Algorithms

J. H. WEGSTEIN, Editor

ALGORITHM 74
CURVE FITTING WITH CONSTRAINTS

J. E. L. Peck,
University of Alberta, Calgary, Alberta, Canada

```
procedure  Curve fitting (k,a,b,m,x,y,w,n,alpha,beta,s,sgmsq,x0,
  gamma,c,z,r);
comment  This procedure finds, by the method of least squares,
  the polynomial of degree n,  k ≤ n < k+m, whose graph con-
  tains (a1, b1), ··· , (akbk) and approximates (x1, y1), ··· ,
  (xm, ym), where wi is the weight attached to the point (xi, yi).
  The details will be found in the reference cited below, where a
  similar notation is used. A nonlocal label "error" is assumed;
  value a, x, y, w;  integer k, m, n, r;  real x0, gamma;  array
    a, b, x, y, w, alpha, beta, s, sgmsq, c, z;
  begin integer i, j;  array w1[1:k];  real p, f, lambda;
  comment  We shall first define several procedures to be used
    in the main program, which begins at the label START;

procedure  Evalue (x, nu);
comment  This procedure evaluates f = s0p0 + s1p1 + ··· +
  sνpν , where p−1(x) = 0,  p0(x) = 1,  β0 = 0  and  pi+1(x)
  = (x − αi)pi(x) − βipi−1(x),  i = 0, 1, ··· , ν−1. The value of
  pν(x) remains in p;
  real x; integer nu;
  begin real p0, temp;  integer i;  p0 := 0;  p := 1;  f := s[0];
for i := 0 step 1 until nu−1 do
  begin temp := p;
  p := (x−alpha[i]) × p−beta[i] × p0;
  p0 := temp;  f := f + p × s[i+1] end i
end Evalue;

procedure  Coda (n, c);
comment  This procedure finds the c's when c0 + c1x + ··· +
  cnx^n = s0p0(x) + ··· + snpn(x);
  integer n;  array c;
  begin integer i,r;  real t1,t2;  array pm,p[0:n];
for r := 1 step 1 until n do
  c[r] := pm[r] := p[r] := 0;
pm[0] := 0;  p[0] := 1;  c[0] := s[0];
for i := 0 step 1 until n−1 do
  begin t2 := 0;
  for r := 0 step 1 until i+1 do
    begin t1 := (t2−alpha[i] × p[r]−beta[i] × pm[r])/lambda;
    t2 := pm[r] := p[r];  p[r] := t1;
    c[r] := c[r] + t1 × s[i+1]end r
  end i
end Coda;

procedure  GEFYT (n,n0,x,y,w,m);
comment  This is the heart of the main program. It computes
  the αi,βi,si,σi², using the method of orthogonal polynomials, as
  described in the reference;
  integer n,n0,m;  array x,y,w;
  begin real dsq,wpp,wpp0,wxpp,wyp,temp;
integer i,j,freedom;  array p,p0[1:m];  boolean exact;
if n−n0 > m ∨ n < n0 then go to error;
beta[n0] := dsq := wpp := 0;  exact := n−n0 ≥ m−1;
for j := 1 step 1 until m do
  begin p[j] := 1;  p0[j] := 0;  wpp := wpp + w[j];
  if ¬ exact then dsq := dsq + w[j] × y[j] × y[j] end initialise;
for i := n0 step 1 until n do
  begin freedom := m−1−(i−n0);  wyp := wxpp := 0;
  for j := 1 step 1 until m do
    begin temp := w[j] × p[j];
    if i < n then wxpp := wxpp + temp × x[j] × p[j];
    if freedom ≥ 0 then wyp := wyp + temp × y[j] end j;
  if freedom ≥ 0 then s[i] := wyp/wpp;
  if ¬ exact then begin dsq := dsq − s[i] × s[i] × wpp;
  sgmsq[i] := dsq/freedom end if;
  if i < n then begin alpha[i] := wxpp/wpp;  wpp0 := wpp;
  wpp := 0;
  for j := 1 step 1 until m do
    begin temp := (x[j]−alpha[i]) × p[j] − beta[i] × p0[j];
    wpp := wpp + w[j] × temp × temp;
    p0[j] := p[j]; p[j] := temp end j;
  beta[i+1] := wpp/wpp0 end if
  end i
end GEFYT;
    START:  for j := 1 step 1 until k do
begin w1[j] := 1;  a[j] = (a[j]−x0)/gamma end j;
GEFYT (k,0,a,b,w1,k);
comment  This finds the polynomial of degree k−1 whose graph
  contains (a1,b1),···,(ak,bk) supplying the αi,βi,si,  0 ≤ i ≤ k;
  begin real rho;  rho := 0;
for j := 1 step 1 until m do
  begin rho := rho + w[j];
  x[j] := (x[j] − x0)/gamma end j;  rho := m/rho;
comment  The factor ρ is used to normalize the weights. We shall
  now put sk = 0 in order to evaluate pk(x) and the polynomial of
  degree k−1 simultaneously;
s[k] := 0;
for j := 1 step 1 until m do
  begin Evalue (x[j],k);
  if p = 0 then go to error;
  y[j] := (y[j] − f)/p;
  w[j] := w[j] × p × p × rho end j
end rho;
comment  We have now normalized the weights and adjusted
  the weights and ordinates ready for the least squares approxi-
  mation;
GEFYT (n,k,x,y,w,m);
comment  The coefficients αi,βi,  0 ≤ i < n, and si,  0 ≤ i ≤ n
  are now ready. The polynomial may be evaluated for x = z1,z2,
  ···,zr, but the variable must be adjusted first. Note that we
  may evaluate the best polynomial of lower degree by decreas-
  ing n;
  begin real x;
for j := 1 step 1 until r do
  begin x := (z[j]−x0)/gamma;
  Evalue (x,n);  comment the values of zj and f should now be
    printed;  end j;
comment  We may now adjust the coefficients for scale and then
  find the coefficients of the power series c0 + c1x + ··· + cnx^n =
  s0p0(x) + ··· + snpn(x);
for i := 0 step 1 until n−1 do
  begin alpha[i] := alpha[i] × gamma + x0;
  beta[i] := beta[i] × gamma end i;  lambda := gamma;
Coda (n,c);
comment  We may now re-evaluate the polynomial from the
  power series;
for j := 1 step 1 until r do
  begin x := z[j];  f := c[n];
  for i := n−1 step −1 until 0 do
    f := f × x + c[i];
  comment  the values of x and f should now be printed;  end j
  end x
end Curve fitting
```

