^{N} \left(\frac{C_j \prod_{i=1}^{M} X_i^{a_{ij}}}{w_j} \right)^{w_j} = \prod_{j=1}^{N} \left(\frac{C_j}{w_j}\right)^{w_j} \left(\prod_{i=1}^{M} X_i^{a_{ij}} \right)^{w}$$

It can be shown that

$$\prod_{j=1}^{N} \left(\prod_{i=1}^{M} X_i^{a_{ij}} \right)^{w_j} = 1$$

Therefore

$$y^* = \prod_{j=1}^{N} \left(\frac{C_j}{w_j}\right)^{w_j} \qquad (11)$$

and hence for (2), (11) becomes:

$$y^* = \left(\frac{1000}{1/3}\right)^{1/3} \left(\frac{4 \times 10^9}{1/3}\right)^{1/3} \left(\frac{2\cdot5 \times 10^5}{1/3}\right)^{1/3} = 3 \times 10^6 \ .$$

The solution to (10) and (11) yields the values for each term of 1×10^6.

$$\begin{aligned} 1000X^1 &= 10^6 \\ 4 \times 10^9 X_1^{-1} X_2^{-1} &= 10^6 \qquad (12) \\ 2\cdot5 \times 10^5 X_2 &= 10^6 \end{aligned}$$

which can be solved by inspection or by taking logarithms. The solution yields $X_1 = 10^3$ and $X_2 = 4$.

Relationship between arithmetic and geometric means

It can be shown that the arithmetic mean is always greater than or equal to the geometric mean of the same numbers (McMillan, 1970, pp. 224-227). This relationship holds as long as all numbers in the set are positive and non-zero. The condition of equality between the means exists when all the numbers in the set are identical. By converting a function (via weights) into its corresponding arithmetic and geometric means and then determining the weights which maximize the geometric mean, the minimum arithmetic mean and the minimum functional value are found (McMillan, 1970, pp. 224-227).

Define equation (13) as follows:

$$y' = \frac{1000X_1}{w_1} + \frac{4 \times 10^9 X^{-1} X_2^{-1}}{w_2} + \frac{2\cdot5 \times 10^5 X_2}{w_3} \qquad (13)$$

where the w_j's are variables to be defined later.

Now convert (13) to equation (2) by multiplying each term by the appropriate w_j,

$$y = \frac{1000X_1}{w_1} w_1 + \frac{4 \times 10^9 X^{-1} X_2{}^{-1}}{w_2} w_2 + \frac{2{\cdot}5 \times 10^5 X_2}{w_3} w_3 \quad (14)$$

If we restrict the w_j's to positive and non-zero values less than one, summing to one, equation (14) is the weighted arithmetic mean of equation (13). The weighted geometric mean of (13) is:

$$G = \left(\frac{1000X_1}{w_1}\right)^{w_1} \left(\frac{4 \times 10^9 X^{-1} X_2{}^{-1}}{w_2}\right)^{w_2} \left(\frac{2{\cdot}5 \times 10^5 X_2}{w_3}\right)^{w_3} \quad (15)$$

If we simplify (15)

$$G = \left(\frac{1000X_1}{w_1}\right)^{w_1} \left(\frac{4 \times 10^9}{w_2}\right)^{w_2} \left(\frac{2{\cdot}5 \times 10^5}{w_3}\right)^{w_3} X_1{}^{(w_1 - w_2)} X_2{}^{(-w_2 + w_3)} \quad (16)$$

It can be shown (McMillan, 1970, pp. 224-227) that the maximum of (16) is found when the sum of the exponents of each of the decision variables is equal to zero. This condition plus the condition that the w_j's sum to one yields the set of equations (10). The maximum of (16) is the minimum of (14), the function to be minimised.

Applications

Applications for geometric programming examples can be found in the references cited earlier. More complex forms involving constrained optimisation problems and forms which violate some of the assumptions for the simple cases can also be found in these references.

One commonly occurring problem for the simple case is found in inventory control. The total cost equation for an inventory system under conditions of instantaneous delivery would appear as:

$$TC = \frac{K_c Q}{2} + \frac{D}{Q} K_0 \quad (15)$$

where K_c is a constant cost of holding an item in inventory per time period, K_0 is the cost of placing an order, D is the demand per time period and Q is the amount to order which minimises the total cost, TC. Given values for the constants K_c, K_0, and D, this problem can be solved using geometric programming.

Limitations

Since all $w_j \geq 0$, equation (11) is solvable for real values only if all $C_j > 0$. To get a system of equations (10) the number of terms must be larger than the number of variables. It is also possible to generate negative weights or no solution to (11). This may occur when variables appear only with positive or only with negative exponents. To insure a solution, each variable should appear with positive and negative exponents.

The degree of difficulty is defined as:

the number of terms − (the number of variables + 1) .

The example here has zero degrees of difficulty and is unconstrained. Solution techniques are available for cases when the degree of difficulty is not zero, the coefficients are negative, only positive (or negative) exponents exist, and non-linear constraints are involved (Duffin, Petersen and Zener, 1967; McMillan, 1970; Wilde and Beightler, 1967, Chapter 4). The techniques are much more cumbersome. However, there are still a large number of cases for which the methodology given here will yield a solution. The program which follows solves problems of the limited case.

References

Chow, C. K. (1974). On Optimization of Storage Hierarchies, *IBM Journal of Research and Development*, Vol. 18, No. 3, pp. 194-204.

Duffin, Petersen, and Zener (1967). *Geometric Programming*. New York: John Wiley and Sons, Inc.

Federowicz, A. J., and Mazumdar, M. (1968). Use of Geometric Programming to Maximize Reliability Achieved by Redundancy, *Operations Research*, Vol. 16, No. 5, pp. 948-954.

Kochenberger, G. A. (1971). Inventory Models: Optimization by Geometric Programming, *Decision Sciences*, Vol. 2, No. 2, pp. 193-205.

