Algorithms

J. G. HERRIOT, Editor

ALGORITHM 275
EXPONENTIAL CURVE FIT [E2]

GERARD R. DEILY (Recd. 27 July 1964 and 16 Apr. 1965)
US Department of Defense, Washington, D. C.
(Now with HRB-Singer, Inc., State College, Pa.)

```
procedure EXPCRVFT (a, b, c, E squared, n, x, y, epsilon, l max,
  flag);
  integer n, l max, flag;
  real a, b, c, E squared, epsilon;
  real array x, y;
comment This algorithm will fit a curve defined by the equation
  y = a × exp(b × x) + c to a set {x_i , y_i} of n data points. The
  Taylor series modification of the classical least squares method
  is utilized to approximate a solution to the system of nonlinear
  equations of condition. After every iteration, the statistic E
  squared is computed as a measure of the goodness of fit. Com-
  mencing with the second iteration, the successive values of E
  squared are differenced, and when the difference in absolute
  value becomes less than epsilon, the calculations cease. If the
  number of iterations necessary to achieve this result exceeds
  l max, a flag is set to a nonzero value and the procedure is termi-
  nated;
begin
  integer i, l, m;
  comment Computation of initial estimates follows;
  b := 2 × ln(abs(((y[n] − y[n−1]) × (x[2] − x[1]))/
                    ((y[2] − y[1]) × (x[n] − x[n−1]))))/
                    (x[n] + x[n−1] − x[2] − x[1]);
  a := (y[n] − y[n−1])/((x[n] − x[n−1])
    × exp((b × (x[n] + x[n−1]))/2) × b);
  m := (n+1) ÷ 2;
  c := y[m] − a × exp(b × x[m]);
  E squared := 0;
  for i := 1 step 1 until n do
    E squared := E squared + (y[i] − c − a × exp(b × x[i]))↑2;
  comment Computation of corrections follows;
  for l := 1 step 1 until l max do
  begin
    real sumex1, sumex2, sumxiex1, sumxiex2, sumxi2ex2, sumyi,
      sumyiex1, sumxyiex1, d11, d12, d13, d22, d23, d33, e1, e2, e3,
      delta11, delta12, delta13, delta22, delta23, delta33, delta, u, v, w,
      save;
    sumex1 := sumex2 := sumxiex1 := sumxiex2 := sumxi2ex2 :=
      sumyi := sumyiex1 := sumxyiex1 := 0;
    for i := 1 step 1 until n do
    begin
      real ex1, ex2, xiex1, xiex2, xi2ex2;
      ex1 := exp(b × x[i]);
      ex2 := ex1↑2;
      xiex1    := x[i] × ex1;
      xiex2    := x[i] × ex2;
      xi2ex2   := x[i] × xiex2;
      sumex1 := sumex1 + ex1;
      sumex2 := sumex2 + ex2;
      sumxiex1  := sumxiex1 + xiex1;
      sumxiex2  := sumxiex2 + xiex2;
      sumxi2ex2 := sumxi2ex2 + xi2ex2;
      sumyi      := sumyi + y[i];
      sumyiex1   := sumyiex1 + y[i] × ex1;
      sumxyiex1  := sumxyiex1 + y[i] × xiex1;
    end computation of sum terms in normal equations;
    d11 := sumex2;
    d12 := sumxiex2 × a;
    d13 := sumex1;
    d22 := sumxi2ex2 × a ↑ 2;
    d23 := sumxiex1 × a;
    d33 := n;
    e1  := − sumex2 × a − sumex1 × c + sumyiex1;
    e2  := − sumxiex2 × a ↑ 2 − sumxiex1 × c × a +
      sumxyiex1 × a;
    e3  := − sumex1 × a − n × c + sumyi;
    delta11 := d22 × d33 − d23 ↑ 2;
    delta12 := d13 × d23 − d12 × d33;
    delta13 := d12 × d23 − d13 × d22;
    delta22 := d11 × d33 − d13 ↑ 2;
    delta23 := d12 × d13 − d11 × d23;
    delta33 := d11 × d22 − d12 ↑ 2;
    delta   := d11 × delta11 + d12 × delta12 + d13 × delta13;
    u := (e1 × delta11 + e2 × delta12 + e3 × delta13)/delta;
    v := (e1 × delta12 + e2 × delta22 + e3 × delta23)/delta;
    w := (e1 × delta13 + e2 × delta23 + e3 × delta33)/delta;
    a := a + u;
    b := b + v;
    c := c + w;
    E squared := 0;
    for i := 1 step 1 until n do
      E squared := E squared + (y[i] − c − a × exp(b × x[i])) ↑ 2;
    if l = 1 then go to retry;
    if abs(save − E squared) < epsilon
    then go to 73
    else if l < l max
      then go to retry
      else go to unfurl;
retry: save := E squared;
  end computation of corrected values of a, b, and c;
unfurl: flag := 1;
73: end least squares curve fit to y = a × exp(b × x) + c
```