Reference: Peck, J. E. L. Polynomial curve fitting with constraint, *Soc. Indust. Appl. Math. Rev.* (1961).

ALGORITHM 75
FACTORS

J. E. L. Peck,
University of Alberta, Calgary, Alberta, Canada

```
procedure factors (n,a,u,v,r,c);
comment This procedure finds all the rational linear factors of
  the polynomial a₀xⁿ + a₁xⁿ⁻¹ + ··· + aₙ₋₁x + aₙ, with integral
  coefficients. An absolute value procedure abs is assumed;
value n,a; integer r,n,c; integer array a,u,v;
begin comment We find whether p divides a₀, 1 ≤ p ≤ |a₀| and
  q divides aₙ, 0 ≤ q ≤ |aₙ|. If this is the case we try (px ± q);
integer p,q,a0,an;
r := 0; c := 1; comment r will be the number of linear factors
  and c the common constant factor;
TRY AGAIN: a0 := a[0]; an := a[n];
for p := 1 step 1 until abs(a0) do
  begin if (a0 ÷ p) × p = a0 then
    begin comment p divides a₀;
    for q := 0 step 1 until abs(an) do
      begin if q = 0 ∨ (an ÷ q) × q = an then
        begin comment q divides aₙ (or q = 0). If p = q we
          may have a common constant factor, therefore; if q
          > 1 ∧ p = 1 then
          begin integer j;
          for j := 1 step 1 until n−1 do
            if (a[j] ÷ q) × q ≠ a[j] then go to NO CONSTANT;
          for j := 0 step 1 until n do
            a[j] := a[j]/q;
          c := c × q; go to TRY AGAIN
          end the search for a common constant factor;
        NO CONSTANT:
          begin comment try (px − q) as a factor;
          integer f,g,i; f := a0; g := 1;
          comment we try x = q/p;
          for i := 1 step 1 until n do
            begin g := g × p; f := f × q + a[i] × g
            end evaluation;
          if f = 0 then
            begin comment we have found the factor (px − q);
            r := r + 1; u[r] := p; v[r] := q;
            comment there are now r linear factors;
              begin comment we divide by (px − q);
              integer i,t; t := 0;
              for i := 0 step 1 until n do
                begin a[i] := t := (a[i] + t)/p; t := t × q
                end i;
              n := n − 1
              end reduction of polynomial. Therefore;
            go to if n = 0 then REDUCED else TRY AGAIN
            end discovery of px − q as a factor. But
          if we got this far it was not a factor so try px + q;
          q := −q; if q < 0 then go to NO CONSTANT
          end trial of px ± q,
        end q divides aₙ and
      end of q loop.
    end p divides a₀, also
  end p loop, which means;
REDUCED: if n = 0 then
  begin c := c × a0; a0 := 1
  end if n = 0
end factors procedure. There are now r (r > 0) rational linear
  factors (uᵢx − vᵢ), 1 < i < r, and the reduced polynomial of
  reduced degree n replaces the original. The common constant
  factor is c. Acknowledgments to Clay Perry.
```