McMillan, C., Jr. (1970). *Mathematical Programming*. New York: John Wiley and Sons, Inc., Chapter VII.

Passy, U. (1970). Modular Design: An Application of Structured Geometric Programming, *Operations Research*, Vol. 18, No. 3, pp. 441-453.

Wilde, D., and Beightler, C. (1967). *Foundations of Optimization*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

```
C    MAIN DRIVER
C
C    INPUT:
C    NUMBER OF TERMS:          COLUMNS 1-5. DATA CARD ONE.
C                              INTEGER CONSTANT.
C    NUMBER OF VARIABLES:      COLUMNS 6-10. DATA CARD ONE.
C                              INTEGER CONSTANT.
C    COST COEFFICIENTS:        COLUMNS IN MULTIPLES OF TWENTY.
C                              COST FOR TERM ONE IN COLUMNS
C                              1-20, TERM TWO, 21-40, ETC. START
C                              ENTERING COEFFICIENTS ON DATA
C                              CARD TWO AND CONTINUE FOR AS
C                              MANY CARDS AS NECESSARY. REAL
C                              CONSTANTS.
C    VARIABLE EXPONENTS:       COLUMNS IN MULTIPLES OF FIVE.
C                              EACH DATA CARD CONTAINS THE
C                              EXPONENTS FOR ALL VARIABLES FOR
C                              A GIVEN TERM. VARIABLE ONE, TERM
C                              ONE IN COLUMNS 1-5, VARIABLE
C                              TWO, TERM ONE IN COLUMNS 6-10,
C                              ETC. USE AS MANY CARDS AS THERE
C                              ARE TERMS. PLACE THESE CARDS AT
C                              THE END OF THE DATA DECK. IF A
C                              VARIABLE IS MISSING FROM A TERM,
C                              ENTER A ZERO FOR THE EXPONENT.
C                              REAL CONSTANTS.
           DIMENSION C(11), A(11, 11)
           DIMENSION WB(11), S(11)
C          DETERMINE THE NUMBER OF TERMS (N) AND VARIABLES
C          (M).
           READ (5, 1000) N, M
     1000  FORMAT (2I5)
C          DETERMINE N COST COEFFICIENTS.
           READ (5, 1010) (C(I), I = 1, N)
     1010  FORMAT (4E20.7)
           WRITE (6, 1020) N, M (C (I),I  = 1, N)
     1020  FORMAT  (1H1///55X,  21HGEOMETRIC  PROGRAMMING//
           151X, I3, 10H TERMS AND, I3, 10H VARIABLES///49X,
           231H COST COEFFICIENTS FOR EACH TERM// (6E21.7))
           WRITE (6, 1030) (I, I = 1, M)
     1030  FORMAT  (////54X,  22HEXPONENTS  OF  VARIABLES//
           16H TERMS, 54X, 9HVARIABLES // 10I12/)
C          DETERMINE EXPONENTS OF VARIABLES (I) IN TERM (J).
           DO 10 J = 1, N
       10  READ (5, 1040) (A(I, J), I = 1, M)
     1040  FORMAT (10F5.0)
           DO 20 J = 1, N
       20  WRITE (6, 1050) J, (A(I, J), I = 1, M)
     1050  FORMAT (I3, 10E12.3)
           CALL GEOM (N, M, C, A, IER, Y, WB, S)
           GO TO (70, 30, 40, 50, 60), IER
       30  WRITE (6, 1060)
     1060  FORMAT  (//47X,  26 HMORE  THAN  ZERO  DEGREES  OF
           110HDIFFICULTY)
           STOP
       40  WRITE (6, 1070)
     1070  FORMAT (//18HMATRIX IS SINGULAR)
           STOP
       50  WRITE (6, 1080)
     1080  FORMAT (//30H NEGATIVE OR ZERO COEFFICIENTS)
           STOP
       60  WRITE (6, 1090)
     1090  FORMAT (//25H NEGATIVE OR ZERO WEIGHTS)
           STOP
       70  WRITE (6,1100) (I, I, = 1, N)
     1100  FORMAT  (//55X,  21HWEIGHTS  FOR  EACH  TERM//63X,
          14HTERM/10X, 11I10//)
           WRITE (6, 1110) (WB(I), I = 1, N)
     1110  FORMAT (15X, 11F10.6//)
           WRITE (6, 1120) Y
     1120  FORMAT (///43X, 30HVALUE OF OBJECTIVE FUNCTION = ,
          1E14.7)
           WRITE (6, 1130)
     1130  FORMAT (//46X, 8HVARIABLE, 14X, 5HVALUE)
           DO 80 J = 1, M
       80  WRITE (6, 1140) J, S(J)
     1140  FORMAT (45X, I5, 15X, E14.7)
           WRITE (6, 1150)
     1150  FORMAT (1H1)
           STOP
           END

         SUBROUTINE GEOM(N, M, C, A, IER, Y, WB, S)
C    THIS PROGRAM WILL DETERMINE THE MINIMUM OF A NON-
C    LINEAR FUNCTION USING THE GEOMETRIC PROGRAMMING
C    TECHNIQUE.
C
C    LIMITATIONS: THE FUNCTION MUST HAVE ONE MORE TERM
C                 THAN VARIABLES. THE FUNCTION MUST HAVE
C                 POSITIVE COEFFICIENTS.
```

```
C                 EVERY VARIABLE MUST APPEAR WITH ONE NEG-
C                 ATIVE AND ONE POSITIVE EXPONENT.
C     REFERENCES: MC MILLAN, MATHEMATICAL PROGRAMMING,
C                 JOHN WILEY AND SONS, NEW YORK (1970)
C                 PAGES 220-241.
C                 WILDE + BEIGHTLER, FOUNDATIONS OF OPTI-
C                 MIZATION, PRENTICE-HALL, INC., ENGLEWOOD
C                 CLIFFS, N.J. (1967) PAGES 27-30, 99-133.
C
      DIMENSION C(11), A(11, 11)
C     C IS A ROW VECTOR OF TERMC OEFFICIENTS
C     A IS A MATRIX OF EXPONENTS OF VARIABLES IN EACH TERM
C       EACH ROW IS A VARIABLE
C       EACH COLUMN IS A TERM
C
C     N: NUMBER OF TERMS (MAXIMUM 11)
C     M: NUMBER OF VARIABLES (MAXIMUM 10)