ALGORITHM 276
CONSTRAINED EXPONENTIAL CURVE FIT [E2]

GERARD R. DEILY (Recd. 27 July 1964 and 16 Apr. 1965)
US Department of Defense, Washington, D. C.
(Now with HRB-Singer, Inc., State College, Pa.)

```
procedure CSXPCVFT (a, b, c, E squared, n, x, y, k, z, epsilon,
  l max, flag, jump);
  integer n, k, l max, flag, jump;
  real a, b, c, E squared, z, epsilon;
  real array x, y;
comment This algorithm will fit a curve defined by the equation
  y = a × exp(b × x) + c to a set {x_i , y_i} of n data points, and
  constrain the curve so it contains the point (x_k , z). The Taylor
```

series modification of the classical least squares method is utilized to approximate a solution to the system of nonlinear equations of condition. After every iteration, the statistic *E squared* is computed as a measure of the goodness of fit. Commencing with the second iteration, the successive values of *E squared* are differenced, and when the difference in absolute value becomes less than *epsilon*, the calculations cease. If the number of iterations necessary to achieve this result exceeds *l max*, a *flag* is set to a nonzero value and the procedure is terminated. In normal usage, the *jump* parameter is brought in as a ZERO.

With certain data sets, convergence difficulties will be experienced. In these cases it is sometimes helpful to first utilize the procedure *EXPCRVFT* [Algorithm 275, *Comm. ACM 9* (Feb. 1966), 85] to obtain initial values for *b* and *c*, and then bring the *jump* parameter in as a ONE in order to bypass the following starting value computations for *b* and *c*.;

```
begin
  integer i, l, m;
  real exp factor;
  if jump = 1 then go to entry;
  comment Computation of initial estimates follows;
  b := 2 × ln(abs(((y[n] − y[n−1]) × (x[2] − x[1]))/
                  ((y[2] − y[1]) × (x[n] − x[n−1]))))/
                  (x[n] + x[n−1] − x[2] − x[1]);
  m := (n+1) ÷ 2;
  exp factor := exp(b × (x[m] − x[k]));
  c := (y[m] − z × exp factor)/(1 − exp factor);
  a := (z − c) × exp(−b × x[k]);
  E squared := 0;
  for i := 1 step 1 until n do
    E squared := E squared + (y[i] − c − a × exp(b × x[i])) ↑ 2;
  comment Computation of corrections follows;
entry: for l := 1 step 1 until l max do
  begin
    real sumex1, sumex2, sumqex1, sumqex2, sumqex1lsex2,
      sumq2ex2, sumyi, sumyiex1, sumqyiex1, zlsc, d11, d12, d22,
      e1, e2, delta, v, w, save;
    sumex1 := sumex2 := sumqex1 := sumqex2 := sumqex1lsex2 :=
      sumq2ex2 := sumyi := sumyiex1 := sumqyiex1 := 0;
    for i := 1 step 1 until n do
    begin
      real q, ex1, ex2, qex1, qex2, qex1lsex2, q2ex2;
      q     := x[i] − x[k];
      ex1   := exp(b × q);
      ex2   := ex1 ↑ 2;
      qex1  := q × ex1;
      qex2  := q × ex2;
      qex1lsex2   := qex1 − qex2;
      q2ex2       := qex2 × q;
      sumex1      := sumex1 + ex1;
      sumex2      := sumex2 + ex2;
      sumqex1     := sumqex1 + qex1;
      sumqex2     := sumqex2 + qex2;
      sumqex1lsex2 := sumqex1lsex2 + qex1lsex2;
      sumq2ex2    := sumq2ex2 + q2ex2;
      sumyi       := sumyi + y[i];
      sumyiex1    := sumyiex1 + ex1 × y[i];
      sumqyiex1   := sumqyiex1 + qex1 × y[i];
    end computation of sum terms in normal equations;
    zlsc := z − c;
    d11  := sumq2ex2 × zlsc ↑ 2;
    d12  := sumqex1lsex2 × zlsc;
    d22  := n − 2 × sumex1 + sumex2;
    e1   := sumqyiex1 × zlsc − sumqex2 × zlsc ↑ 2 −
      sumqex1 × zlsc × c;
    e2 := sumyi − sumyiex1 + sumex1 × (2 × c − z) +
      sumex2 × zlsc − n × c;
    delta := d11 × d22 − d12 ↑ 2;
    v  := (e1 × d22 − e2 × d12)/delta;
    w  := (e2 × d11 − e1 × d12)/delta;
    b  := b + v;
    c  := c + w;
    a  := (z − c) × exp(−b × x[k]);
    E squared := 0;
    for i := 1 step 1 until n do
      E squared := E squared +
        (y[i] − c − a × exp(b × x[i])) ↑ 2;
    if l = 1 then go to retry;
    if abs(save − E squared) < epsilon
    then go to 73
    else if l < l max
      then go to retry
      else go to unfurl;
retry: save := E squared;
  end computation of corrected values of a, b, and c;
unfurl: flag := 1;
73: end constrained least squares fit to y = a × exp(b × x) +
```