ALGORITHM 76
SORTING PROCEDURES

Ivan Flores
Private Consultant, Norwalk, Connecticut

comment The following ALGOL 60 algorithms are procedures for the sorting of records stored within the memory of the computer. These procedures are described in detail, flow-charted, compared, and contrasted in "Analysis of Internal Computer Sorting" by Ivan Flores [*J. ACM 8* (Jan. 1961)]. Although sorting is usually a business computer application, it can be described completely in ALGOL if we stretch our imagination a little. Sorting is ordering with respect to a key contained within the record. If the key *is* the active record, the sorting is trivial. A means is required to extract the key from the record. This is essentially string manipulation, for which no provision, as yet, has been made in ALGOL. We circumambulate this difficulty by defining an **integer procedure** K(I) which "creates" a key from the record, I. ALGOL *does* provide for machine language code substitutions, which is one way to think of K(I). This could be more accurately represented by using the string notation proposed by Julien Green ["Remarks on ALGOL and Symbol Manipulation," *Comm. ACM 2* (Sept. 1959), 25–27]. The function **sub** ($,i,g) represents the procedure, K(I). $ corresponds to the record I, i corresponds to the starting position of the key and g corresponds to the length of the key. Both i and g are **values** which must be specified when the sort procedure is called for as a statement instead of a declaration.

Another factor, which might vex some, is that the key might be alphabetic instead of numeric. Then, of course, K(I) would not be integer. It would, however, be string when such is defined eventually. Note, also, that keys are frequently compared. This is done using the ordering relations ">" for "greater than," etc. These are not really defined in the ALGOL statement [NAUR, PETER, ET AL. "Report on the Algorithmic Language ALGOL 60". *Comm. ACM 3* (May 1960), 294–314]. They can simply be defined so that Z > Y > ··· > A > 9 > ··· > 1 > 0. Also the assignment X[i] := z should be interpreted as "Assign the key 'z' which is larger than any other key." For any sort procedure (I,N,S), "I" is the set of unsorted records, "N" is their number, and "S" the sorted set of records.

