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 i := 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).

CERTIFICATION OF ALGORITHM 74
CURVE FITTING WITH CONSTRAINTS [J. E. Peck, *Comm. ACM*, Jan. 62]

KAZUO ISODA
Japan Atomic Energy Research Institute, Tokai, Ibaraki, Japan

Algorithm 74 was hand-compiled into SOAP IIa for the IBM 650 and run successfully with no corrections except the case in which the origin (0, 0) are given as both a constraint and a sample.