# Algorithms

J. G. HERRIOT, Editor

## ALGORITHM 295
## EXPONENTIAL CURVE FIT [E2]

H. Späth (Recd. 29 Apr. 1966)
Institut für Neutronenphysik und Reaktortechnik,
Kernforschungszentrum Karlsruhe, Germany

```
procedure expfit (x, y, p, n, ca, ce, eps, a, b, c, s, fx, exit);
  value n, ca, ce, eps;   integer n;   real ca, ce, eps, a, b, c, s;
  label exit;   array x, y, p, fx;
```

**comment** If the method of least squares is used to determine the parameters $a$, $b$, $c$ of a curve $f(x) = a + be^{-cx}$ which approximates $n$ data points $(x_i, y_i)$ with associated weights $p_i$, then

$$s(a, b, c) = \sum_{i=1}^{n} p_i(y_i - f(x_i))^2 \qquad \text{(I)}$$

must be a minimum. A necessary condition for this is that

$$\frac{\partial s}{\partial a} = \frac{\partial s}{\partial b} = \frac{\partial s}{\partial c} = 0. \qquad \text{(II)}$$

Usually (see [1]) it is attempted to solve this system of nonlinear equations by an iterative method which is based upon the linearization of $f$ in (II) and the convergence of which depends on the given starting values for $a$, $b$, $c$.

A simpler and more effective way which can always be chosen if there is only one nonlinear parameter in $f$ is the following: It is always possible to eliminate $a = a(c)$ and $b = b(c)$ from the equations $\partial s/\partial a = 0$ and $\partial s/\partial b = 0$ and to put these expressions into $\partial s/\partial c = 0$. This gives only one equation in one variable

$$F(c) := \frac{\partial s}{\partial c}(a(c), b(c), c) = 0.$$

If a value $c'$ is calculated with $F(c') = 0$ then the corresponding values of $a$ and $b$ are obtained from $a' = a(c')$ and $b' = b(c')$.

The following procedure is based upon this idea which is fully treated in [2]. It allows to find a triple $(a, b, c)$ which solves (II) if you make available a nonlocal procedure *Rootfinder* which is able to get a zero $c$ of a function $F(c)$ in the interval $[ca, ce]$ with the relative accuracy *eps*, if $sign(F(ca)) \neq sign(F(ce))$ otherwise leaving to the global label *exit*. As the above $F(c)$ is discontinuous at $c = 0$, $[ca, ce]$ must not contain 0. [The speed and efficiency of the algorithm depend on the choice of the procedure *Rootfinder*.—Ref.]

Most of the symbols are self-explanatory. The array *fx* finally contains the values $a + be^{-cx_i}$;

```
begin integer i;   real t, u, v, w, fc, h0, h1, h2, h3, h4, h5, h6, h7;
  procedure fronc (c, fc);   value c;   real c, fc;
  comment  computes for a given c the value fc = F(c) and
    a = a(c), b = b(c);
  begin h0 := h1 := h2 := h3 := h4 := h5 := h6 := h7 := 0;
    for i := 1 step 1 until n do
    begin
      t := x[i];   u := exp(−c×t);   v := p[i];   w := y[i];
      h0 := h0 + v;   h1 := h1 + u × v;   h2 := h2 + u × u × v;
      h3 := h3 + v × w;   h4 := h4 + u × v × w;
        h5 := h5 + t × u × v;
      h6 := h6 + t × u × u × v;   h7 := h7 − u × v × w × t
    end i;
    t := 1.0/(h0×h2−h1×h1);   a := t × (h2×h3−h1×h4);
    b := t × (h0×h4−h1×h3);   fc := h7 + (h5×a+h6×b)
  end fronc;
  Rootfinder (fronc, ca, ce, eps, c, exit);   t := 0;
  for i := 1 step 1 until n do
  begin
    v := fx[i] := a + b × exp(−c×x[i]);   v := v − y[i];
    t := t + p[i] × v × v
  end i;
  s := t
end expfit
```

References:

1. Deily, G. R. Algorithm 275, Exponential curve fit. *Comm. ACM 9* (Feb. 1966), 85.
2. Oberländer, S. Die Methode der kleinsten Quadrate bei einem dreiparametrigen Exponentialansatz. *ZAMM 43* (1963), 493–506.

## ALGORITHM 296
## GENERALIZED LEAST SQUARES FIT BY ORTHOGONAL POLYNOMIALS [E2]

G. J. Makinson (Recd. 30 Sept. 1965 and 29 Aug. 1966)
University of Liverpool, Liverpool 3, England

```
procedure LSFITUW (f, x, w, m, k, si, p, l, al, be, s);   value m, k;
  integer m, k;   array f, w, si, p, x, al, be, s;   Boolean l;
```

**comment** *LSFITUW* accepts $m$ observations $x[i]$, $f[i]$, $i = 1, 2, \cdots, m$ each with its associated weight $w[i]$. The weights should be provided inversely proportional to the standard error of the observations.