Caution, these algorithms were developed purely for the love of it: No one was available with the combined knowledge of sorting and ALGOL to check this work. Hence each algorithm should be carefully checked before use. I will be glad to answer any questions which may arise;

```
Sort insert (I,N,S); value N; array I[1:N], S[1:N];
  integer procedure K(I); integer N;
  begin integer i, j, k;
    S[1] := I[1];
    for i := 2 step 1 until N do begin
      for j := i − 1, j − 1 while K(I[i]) > K(S[j]) do
      for k := i step − 1 until j + 1 do
        S[k] := S[K − 1];
      S[j + 1] := I[i] end end
```

```
Sort count (I,N,S); value N; array I[1:N], S[1:N];
  integer procedure K(I); integer N;
  begin integer array C[1:N]; integer i,j;
    for i := 1 step 1 until N do C[i] := 0;
    for i := 2 step 1 until N do
      for j := 1 step 1 until i − 1 do
        if K(I[i]) > K(I[j]) then C[i] := C[i] + 1
        else C[j] := C[j] + 1;
    for i := 1 step 1 until N do
      S[C[i]] := I[i] end
```

```
Sort select (I,N,S); value N;  array I[1:N], S[1:N];
  integer procedure K(I);  integer N;
  begin integer i,j,A,h;
    for i := 1 step 1 until N do begin
    h := K(I[1]);
    for j := 2 step 1 until N do
    if h > K(I[j]) then begin h := K(I[j]);  A := j end;
    S[i] := I[A];
    I[A] := z end end
Sort select exchange (I,N);  value N;  array I[1:N];
  integer procedure K(I);  integer N;
  begin integer h,i,j,H;  real T;
    for i := 1 step 1 until N do begin
      H := K(I[i]);  h := i;
      for j := i + 1 step 1 until N do
        if K(I[j]) < H then begin
        H := K(I[j]);  h := j end
      T := I[i];  I[i] := I[h];  I[A] := T end
  end
Sort binary insert (I,N,S);  value N;  array I[1:N], S[1:N];
  integer procedure K(I);  integer N;
  begin integer i,k,j,l;
    if K(I[1]) < K(I[2]) then begin
      S[1] := I[1];  S[2] := I[2] end
      else begin S[1] := I[2];  S[2] := I[1] end;
  start:  for   i := 3 step 1 until N do begin
                j := (i + 1) ÷ 2;
  find spot:    for k := (i + 1) ÷ 2, (k + 1) ÷ 2 while k > 1 do
                  if K(I[i]) < K(S[j]) then j := j - k
                  else j := j + k;
                if K(I[i]) ≥ K(S[j]) then j := j - 1;
  move items:   for l := i step - 1 until j do
                  S[l + 1] := S[l];
  enter this
  one:          S[j] := I[i] end end
Sort address calculation (I,N,S,F);  value N;
  array S[1:M], I[1:N];  integer procedure F(K), K(I);
  integer N,M;
             begin integer i,j,G,H,F,M;
               M := entier(2.5 × N)
               for i := 1 step 1 until M do S[i] = 0;
 Address:      for i := 1 step 1 until N do begin
               F := F(K(I[i]));
               if S[F] = 0 then begin S[F] := I[i];
                 go to NEXT end
               else if K(S[F]) > K(I[i]) then go to SMALLER;
 LARGER:       for H := F, H + 1 while K(S[H]) < K(I[i]) do
               for G := H, G + 1 while K(S[G]) ≠ 0 do
               for j := G step -1 until H + 1 do
                 S[j] := S[j - 1];
               S[H] := I[i];  go to NEXT;
 SMALLER:      for H := F, H - 1 while K(S[H]) > K(I[i]) do
               for G := H, G - 1 while K(S[G]) ≠ 0 do
               for j := G step 1 until H - 1 do
                 S[j] = S[j + 1];
               S[H] := I[i];
 NEXT:       end end
Sort quadratic select (I,N,S);  value N;  array I[1:N], S[1:N];
  integer procedure K(I);  integer N;
                 begin integer i,j,k,C,D,J,M;
                   integer array C[1:M], D[1:M];
                   array I[1:M, 1:M];
  Divide inputs:  M := entier (sqrt (N)) + 1;  j := k := 1;
                  for i := 1 step 1 until N do begin
                    I[j,k] := I[i];  k := k + 1;
                    if k > M then begin k := 1;
                      j := j + 1 end end
Fill up inputs:  I[j,k] := z;  k := k + 1;
                 if k > M then begin k := 1;  j := j + 1 end
                 if j ≤ M then go to Fill up inputs;
Set controls:    for j := 1 step 1 until M do begin
                 C[j] := K(I[j, 1]);  D[j] := 1;
                 for k = 2 step 1 until M do
                   if C[j] > K(I[j,k]) then begin
                   C[j] := K(I[j,k]);  D[j] := k end end;
                 i := 1;
Find least:      C := C[1];  D := D[1];  J := 1;
                 for j := 2 step 1 until M do