ALGORITHM 277
COMPUTATION OF CHEBYSHEV SERIES COEFFICIENTS [C6]

LYLE B. SMITH (Recd. 15 July 1965, 23 July 1965 and 20 Sept. 1965)
Stanford University, Stanford, California

```
procedure CHEBCOEFF (F, N, ODD, EVEN, A);
  value N;
  Boolean ODD, EVEN;
  integer N;
  real procedure F;
  array A;
```

comment This procedure approximates the first $N+1$ coefficients, a_n, of the infinite Chebyshev series expansion of a function $F(x)$ defined on $[-1, 1]$.

$$F(x) = \sum_{n=0}^{\infty}{}' a_n T_n(x), \qquad (1)$$

where $\sum'$ denotes a sum whose first term is halved, and $T_n(x)$ denotes the Chebyshev polynomial of the first kind of degree n, defined by

$$T_n(x) = \cos n\theta, \qquad x = \cos\theta \qquad (n = 0, 1, 2, \cdots).$$

The truncated series $\sum_{n=0}^{\prime N} a_n T_n(x)$, gives an approximation to $F(x)$ which has maximum error almost as small as that of the "best" polynomial approximation of degree N, see [1]. In this procedure the coefficients, a_n, are closely approximated by $B_{n,N}$, $n = 0(1)N$, which are the coefficients of a "Lagrangian" interpolation polynomial coincident with $F(x)$ at the points x_i, $i = 0(1)N$ where $x_i = \cos(\pi i/N)$, see [2]. The $B_{n,N}$ are given by

$$B_{n,N} = \frac{2}{N} \sum_{i=0}^{N}{}'' F(x_i) T_n(x_i) = \frac{2}{N} \sum_{i=0}^{N}{}'' F(x_i) T_i(x_n),$$

where $\sum''$ denotes a sum whose first and last terms are halved. The $B_{n,N}$ are evaluated by a recurrence relation described by Clenshaw in [1] and improved by John Rice [5]. This recurrence relation can also be used to evaluate the truncated series, $\sum_{n=0}^{\prime N} a_n T_n(x)$, once *CHEBCOEFF* has found values for the coefficients. For even N a relation between $B_{n,N/2}$ and $B_{n,N}$ (pointed out by Clenshaw [3, p. 27]) is used in computing $B_{n,N}$.

For large N, $B_{n,N}$ is very close to a_n. In [2] the relation is given as

$$B_{n,N} = a_n + \sum_{p=1}^{\infty} (a_{2pN-n} + a_{2pN+n}). \quad (2)$$

This shows that $\frac{1}{2}B_{N,N}$ approximates a_N quite well for large N since from (2) we see that

$$\tfrac{1}{2}B_{N,N} = a_N + a_{3N} + \cdots . \quad (3)$$

For even N a simple check on the accuracy is available. Since the relation

$$B_{n,N} = B_{n,N/2} - B_{N-n,N}, \qquad n = 0(1)N/2-1 \quad (4)$$

is used in the computation, the difference

$$B_{n,N/2} - B_{n,N} = B_{N-n,N}, \quad (5)$$

which measures in some sense the accuracy of the approximation, is available to the user. For instance, in the example below with $N = 8$ the number $A[7]$ is the difference between $A[1]$ for $N = 4$ and $A[1]$ for $N = 8$.

Parameter Explanation. If the function F is odd or even then the Boolean parameters *ODD* or *EVEN* should be true respectively in which case every other coefficient in the array A will be zero. The array A will contain the coefficients of the truncated series with $N+1$ terms.