$x[1]$ should be algebraically the smallest abscissa and $x[m]$ the largest.

The coefficients of the best fitting polynomial of degree $k$ or less, where $k < m - 1$, are obtained in $p[0:k]$, with $p[0]$ the independent term. $si[0:k]$ contains the measures of the goodness of fit of each polynomial tested. The $si[t]$ are examined successively and the best polynomial is chosen of degree $h$ if $h$ is the first value of $t$ found such that $si[h] < si[h+1]$ provided that $si[j] > 0.6 \times si[h]$ for $k \geqq j > h + 1$. If $h$ is the first value of $t$ found such that $si[h] < si[h+1]$ but then a $j$ is found that satisfies $si[j] \leqq 0.6 \times si[h]$ for $j > h + 1$ the procedure will choose the polynomial of degree $j$ as best fit.

If an $h$ such that $si[h] < si[h+1]$ is not found then the polynomial is chosen of degree $k$. *LSFITUW* uses the procedure *POLYX* $(a, b, c, d, n)$ [Algorithm 29, *Comm. ACM 3* (Nov. 1960), 604] to transform its results from the interval $(-2,2)$ to the interval $(x[1], x[m])$.

Normally $l$ should be **false** but if the choice made is to be overruled after consideration of the *si* and the best fitting polynomial is required to be strictly of degree $k$, then $l$ should be **true**.

The programming is as outlined by G. E. Forsythe, [*J. Soc. Indust. Appl. Math. 5* (1957), 74–88] and originally programmed by J. G. Mackinney [Algorithm 28/29, *Comm. ACM 3* (Nov. 1960), 604]. *LSFITUW* incorporates remarks made by D. B. MacMillan [*Comm. ACM 4* (Dec. 1961), 544].

The variables in the paper of Forsythe have been abbreviated as follows.

*al*[*i*] is *alpha*[*i*], *be*[*i*] is *beta*[*i*], *si*[*i*] is $(sigma[i]) \uparrow 2$, *s*[*i*] is the same, *om* is *omega*, *lw* is $w[i, i]$, *tw* is $w[i+1, i+1]$, *ctp*[*j*] is the coefficient of $x \uparrow j$ in This (the current) orthogonal polynomial, *clp*[*j*] is the coefficient of $x \uparrow j$ in the Last (previous) orthogonal polynomial, *cp*[*j*] is the coefficient of $x \uparrow j$ in the most recently calculated polynomial of best fit, *tp*[*i*] is the value at *x*[*i*] of the present orthogonal

```
    polynomial, lp[i] is the value at x[i] of the last orthogo-
    nal polynomial, simin is the least value of (sigma[i]) ↑ 2
    found so far, swx becomes false as soon as (sigma[i+1]) ↑
    2 ≧ (sigma[i]) ↑ 2 one time, comp becomes true if swx
    is false and some (sigma[i]) ↑ 2 < 0.6 × simin;
egin integer i, j;  real du, delsq, om, lw, tw, simin, a, b;
 array clp, cpsave, cp[0:k], clp[-1:k], lp, tp[1:m];
  Boolean swx, comp;
 comment  initialization;
 for i := 0 step 1 until k do cp[i] := 0;  simin := 0;
 swx := true;  be[0] := clp[0] := clp[-1] := delsq := om := 0;
   ctp[0] := 1;  tw := 0;  comp := false;
 for i := 1 step 1 until m do
 begin
   delsq := delsq + w[i] × f[i] ↑ 2;  tp[i] := 1;
   lp[i] := 0;  om := om + w[i] × f[i];  tw := tw + w[i]
 end;
 s[0] := cp[0] := om/tw;  delsq := delsq - s[0] × om;
   si[0] := delsq/(m-1);
 comment  transformation of abscissa;
 a := 4/(x[m]-x[1]);  b := - 2 - a × x[1];
 for i := 1 step 1 until m do x[i] := a × x[i] + b;
 comment  main computation loop;
 for i := 0 step 1 until k - 1 do
 begin
   du := 0;
   for j := 1 step 1 until m do du := du + w[j] × x[j] × tp[j] ↑ 2;
   al[i + 1] := du/tw;  lw := tw;  tw := om := 0;
   for j := 1 step 1 until m do
   begin
     du := be[i] × lp[j];
     lp[j] := tp[j];
     tp[j] := (x[j]-al[i+1]) × tp[j] - du;
     tw := tw + w[j] × tp[j] ↑ 2;
     om := om + w[j] × f[j] × tp[j]
   end;
   be[i+1] := tw/lw;  s[i+1] := om/tw;
   delsq := delsq - s[i+1] × om;  si[i+1] := delsq/(m-i-2);
   if l then go to L1;
   if ¬ comp then
   begin
     if swx then
     begin
     if si[i+1] ≧ si[i] then
       begin
         comment higher power appears not to improve fit;
         swx := false;
         simin := si[i];
         for j := 0 step 1 until k do
           cpsave[j] := cp[j]
       end;
       go to L1
     end;
     if si[i+1] < 0.6 × simin then comp := true;
     comment  termination of main loop at superior fit to first
       one found;
     comment  recursion to obtain the coefficients cp of the
       polynomial of best fit of degree i + 1;
L1:  for j := 0 step 1 until i do
     begin
       du := clp[j] × be[i];
       clp[j] := ctp[j];
       ctp[j] := clp[j-1] - al[i+1] × ctp[j] - du;
       cp[j] := cp[j] + s[i+1] × ctp[j]
     end;
     cp[i+1] := s[i+1];  ctp[i+1] := 1;  clp[i+1] := 0;
     if ¬ (comp ∨ swx) then
       begin
         if i = k - 1 then
         for j := 0 step 1 until k do
           cp[j] := cpsave[j]
       end
     end
   end end of main computation loop. Transformation of poly-
     nomial follows;
   POLYX(a, b, cp, p, k)
 end LSFITUW
```