C     THESE MAXIMA CAN BE CHANGED BY ALTERING THE DIMEN-
C     SION STATEMENTS IN ALL THE SUBROUTINES.
C
C     IER: ERROR RETURN
C       1 NO ERRORS
C       2 MORE THAN ZERO DEGREES OF DIFFICULTY
C       3 SINGULAR MATRIX
C       4 NEGATIVE OR ZERO TERM COEFFICIENTS
C       5 NEGATIVE OR ZERO WEIGHTS
C
C     SUBROUTINES REQUIRED:  MATINV (INVERSION ROUTINE)
C                            MATMLT (MATRIX MULTIPLICATION)
C
        DIMENSION WA(11), WB(11), AS(11, 11), WC(11), L(11), L1(11),
        1S(11)
        IER = 1
C       CHECK FOR ZERO DEGREES OF DIFFICULTY
        IF (N - (M + 1)) 10, 20, 10
   10   IER = 2
        RETURN
C       CHECK COEFFICIENTS
   20   DO 40 J = 1, N
        IF (C(J)) 30, 30, 40
   30   IER = 4
        RETURN
   40   CONTINUE
C       EXPONENTS FORM M LINEAR EQUATIONS IN N VARIABLES.
C       WEIGHTS MUST SUM TO ONE.
        DO 50 J = 1, N
   50   A(N, J) = 1.
C       SAVE PART OF A FOR LATER USE IN SOLUTION OF DE-
C       CISION VARIABLES.
        DO 60 I = 1, M
        DO 60 J = 1, M
   60   AS (J, I) = A(I, J)
C       SOLVE FOR WEIGHTS BY TAKING INVERSE.
        CALL MATINV (A, N, D, L, L1)
        IF (D) 80, 70, 80
   70   IER = 3
        RETURN
C       SET UP CONSTANTS FOR LINEAR EQUATION.
   80   DO 90 I = 1, M
   90   WA(I) = 0.
        WA(N) = 1.
C       MULTIPLY INVERSE TIMES COLUMN VECTOR OF CONSTANTS
C       TO GET WEIGHTS
        CALL MATMLT (A, WA, WB, N, N, 1)
C       CALCULATE VALUE OF OBJECTIVE FUNCTION.
        Y = 1.
        DO 110 J = 1, N
C       CHECK FOR NEGATIVE OR ZERO WEIGHTS
        IF (WB(J)) 100, 100, 110
  100   IER = 5
        RETURN
  110   Y = Y*((C(J)/WB(J))**WB(J))
C       DETERMINE VALUES OF DECISION VARIABLES.
C       REMOVE CONSTANT (C(J)) TO GET NON-LINEAR EQUATION
C       IN FORM OF VARIABLES = CONSTANT. TAKING LOGS OF
C       BOTH SIDES TRANSLATES INTO EXPONENT*LOG(VAR(I)) +
C       EXPONENT*LOG(VAR(I + 1)) + . . . + EXP*LOG(VAR(M)) =
C       LOG(CONSTANT) WHICH IS SYSTEM OF N LINEAR EQUA-
C       TIONS IN M VARIABLES. NTH EQUATION IS REDUNDANT.
C       SOLVE FOR LOG(VAR) USING MATRIX METHODS.
        CALL MATINV (AS, M, D, L, L1)
        IF (D) 130, 120, 130
  120   IER = 3
        RETURN
  130   DO 140 J = 1, M
  140   WA(J) = ALOG((WB(J)*Y)/C(J))
        CALL MATMLT (AS, WA, S, M, M, 1)
C       SOLUTION OF VARIABLES
        DO 150 J = 1, M
  150   S(J) = EXP(S(J))
        RETURN
        END

        SUBROUTINE MATINV(A, N, D, L, M)
C     THE INVERSE OF THE MATRIX A IS CALCULATED USING GAUSS-
C     JORDAN WITH COMPLETE PIVOTING. THE INVERSE REPLACES
C     THE ORIGINAL MATRIX. L AND M ARE WORK VECTORS OF
C     LENGTH N. THE DETERMINANT D IS CALCULATED.
C         REFERENCE: IBM'S SCIENTIFIC SUBROUTINE PACKAGE;
C                    MINV
        DIMENSION A(11, 11), L(11), M(11)
        D = 1.0
        DO 190 K = 1, N
        L(K) = K
        M(K) = K
        BIG = A(K, K)
        DO 20 I = K, N
        DO 20 J = K, N
        IF (ABS(BIG) - ABS(A(I, J))) 10, 20, 20
   10   BIG = A(I, J)
        L(K) = I
        M(K) = J
   20   CONTINUE
C     CHECK FOR SINGULARITY
        IF (BIG) 40, 30, 40
   30   D = 0.0
        RETURN
C     INTERCHANGE ROWS
   40   I = L(K)
        IF (I - K) 50, 70, 50
   50   DO 60 J = 1, N
        TEMP = -A(K, J)
        A(K, J) = A(I, J)
   60   A(I, J) = TEMP
C     INTERCHANGE COLUMNS
   70   J = M(K)
        IF (J - K) 80, 100, 80
   80   DO 90 I = 1,N
        TEMP = -A(I, K)
        A(I, K) = A(I, J)
   90   A(I, J) = TEMP
C     DIVIDE COLUMN BY MINUS PIVOT
  100   DO 120 I = 1, N
        IF (I - K) 110, 120, 110
  110   A(I, K) = A(I, K)/(-BIG)
  120   CONTINUE
C     REDUCE MATRIX
        DO 160 I = 1, N
        IF (I - K) 130, 160, 130
  130   TEMP = A(I, K)
        DO 150 J = 1, N
        IF (J - K) 140, 150, 140
  140   A(I, J) = TEMP*A(K, J) + A(I, J)
  150   CONTINUE
  160   CONTINUE
C     DIVIDE ROW BY PIVOT
        DO 180 J = 1, N
        IF (J - K) 170, 180, 170
  170   A(K, J) = A(K, J)/BIG
  180   CONTINUE
C     CALCULATE DETERMINANT
        D = D*BIG
C     TAKE RECIPROCAL
  190   A(K, K) = 1.0/BIG
C     BACK SUBSTITUTION
        NM1 = N - 1
        IF (NM1) 200, 270, 200
  200   DO 260 KK = 1, NM1
        K = N - KK
        J = L(K)
        IF (J - K) 210, 230, 210
  210   DO 220 I = 1, N
        TEMP = A(I, K)
        A(I, K) = -A(I, J)
  220   A(I, J) = TEMP
  230   I = M(K)
        IF (I - K) 240, 260, 240
  240   DO 250 J = 1, N
        TEMP = A(K, J)
        A(K, J) = -A(I, J)
  250   A(I, J) = TEMP
  260   CONTINUE
  270   RETURN
        END


        SUBROUTINE MATMLT(A, B, C, N, M, L)
        DIMENSION A(11, M), B(11, L), C(11, L)
C     THIS SUBROUTINE MULTIPLIES AN N BY M MATRIX(A) BY AN M
C     BY L MATRIX(B) TO FORM AN N BY L OUTPUT MATRIX (C)
        DO 10 J = 1, L
        DO 10 I = 1, N
        C(I, J) = 0.0
        DO 10 K = 1, M
   10   C(I, J) = C(I, J) + A(I, K) * B(K, J)
        RETURN
        END
```