                   if C > C[j] then begin C := C[j];
                     D := D[j];  J := j end;
Fill file:       S[i] := I[J,D];  i := i + 1;  I[J,D] := z;
                 if i = N + 1 go to STOP;
Reset controls:  for j := J do begin
                 C[j] := K(I[j, 1]);  D[j] := 1;
                 for k := 2 step 1 until M do
                   if C[j] > K(I[j,k]) then begin C[j] :=
                     K(I[j,k];  D[j] := k end end;
                 go to Find least;
STOP:            end
Presort quadratic selection (I,N,S);  value N;
  array I[1:N], S[1:N];  integer procedure K(I);  integer N;
                 begin integer i,j,k,C,J,M;
                   integer array C[1:M], D[1:M];
                     array I[1:M,1:M];
  Divide inputs:  M := entier (sqrt(N)) + 1;  j := k := 1;
                  for i := 1 step 1 until N do begin
                    I[j,k] := I[i];  k := k + 1;
                    if k > M then begin k := 1;
                      j := j + 1 end end
  Fill up inputs: I[j,k] := z;  k := k + 1;
                  if k > M then begin k := 1;  j = j + 1 end
                  if j ≤ M then go to Fill up inputs;
  First sort:     for j := 1 step 1 until M do
                  sort select exchange (I[j,k],M);
  Set controls:   for j := 1 step 1 until M do begin
                    C[j] := K(I[j,1]);  D[j] := 1 end
                  i := 1;
  Find least:     C := C[1];  J := 1;
                  for j := 1 step 1 until M do
                    if C > C[j] then begin C := C[j];
                      J := j end;
  Fill file:      S[i] := I[J,D[J]];  i := i + 1;
                  if i = N + 1 go to STOP
  Reset control:  for j := J do begin
                    D[j] := D[j] + 1;
                    if D[j] > M then C[j] := z else C[j] :=
                      K(I[j, D[j]]) end
                  go to Find least;
  STOP:           end
Sort binary merge (I,N,S);  value N; array I[1:N];
  integer procedure K(I);  integer N;
  begin real array S[1:N];
  integer array A[0:1, 0:J[a]], B[0:1, 0:K[b]], Aloc[0:1, 0:J[a]],
    Bloc[0:1, 0:K[b]], J[0:1], K[0:1], j[0:1], k[0:1];
    integer a,b,i,j,k;
  distribute:        a := b := j[0] := j[1] := 1;
                     for i := 1 step 1 until N do begin
                       if K(I[i]) < K(I[i-1] then
                         if a = 1 then a := 0 else a := 1;
                       A[a, j[a]] := K(I[i]);  Aloc[a, j[a]] := i;
                         j[a] := j[a] + 1 end;
                     J[0] := j[0];  J[1] := j[1];
  next sort:         begin a := b := j[0] := j[1] := k[0] :=
                       k[1] := 1;
```

```
two inputs:        if A[1, j[1]] ≤ A[0, j[0]] then a := 1 else
                     a := 0;
                   B[b, k[b]] := A[a, j[a]];
                     Bloc[b, k[b]] := Aloc[a, j[a]];
                   j[a] := j[a] + 1;  k[b] := k[b] + 1;
                   if A[a, j[a]] ≥ A[a, j[a] − 1] then go to two
                     inputs else
                   if a = 1 then a := 0 else a := 1;
single step:       B[b, k[b]] := A[a, j[a]];
                     Bloc[b, k[b]] := Aloc[a, j[a]];
                   j[a] := j[a] +1;  k[b] := k[b] + 1;
                   if A[a, j[a]] ≥ A[a, j[a] − 1] then go to
                     single step;
switch file:       if b = 1 then b := 0 else b := 1;
check rollout:     for a := 0, 1 do
                     if j[a] = J[a] then go to rollout;
                   go to two inputs;
rollout:           B[b, k[b]] := A[a, j[a]];
                     Bloc[b, k[b]] := Aloc [a, j[a]];
                   k[b] := k[b] + 1;  j[a] := j[a] + 1;
                   if j[a] = J[a] then go to interchange files;
                   if A[a, j[a]] < A[a, j[a] − 1] then
                     if b = 1 then b := 0 else b := 1;
                   go to rollout;
interchange files: K[0] := k[0];  K[1] := k[1];
                   if K[0] = 1 then go to output end
                   for b := 1, 0 do begin
                     for k[b] := 1 step 1 until K[b] do begin
                       A[b, k[b]] := B[b, k[b]];
                         Aloc[b, k[b]] := Bloc[b, k[b]];
                       J[b] := K[b] end end
                   go to next sort;
output:            for i := 1 step 1 until N do
                       S[i] := I[Bloc[0, i]];
                   end
```