Example. For the function $F(x) = e^x$ the following values were computed for $A[n]$ with $N = 4$ and $N = 8$. The computations were done using this procedure written in Extended Algol for the Burroughs B5500 computer. Also shown are computed values for the coefficients of the "best" polynomial of degree 8 from [4] (digits differing from the correct result are in italics).

n	$A[n]$ with $N = 4$	$A[n]$ with $N = 8$	"Best" a_n from [4]	Correct a_n from [1]
0	2.53213 *21539*	2.53213 17555	2.53213 17555	2.53213 17555
1	1.1303*2* *14175*	1.13031 82080	1.13031 82080	1.13031 82080
2	0.271*54* *03174*	0.27149 53395	0.27149 53395	0.27149 53395
3	0.04*487* *97762*	0.04433 68498	0.04433 68498	0.04433 68498
4	0.00547 42404	0.00547 42404	0.00547 42404	0.00547 42404
5		0.00054 29263	0.00054 29263	0.00054 29263
6		0.00004 4977*9*	0.00004 49773	0.00004 49773
7		0.00000 3*2095*	0.00000 31984	0.00000 31984
8		0.00000 01992	0.00000 01998	0.00000 01992;

```
begin
  integer i, m, N2, S1, S2, T1;
  real b0, b1, b2, pi, TWOX, FXN2;
  array FX, X[0:N];
  Boolean TEST;
  pi := 3.14159265359;
  N2 := N ÷ 2;
  comment If N is even TEST is set to true;
  if 2 × N2 = N then TEST := true
  else TEST := false;
  comment Compute the necessary function values;
  for i := 0 step 1 until N do
  begin
    X[i] := cos(pi × i/N);
    FX[i] := F(X[i]);
  end;
  S2 := 1; S1 := 0;
  comment If F(x) is odd or even initialize accordingly;
  if ODD then
  begin
    for m := 0 step 2 until N do
      A[m] := 0;
    S2 := 2; S1 := 1;
  end else
  if EVEN then
  begin
    for m := 1 step 2 until N do
      A[m] := 0;
    S2 := 2; S1 := 0;
  end;
  comment If TEST is true the coefficients are computed in
    two steps;
  FXN2 := FX[N]/2.0;
  if TEST then
  begin
    for m := S1 step S2 until N2 do
    begin
      b1 := 0;
      b0 := FXN2;
      TWOX := 2.0 × X[2 × m];
      for i := N-2 step -2 until 2 do
      begin
        b2 := b1; b1 := b0;
        b0 := TWOX × b1-b2 + FX[i];
      end;
      A[m] := 2.0 × (X[2×m]×b0-b1+FX[0]/2.0)/N2;
    end;
    A[N2] := A[N2]/2.0;
    T1 := S1;
    if ODD ∨ EVEN then
    begin
      if 2 × (N2 ÷ 2) = N2
      then S1 := N2 + 2 - S1
      else S1 := N2 + 1 + S1;
    end
    else S1 := N2 + 1;
  end;
  comment Compute the desired coefficients;
  for m := S1 step S2 until N do
  begin
    b1 := 0;
    b0 := FXN2;
    TWOX := 2.0 × X[m];
    for i := N-1 step - 1 until 1 do
    begin
      b2 := b1; b1 := b0;
      b0 := TWOX × b1 - b2 + FX[i];
    end;
    A[m] := 2.0 × (X[m]×b0-b1+FX[0]/2.0)/N;
  end;
  if TEST then
  begin
    for i := T1 step S2 until N2-1 do
      A[i] := A[i] - A[N-i];
  end;
  A[N] := A[N]/2.0;
end CHEBCOEFF
```


References:

1. Clenshaw, C. W. *Chebyshev Series for Mathematical Functions.* MR 26 #362, Nat. Phys. Lab. Math. Tables, Vol. 5, Dep. Sci. Ind. Res., Her Majesty's Stationery Off., London, 1962.
2. Elliott, D. Truncation errors in two Chebyshev series approximations. *Math. Comp. 19* (1965), 234–248.
3. Clenshaw, C. W. A comparison of "best" polynomial approximations with truncated Chebyshev series expansions. *J. SIAM* {B}, *1* (1964), 26–37.
4. Computed values by Dr. C. L. Lawson. (private communication)
5. Rice, John. On the conditioning of polynomials and rational forms. (submitted for publication).