Test results

1. Minimise

$$TC = \frac{3Q}{2} + \frac{15{,}000}{Q}$$

Solution:

$$Q = 100$$
$$TC = 300$$

This is a solution to the Economic Order Quantity problem and is due to: Bierman, Bonini and Hausman, *Quantitative Analysis for Business Decisions*, 4th edition, Homewood, Illinois: Irwin (1973), pp. 174-175.

2. Minimise

$$f = 40/X_1X_2X_3 + 10X_1X_2 + 40X_2X_3 + 20X_1X_3$$

Solution:

$$f = 100$$
$$X_1 = 2$$
$$X_2 = 1$$
$$X_3 = 0{\cdot}5$$

Due to: Claude McMillan, Jr., *Mathematical Programming*, New York: John Wiley and Sons, Inc. (1970), pp. 229-230.

3. Minimise

$$y = 1{,}000X_1 + 4 \times 10^9 X_1^{-1} X_2^{-1} + 2{\cdot}5 \times 10^5 X_2$$

Solution:

$$y = 3 \times 10^6$$
$$X_1 = 1{,}000$$
$$X_2 = 4$$

Due to: Wilde and Beightler, *Foundations of Optimization*, Englewood Cliffs, New Jersey: Prentice-Hall, Inc. (1967), pp. 28-30.

Algorithm 88

AN ELECTORAL METHOD

I. D. Hill,
Clinical Research Centre,
Watford Road, Harrow,
Middx., HA1 3UJ

Author's note

In Hill (1974) I suggested a voting method for the case of many candidates for one position, in which the analysis of votes was made by assessing the result of a 'straight fight' between each possible pair of candidates.

The present algorithm is an extension of the method, for more than one position to be filled, giving an order of preference so that the appropriate number of successful candidates may be taken from the top of the ordered list.

The voters are expected to place the number 1 against their first choice, 2 against their second choice, and so on. Since ties are allowed, the difficulty of having to rank *every* candidate from a large number is avoided. Thus if a voter merely desires to vote for candidate *A* and has no preferences among the rest, he can mark *A* as 1, and all the rest as 2. On the other hand, if the only thing he really cares about is voting against candidate *D*, he can mark *D* as 2 and all the rest as 1.

It is further suggested that the voter should be allowed to leave names unnumbered, and any such would be counted as having been ranked equal bottom, below all the numbered ones.

This method has the advantage of rendering 'strategic voting' useless and lets all voters vote according to their genuine preferences without the fear of thereby producing an unwanted result. It also makes 'wrecking' candidatures impossible, as 'splitting' the vote has no effect.

Where there is more than one vacancy, it will produce as second choice the candidate who would have been first choice if the actual first choice had not been standing, and so on for further choices. It cannot therefore be expected to produce a *proportionally representative* result as between parties. Where voting is for individuals rather than for parties this is probably not a disadvantage.

Sometimes ties, or inconsistencies, may be found in the resulting order of preference. In these circumstances several candidates will be given the same preference number in the list produced by the algorithm, and the humans will be left to sort the situation out. This seems preferable to any automatic solution depending on random selection. For a discussion of the inconsistencies that may arise, even though each individual's voting must be self-consistent, see Gardner (1974).

The format of the data should be one line for each voter, starting in column 1 and giving two digits (including leading zeros) for each preference number, with no spaces between numbers. An unnumbered candidate should be recorded as preference number 99.

For example, if there are six candidates, and a voter's preferences are:

Candidate	Preference
1	2
2	1
3	
4	2
5	
6	

the data line for that voter should be 020199029999.

The parameters *inch* and *outch* refer to input and output channel numbers.

References

Gardner, M. (1974). Mathematical Games—on the paradoxical situations that arise from nontransitive relations. *Scient. Amer.*, Vol. 231, No. 4, pp. 120-124.