CERTIFICATION OF ALGORITHM 30
NUMERICAL SOLUTION OF THE POLYNOMIAL EQUATION [K. W. Ellenberger, *Comm. ACM 3* (Dec. 1960), as corrected in the previous Certification by William J. Alexander, *Comm. ACM 4* (May 1961)]

Kalman J. Cohen
Graduate School of Industrial Administration, Carnegie Institute of Technology, Pittsburgh, Pa.

The ROOTPOL procedure originally published by Ellenberger as corrected and modified by Alexander was coded for the Bendix G20 in 20-GATE. Some serious errors were found in the third and fourth lines above the statement labelled "REVERSE" in Ellenberger's Algorithm which were not mentioned in Alexander's Certification. First, the function "log" is not a standard function in Algol 60; it is clear from the context, however, that Ellenberger intends this to be the logarithm function to the base 10. Second, Ellenberger's Algorithm failed to divide the accumulated sum of the logarithms by n+1 before taking the antilogarithm.

The correct, and slightly simplified, manner in which the third and fourth lines above the statement labelled "REVERSE" should read is:

```
if h_j ≠ 0 then s := ln(abs(h_j))
end;  s := s/(n+1);  s := exp(s);
```

With these corrections, the numerical results obtained essentially agree with those reported by Alexander.

CERTIFICATION OF ALGORITHM 50
INVERSE OF A FINITE SEGMENT OF THE HILBERT MATRIX [J. R. Herndon, *Comm. ACM 4* (Apr. 1961)]

B. Randell
Atomic Power Division, The English Electric Co., Whetstone, England

INVHILBERT was translated using the Deuce Algol compiler and the following corrections being needed.

1. S[1, 1] = n × n, replaced by S[1, 1] := n × n;
2. S[j, i] := S(j, 1]/(i + j − 1)
replaced by S[j, i] := S[j, i]/(i + j − 1)

The compiled program, which used a 20 bit mantissa floating point notation then produced the following 4 × 4 segment

16.0	−120.0	240.0002	−140.0
−120.0	1200.0	−2700.0	1680.0019
240.0	−2700.0	6480.0	−4200.0
−140.0	1680.0019	−4200.0	2800.0039

CERTIFICATION OF ALGORITHM 66
INVRS (J. Caffrey, Comm. ACM. July 1961)

B. Randell, C. G. Broyden.
Atomic Power Division, The English Electric Company, Whetstone, England.

INVRS was translated using the Deuce Algol Compiler, and needed the following correction.

The repeat of the line,

begin pivot := 1.0/t[1, 1];

was deleted.

The compiled program, which used a 20 bit mantissa floating point notation, was tested using Wilson's matrix

5	7	6	5
7	10	8	7
6	8	10	9
5	7	9	10

and gave results

67.9982	−40.9991	−16.9995	9.9997
−40.9991	24.9995	9.9997	−5.9998
−16.9995	9.9997	4.9998	−2.9999
9.9997	−5.9998	−2.9999	1.9999

(The output routine completed the symmetric matrix)

INVRS will in fact invert non-positive symmetric matrices, the only restriction appearing to be that the leading minors of the matrix must be non-zero. The variable T[1, 1] takes as its successive values ratios of the (r + 1)th to the r th leadng minors of the matrix, and if it becomes zero the variable 'pivot' cannot be computed.

The following matrix, for which the successive values of T[1, 1] were +2, −2, −1, −0.6, +5 gave results correct to one unit in the fifth significant figure.

2	−3	1	−1	4
−3	2	−4	3	−2
1	−4	−3	2	4
−1	3	2	−2	−3
4	−2	4	−3	2

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