## Algorithms Policy • Revised August, 1966

### (Includes Fortran)

A contribution to the Algorithms Department should be in the form of an algorithm, a certification, or a remark. Contributions should be sent in duplicate to the editor, typewritten double spaced. Authors should carefully follow the style of this department with especial attention to indentation and completeness of references.

An algorithm must normally be written in the ALGOL 60 Reference Language [*Comm. ACM 6* (Jan. 1963), 1-17] or in ASA Standard FORTRAN or Basic FORTRAN [*Comm.* ACM 7 (Oct. 1964), 590-625]. Consideration will be given to algorithms written in other languages provided the language has been fully documented in the open literature and provided the author presents convincing arguments that his algorithm is best described in the chosen language and cannot be adequately described in either ALGOL 60 or FORTRAN.

An algorithm written in ALGOL 60 normally consists of a commented procedure declaration. It should be typewritten double spaced in capital and lower-case letters. Material to appear in **boldface** type should be underlined in black. Blue underlining may be used to indicate italic type, but this is usually best left to the Editor. An algorithm written in FORTRAN normally consists of a commented subprogram. It should be typewritten double spaced in the form normally used for FORTRAN or it should be in the form of a listing of a FORTRAN card deck together with a copy of the card deck. Each algorithm must be accompanied by a complete driver program in its language which generates test data, calls the procedure, and produces test answers. Moreover, selected previously obtained test answers should be given in comments in either the driver program or the algorithm. The driver program may be publishedwith the algorithm if it would be of major assistance to a user.

For ALGOL 60 programs, input and output should be achieved by procedure statements, using any of the following eleven procedures (whose body is not specified in ALGOL) [See "Report on Input-Output Procedures for ALGOL 60," *Comm. ACM 7* (Oct. 1964), 628-629]:

| | | | |
|---|---|---|---|
| *insymbol* | *inreal* | *outarray* | *ininteger* |
| *outsymbol* | *outreal* | *outboolean* | *outinteger* |
| *length* | *inarray* | *outstring* | |

If only one channel is used by the program for output, it should be designated by 1 and similarly a single input channel should be designated by 2. Examples:

```
outstring (1, 'x=');  outreal (1,x);
for i := 1 step 1 until n do outreal (1,A[i]);
ininteger (2, digit [17]);
```

For FORTRAN programs, input and output should be achieved as described in the ASA preliminary report on FORTRAN and Basic FORTRAN.

It is intended that each published algorithm be well organized, clearly commented, syntactically correct, and a substantial contribution to the literature of Algorithms. It is necessary but not sufficient that a published algorithm operate on some machine and give correct answers. It must also communicate a method to the reader in a clear and unambiguous manner. All contributions will be refereed both by human beings and by an appropriate compiler. Authors should pay considerable attention to the correctness of their programs, since referees cannot be expected to debug them.

Certifications and remarks should add new information to that already published. Readers are especially encouraged to test and certify previously uncertified algorithms. Rewritten versions of previously published algorithms will be refereed as new contributions and should not be imbedded in certifications or remarks.

Galley proofs will be sent to authors; obviously rapid and careful proofreading is of paramount importance.

Although each algorithm has been tested by its author, no liability is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm.—J.G.Herriot