Hill, I. D. (1974). A new suggestion for multi-candidate elections. *Honeywell Computer J.*, Vol. 8, pp. 107-109.

```
procedure elect (voters, candidates, inch, outch);
value voters, candidates, inch, outch;
integer voters, candidates, inch, outch;
  begin integer i, j, k, m, n; Boolean f, h;
  integer array a[1:candidates, 1:candidates], b[1:candidates];
  for j := 1 step 1 until candidates do
  for k := 1 step 1 until candidates do a[j, k] := 0;
  m := candidates − 1;
  for i := 1 step 1 until voters do
    begin
    comment read the voter's preference number for each candidate;
    for j := 1 step 1 until candidates do input1(inch, 'DD', b[j]);
    input0(inch, '/');
    comment determine result for straight fight between each possible
    pair;
    for j := 1 step 1 until m do
    for k := j + 1 step 1 until candidates do
    if b[j] > b[k] then a[j, k] := a[j, k] + 1 else
    if b[k] > b[j] then a[k, j] := a[k, j] + 1
    end i loop;
  for j := 1 step 1 until candidates do b[j] := 0;
  comment give results for each possible straight fight, and form table
  of number of candidates beaten by each candidate;
  for j := 1 step 1 until candidates do
    begin
    output0(outch, '/');
    for k := 1 step 1 until candidates do
    if j ≠ k then
      begin
      i := a[j, k]; n := a[k, j];
      output1(outch, '/'Candidate␣' ZD', j);
      if i < n then
        begin
        output0(outch, "   beats"); b[j] := b[j] + 1
        end

      else if i > n then output0(outch, "␣loses␣to")
      else output0(outch, "␣ties␣with");
      output3(outch, "␣candidate␣' ZD' ␣by␣' ZD '␣votes␣
      to␣' ZD', k, n, i)
      end k loop
    end j loop;
  comment produce table of overall preferences;
  output0(outch, '///'Order␣of␣preference:'/');
  i := 1; f := false;
```

```
for m := m step −1 until 0 do
  begin
  h := false;
  for j := 1 step 1 until candidates do
  if b[j] = m then
    begin
    if h then f := true else
      begin
      h := true;
      output1(outch, ‘/‘Choice␣number␣’ZD
      ‘␣:␣ candidate␣number␣’’, i)
      end;
    output1(outch, ‘ZDBB’,j); i := i + 1
    end j loop
  end m loop;
if f then
  begin
  comment there has been at least one tie or inconsistency;
  output0(outch, ‘//‘Where␣several␣candidates␣appear␣
  against␣the_same␣choice’/‘number,␣the␣voting␣has␣
  been␣such␣that␣it␣is␣not␣possible’/‘to␣choose␣
  between␣them␣by␣this␣method’’)
  end
end elect
```

Note on Algorithms 78 and 88

COUNTING PREFERENTIAL VOTES IN MULTI-MEMBER CONSTITUENCIES USING ABSOLUTE MAJORITY CRITERIA and AN ELECTORAL METHOD

I. D. Hill,
Clinical Research Centre,
Watford Road, Harrow,
Middx., HA1 3UJ

These algorithms have been compared using the data given with Algorithm 78. Perhaps the data may have been chosen deliberately to demonstrate a difficult case, for Algorithm 78 frequently finds tied values and has to resort to a pseudo-random selection. This explains why, in the results given with the algorithm the computer run elects candidates *F*, *E*, *D* and *C* while manual counting using the same method elects *A*, *D*, *C* and *E* (in that order of preference in each case).

Examination of the method and the data shows that these are not the only possibilities. **Fig. 1** shows the possible results, with the probability that each path is taken (on the assumption that a good random number generator is used instead of the RANF function of Algorithm 78 (Hill and Wedderburn, 1974)).

Among the strange results from this method, it may be noted that candidate *F* has a 50 per cent chance of being elected as first choice, yet if he loses the toss at that point he cannot be elected until the sixth choice, even though there are only seven candidates altogether.

By contrast, Algorithm 88 gives candidate *C* as definite first choice, and *D* as second. For third, fourth and fifth places there is some doubt since

A beats *B* (11 votes to 9)
A ties with *E* (10 votes all)
B ties with *E* (10 votes all)

Algorithm 88 does not sort this out, but presents the evidence of why the situation is difficult. *F* comes sixth and *G* seventh without question.

The results from Algorithm 88 are therefore much neater than those from Algorithm 78. Of course, a neat result is not necessarily preferable to an untidy one if there are other ways in which the untidy one is preferable, but I cannot see any.

As there were four places to be filled, *C* and *D* would be elected, while *F* and *G* would be eliminated. I suggest that the remaining two places should be filled by means of a further election with *A*, *B* and *E* as the only candidates.

The result could be inconsistent again. With the voters' minds concentrated on just these three it would be less likely to be, but some tie-breaking rule would have to be available in case it were. There are several possibilities, but almost anything is better than choosing at random.

I acknowledge some comments from Tran Van Hoa on my initial draft, which have led to a change in this version.