ALGORITHM 278
GRAPH PLOTTER [J6]

P. LLOYD (Recd. 4 June 1965)
Queen Mary College, London, England

```
procedure graphplotter (N, x, y, m, n, xerror, yerror, g, L, S, EM,
  C0, C1, C2, C3, C4, label);
  value N, m, n, xerror, yerror, g, L, S;
  array x, y;
  integer N, g, m, n, L, S;
  real xerror, yerror;
  string EM, C0, C1, C2, C3, C4;
  label label;
comment This procedure is intended to be used to give an approximate graphical display of a multivalued function, y[i, j] of x[i], on a line printer. Output channel N is selected for all output from graphplotter. The display is confined to points for which 1 ≤ i ≤ m and 1 ≤ j ≤ n where 2 ≤ n ≤ 4. If n = 1, then y is considered to be a one-dimensional array y[i] and the display is again given for 1 ≤ i ≤ m. The format of the print out is arranged so that a margin of g spaces separates the display from the left-hand side of the page. L and S denote the number of lines down the page and the number of spaces across the page which the display will occupy. The graph is plotted so that lines 1 and L correspond to the minimum and maximum values of x, and the spaces 1 and S correspond to the minimum and maximum values of y, that is, y is plotted across the page and x down the page. After the graph has been plotted, the ranges of x and y for which the display is given are printed out in the order as above, separated from the display by a blank line. The strings EM ··· C4 must be such that they occupy only one character position when printed out. The characters of C1 C2 C3 C4 represent y[i,1] y[i,2] y[i,3] y[i,4]. EM is the character printed out round the perimeter of the display. C0 is printed at empty positions. At coincident points the order of precedence of the characters is C1 C2 C3 C4 EM C0. For the special case n=1 the character C1 represents y[i]. Control is passed from the procedure to the point labeled label if the interval between the maximum value and minimum values of x[i] is less than xerror, or if the range of y is less than yerror. If the values of x[i] occur at equal intervals, choosing L=m will make one line equivalent to one interval of x;
begin
  real p, q, xmax, xmin, ymax, ymin;
  integer i, j;
  integer array plot[1:L,1:S];
  xmax := xmin := x[1];
  for i := 2 step 1 until m do
  begin
    if x[i] > xmax then xmax := x[i];
    if x[i] < xmin then xmin := x[i]
  end of hunt for maximum and minimum values of x;
  if n=1 then go to N1A;
  ymax := ymin := y[1,1];
  for i := 1 step 1 until m do
    for j := 1 step 1 until n do
    begin
      if y[i,j] > ymax then ymax := y[i,j];
      if y[i,j] < ymin then ymin := y[i,j]
    end of hunt for maximum and minimum values of y;
escape: if abs(xmax−xmin) < xerror ∨ abs(ymax−ymin) <
  yerror then go to label;
  p := (L−1)/(xmax−xmin);  q := (S−1)/(ymax−ymin);
  for i := 1 step 1 until L do
    for j := 1 step 1 until S do plot[i,j] := 2;
  for i := 1, L do
    for j := 1 step 1 until S do plot[i,j] := 1;
  for i := 2 step 1 until L−1 do
    for j := 1, S do plot[i,j] := 1;
  if n = 1 then go to N1B;
  for i := 1 step 1 until m do
    for j := n step −1 until 1 do
      plot[1+entier(0.5+p×(x[i]−xmin)),
        1+entier(0.5+q×(y[i,j]−ymin))] := j+2;
plotter:
  for i := 1 step 1 until L do
  begin
    NEWLINE(N,1);  SPACE(N,g);
    comment NEWLINE and SPACE must be declared
      globally to graphplotter, NEWLINE(N,p) outputs p carriage returns and p line feeds on channel N, SPACE(N,p)
      outputs p blank character positions on channel N;
    for j := 1 step 1 until S do
    begin
      switch SW := SW1, SW2, SW3, SW4, SW5, SW6;
      go to SW[plot[i,j]];
SW1: outstring(N,EM);  go to fin;
SW2: outstring(N,C0);  go to fin;
SW3: outstring(N,C1);  go to fin;
SW4: outstring(N,C2);  go to fin;
SW5: outstring(N,C3);  go to fin;
SW6: outstring(N,C4);
fin:
    end
  end of display output;
  NEWLINE(N,2);  SPACE(N,g);  outreal(N,xmin);
    outreal(N,xmax);
  outreal(N,ymin);  outreal(N,ymax);
  go to end;
N1A:
  ymax := ymin := y[1];
  for i := 2 step 1 until m do
  begin
    if y[i] > ymax then ymax := y[i];
    if y[i] < ymin then ymin := y[i]
  end of hunt for maximum and minimum values of y when
    n = 1;
  go to escape;
N1B:
  for i := 1 step 1 until m do
    plot[1+entier(0.5+p×(x[i]−xmin)),
      1+entier(0.5+q×(y[i]−ymin))] := 3;
  go to plotter;
end:
end of graphplotter
```