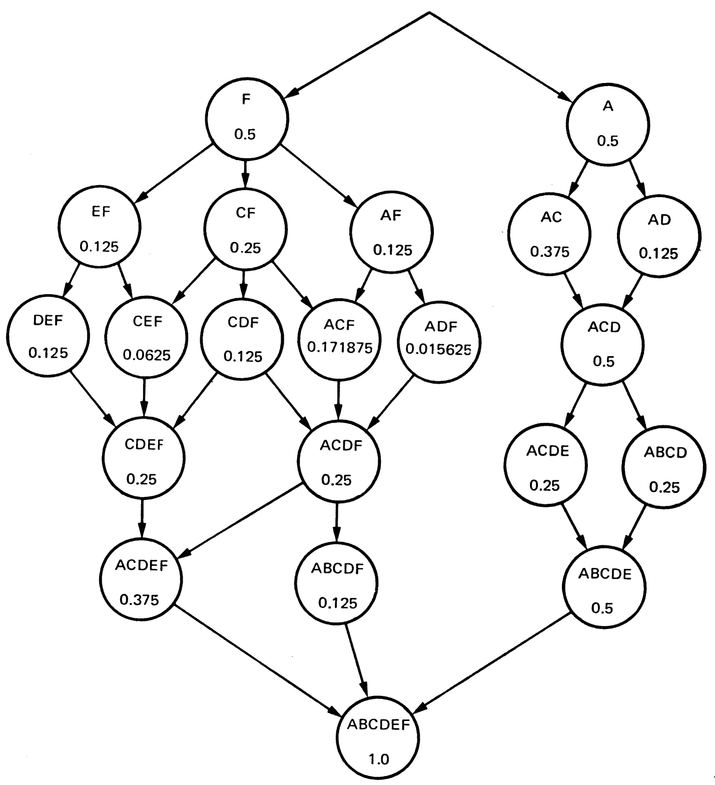

Fig. 1 Possible results from Algorithm 78, with probability at each node

References

Hill, I. D. (1975). Algorithm 88. *Computer J.*, Vol. 18, pp. 89-90.
Hill, I. D., and Wedderburn, R. W. M. (1974). Note on Algorithm 78. *Computer J.*, Vol. 17, p. 380.
Tran Van Hoa (1973). Algorithm 78. *Computer J.*, Vol. 16, pp. 273-276.

Note on Algorithm 81

DENDROGRAM PLOT

F. James Rohlf
Department of Ecology and Evolution
State University of New York at Stony Brook
Stony Brook, NY 11790 USA

A number of editorial misdemeanours occurred with Algorithm 81 (Vol. 17, No. 1) and the text of the corrected algorithm is now published again in full, with sincere apologies to the Author.

Editor

Author's note

Description

The results of hierarchic, nested, cluster analyses are usually shown in the form of a tree-like diagram called a dendrogram (see Sokal and Sneath, 1963 for a general account). In such a diagram (e.g. **Fig. 1**) the labels for the objects being clustered are plotted across the top and the clustering (or 'splitting levels') are shown along the ordinate. Clusters can be found for any threshold level, *h*, by drawing a horizontal line across the figure at a level corresponding to *h* on the ordinate. Each vertical line in the dendrogram cut by this horizontal line corresponds to a cluster whose members are the objects connected to that fragment of the dendrogram. Rohlf (1973) gives a simple method of describing any dendrogram by a

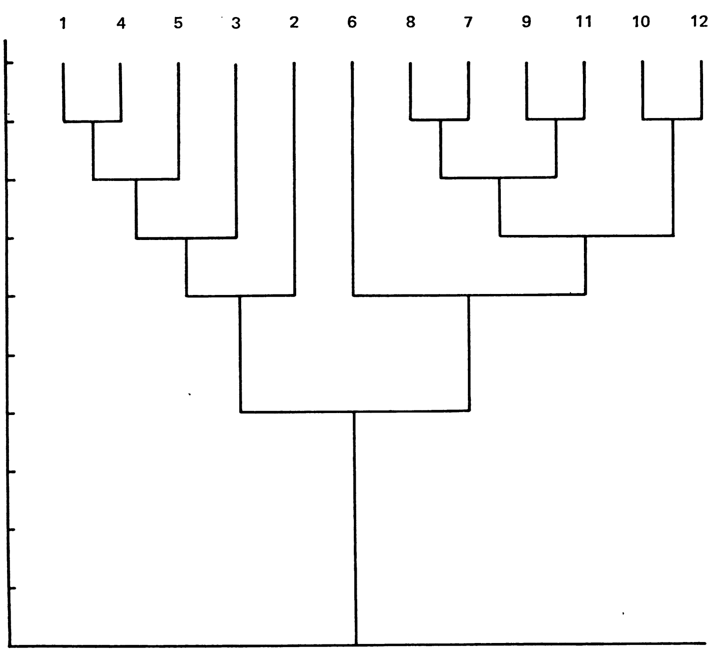

Fig. 1 Dendrogram for 12 objects

'tree matrix' which consists of two lists. One contains the labels of the n objects in the order in which they are to appear across the top of the dendrogram and the other contains $n - 1$ numbers which describe the branching patterns of the tree. If the dendrogram in Fig. 1 is plotted in the mathematically equivalent (but less aesthetically pleasing) manner shown in **Fig. 2** then one notices that there are n vertical lines (one for each object) and each (except the last) drops down from an object label until a certain point is reached and then turns to the left and continues until it intersects another vertical line. The list of the $n - 1$ heights of the 'bend' in each of these lines is sufficient to describe the dendrogram.

A diagram such as that given in Fig. 2 is easily computed for output onto a line printer (see Rohlf, 1973, for a simple program). For use by persons not familiar with cluster analysis, dendrograms in the form shown in Fig. 1 are much more desirable. The programs which I have examined seem unnecessarily complex and require moderate amounts of storage. The program by McCammon and Wenninger (1970) plots what he calls a 'dendrograph' (a dendrogram with the object labels at the tips of the dendrogram spaced proportional to the within-group dissimilarity). Most of the computations performed as objects are being clustered so that the final task of plotting is simple but it requires six arrays of length n (in addition to the label information). This program also has the disadvantage that it cannot be used to plot an existing dendrogram since most of the logic is embedded in the clustering program.

In the program by Bonham-Carter (1967) the dendrogram plotting is separated out from the clustering algorithm but it requires input in a less compact form. Each cycle in the associated clustering program (starting from the top of the dendrogram) stores code numbers for objects in the left and right branches and the y-coordinates for the horizontal line joining these two vertical lines. The plotting program then reads this tape and computes the four x and y-coordinates which describe these three line segments and then saves the x-coordinate of the centre of the horizontal line for processing at a later step.

The program TREE given in Anderberg (1972) requires eight arrays of length n to describe the tree and more than $26n$ storage locations for scratch arrays. The algorithm itself 'can be grasped intuitively through an example but is very difficult to understand from a formal statement of the pertinent operations' (Anderberg, 1972).

The algorithm given below is relatively simple and only requires three arrays of length n (in addition to an array which contains whatever labels are desired for the objects). One of these arrays (LEV) is the list of clustering levels described above.

The FORTRAN program given below is based on the following algorithm:

1. Input n, the number of objects, and the 'tree matrix' giving the labels for the objects, LAB, and the clustering levels LEV.
2. The plot is scaled to go from $x = 1$ to n and from $y = y_{\min}$ to $y_{\max}$. Set the variable $CURRY$ equal to a quantity less than $y_{\min}$ and set $LEV_n = y_{\min}$. In the account which follows we will assume that the scale along the ordinate goes from $y_{\min}$ corresponding to the least amount of similarity at the bottom of the diagram to $y_{\max}$ corresponding to the highest degree of similarity at the top. If dissimilarity coefficients are being analysed then the scale will have to be reversed and other obvious changes made.
3. Plot a vertical line for each of the $i = 1, 2, \ldots, n$ objects. The co-ordinates of the top of each line where the labels are plotted are $(i, y_{\max})$ and the co-ordinates of the bottoms are $(i, \{\max LEV_i, LEV_{i-1}\})$. For $i = 1$ the y co-ordinate is always LEV_1. The co-ordinates of the bottom end of these lines are also stored into arrays XC and YC.
4. Find the first $i (i = 1, 2, \ldots, n - 1)$ for which $LEV_i \geq LEV_{i+1}$.
5. Plot a horizontal line from (XC_i, YC_i) to (XC_{i+1}, YC_{i+1}). Plot a vertical line from the centre of this line down to $\max \{LEV_i, LEV_{i+1}\}$ (for $i = 1$ always use LEV_1). These steps symbolise the process of two clusters being merged and replaced by a new one. Store the co-ordinates of the bottom of this last line in XC_{i+1} and YC_{i+1}, delete the ith entries in LEV, XC, and YC and then close up.
6. Set $n = n - 1$. If $n > 1$ go to step 4, otherwise STOP.

Step 3 is based on the observation that if object i is the left-most number of a cluster then LEV_{i-1} will be less than LEV_i and the vertical line coming down from the label of object i ends at level LEV_i. Otherwise the vertical line ends at LEV_{i-1}.

Steps 4 and 5 are based on the fact that the ith and $(i + 1)$th pair of vertical lines can be connected and replaced by a single vertical line only if they are the left-most pair of objects in a cluster. This is true only if $LEV_{i-1} < LEV_i > LEV_{i+1}$ (LEV_0 and LEV_{n+1} are defined to be less than $y_{\min}$). The particular order in which this iterative search is made takes into account the fact that dendrograms tend to be asymmetrical as shown in the examples. This speeds up the deletion and compaction process on arrays LEV, XC, and YC in which entries (if any) to the left of entry i are moved one position to the right.

The computational effort varies from being proportional to n for dendrograms (like the left-half of Fig. 1) to being proportional to n^2 (for dendrograms the mirror image of the left half of Fig. 2).

To print a dendrogram sideways on a line printer one need only store the x-co-ordinates and the upper and lower y-co-ordinates of each vertical line as well as a code to indicate if the line branches from the left or right end of a horizontal line. These lists are then sorted so that the x-co-ordinates go from low to high. Then if the

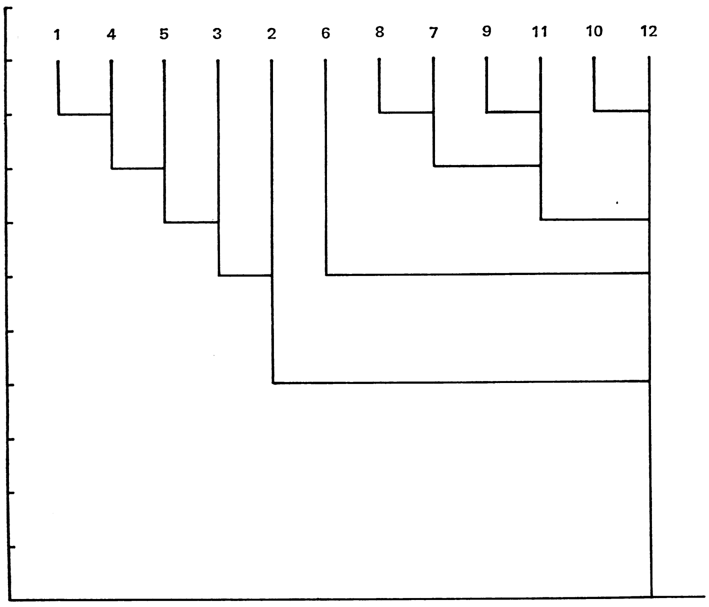

Fig. 2 Dendrogram of Fig. 1 replotted to show that only n – 1 level numbers are needed to describe branching form of tree

x-co-ordinates are coded into line numbers (going from 1 to $2n - 1$ and the y-co-ordinates are coded into positions in an array used to output the alphanumeric characters for printing a line on the line printer, the following steps will print the dendrogram.

1. Clear the output array to blanks.
2. Set line number equal to 1.
3. For *all* entries in the lists having x-co-ordinates equal to the current line number, store an '*' from the position corresponding to upper y-co-ordinate through to the lower y-co-ordinate in the array.
4. Print the output array (also output an object label if the current line number is an odd number).
5. For *all* entries in the lists having x-co-ordinates equal to the current line number, clear the output array to blanks from positions corresponding to the upper y-co-ordinate through to the lower y-co-ordinate -1. If this entry corresponds to a right branch in the dendrogram, then also clear the position corresponding to the lower y-co-ordinate.
6. If the current line number is less than $2n - 1$, then add 1 to the line number and go to step 3, otherwise STOP.

The horizontal lines in the plotted dendrogram are printed vertically by the simple device of not clearing the position in the output array corresponding to the end of a *left* branch in the dendrogram. Since it is not cleared it will appear in subsequent lines until a right branch is printed.

Since plotting routines are not standardised the FORTRAN program given below includes calls to hypothetical plotting routines for which the user will have to substitute locally available equivalents. This should prove to be no problem. Subroutine SCALE ($X_{\min}$, $X_{\max}$, $Y_{\min}$, $Y_{\max}$) does whatever is necessary to set up the transformation from problem units to device dependent units. Subroutine *MOVE* (X, Y) raises the pen and moves it to a position corresponding to co-ordinates X, Y in problem units. Subroutine *DRAW* (X, Y) lowers the pen and draws a straight line to co-ordinates X, Y in problem units. Subroutine *LABEL* (X, Y, *ALPHA*) does whatever is necessary to output the alphanumeric object labels with the first character plotted at co-ordinates X, Y in problem units. The FORTRAN program has been tested on an IBM 370/155 using both levels G and H FORTRAN IV and on a PDP-10. The method has also been implemented on a Hewlett-Packard Model 9820 desk top calculator which has a plotter.

Acknowledgements

This work was supported in part by a grant (B035234) from The National Science Foundation. This paper is contribution number 82 of the Program in Ecology and Evolution at the State University of New York at Stony Brook. The reviewer suggested an improvement to the program which increased the efficiency of the pen movement.

References

ANDERBERG, M. R. (1972). *Cluster analysis for applications*, Technical report, OAS-TR-72-1, Kirtland Air Force Base, New Mexico, USA, 514 pp.

BONHAM-CARTER, G. F. (1967). FORTRAN IV program for Q-mode cluster analysis of nonquantitative data using IBM 7090/7094 computers, *Computer Contribution* 17, State Geological Survey, University of Kansas. 28 pages.

MCCAMMON, R. B., and WENNINGER, G. (1970). The dendrograph, *Computer Contribution* 48, State Geological Survey, University of Kansas. 28 pages.

ROHLF, F. J. (1973). Hierarchical clustering using the minimum spanning tree. *Computer Journal*, Vol. 16, pp. 93-95.

SOKAL, R. R., and SNEATH, P. H. A. (1963). *Principles of Numerical Taxonomy*, W. H. Freeman and Co., San Francisco, 359 pages.

```
      SUBROUTINE SYMDEN(N, LAB, LEV, XC, YC, YMIN, YMAX)
C  PROGRAM TO PLOT A DENDROGRAM WITH CENTERED STEMS
C  N      = NUMBER OF OBJECTS
C  LAB    = LIST OF OBJECT LABELS (OF LENGTH N)
C  LEV    = LIST OF CLUSTERING LEVELS (N - 1 NUMBERS BUT
C           LEV(N) IS USED)
C  **NOTE = LEV IS DESTROYED BY THE PROGRAM
C  XC     = SCRATCH ARRAY TO HOLD X-COORDINATES
C  YC     = SCRATCH ARRAY TO HOLD Y-COORDINATES
C  YMIN   = VALUE OF VARIABLE CORRESPONDING TO THE
C           LEAST SIMILARITY
C  YMAX   = VALUE OF VARIABLE CORRESPONDING TO THE
C           MAXIMUM SIMILARITY
      REAL LAB(N), LEV(N), XC(N), YC(N)
      CALL SCALE(0., FLOAT(N + 1), 0., 1.)
      LEV(N) = YMIN
      YRANGE = YMAX - YMIN
      YHIGH = -1.
C  FOR EACH OBJECT DRAW A VERTICAL LINE AND LABEL
      DO 100 IC = 1, N
        C = IC
        Y = LEV(IC)
C  CODE Y INTO RANGE 0 TO 1
        Y = (Y - YMIN)/YRANGE
        LEV(IC) = Y
C  HEIGHT = CURRENT LEV OR PREVIOUS (WHICHEVER IS THE
C  LARGEST)
        IF(Y.GT.YHIGH) YHIGH = Y
        CALL MOVE(C, YHIGH)
        CALL DRAW(C, 1.)
        CALL LABEL(C, 1.01, LAB(IC))
        XC(IC) = C
        YC(IC) = YHIGH
        YHIGH = Y
  100 CONTINUE
C  DRAW THE REST OF THE DENDROGRAM
      ISTART = 1
      X = 0.
  150 NM1 = N - 1
      DO 200 I = ISTART, NM1
        IC = I
        IF(LEV(I).GE.LEV(I + 1)) GO TO 230
  200 CONTINUE
  230 Y = YC(IC)
C  DRAW HORIZONTAL LINE
      IF(X.EQ.XC(IC + 1)) GO TO 240
      CALL MOVE(XC(IC), Y)
      CALL DRAW(XC(IC + 1), Y)
      GO TO 250
  240 CALL DRAW(XC(IC), Y)
C  DRAW VERTICAL LINE AT CENTER OF HORIZONTAL LINE
  250 X = (XC(IC) + XC(IC + 1))/2.
      XC(IC + 1) = X
      CALL MOVE(X, Y)
      Y = LEV(IC + 1)
      IF(IC.GT.ISTART.AND.LEV(IC - 1).GT.Y)Y = LEV(IC - 1)
      YC(IC + 1) = Y
      CALL DRAW(X, Y)
C  DELETE ENTRY IC AND CLOSE UP SPACE
      ISTART = ISTART + 1
      IF(IC.LT.ISTART) GO TO 350
      DO 300 I = ISTART, IC
        II = IC - I + ISTART
        LEV(II) = LEV(II - 1)
        XC(II) = XC(II - 1)
        YC(II) = YC(II - 1)
  300 CONTINUE
C  LOOP BACK UP IF NOT DONE
  350 IF(ISTART.LE.NM1) GO TO 150
      RETURN
      END
```

Contributions should be addressed to:
R. F. Shepherd, Editor
Algorithms Supplement
Computing Centre
Chelsea College
University of London
Pulton Place
London SW6 5PR