Numerische Mathematik 10, 177—183 (1967)

Smoothing by Spline Functions

CHRISTIAN H. REINSCH

Received February 3, 1967

1. Introduction

During the last years, spline functions have found widespread application, mainly for the purpose of interpolation [1]. However, there may be a demand to replace strict interpolation by some kind of smoothing. Usually, such a situation occurs if the values of the ordinates are given only approximately, for example if they stem from experimental data. In the case in which, for theoretical reasons, the form of the underlying function is known a priori, it is recommended that the latter be approximated by an appropriate trial function which is fitted to the data points by application of the usual least squares technique. Otherwise a spline function may be used. The following algorithm will furnish such a spline function, optimal in a sense specified below. Its application is mainly for curve plotting.

2. Formulation

Let x_i, y_i, $i=0, \ldots, n$ be given and assume that

$$x_0 < x_1 < \cdots < x_n,$$

(the treatment may easily be extended to the case of confluent abscissae). The smoothing function $f(x)$ to be constructed shall

$$\text{minimize} \quad \int_{x_0}^{x_n} g''(x)^2 dx \tag{1}$$

among all functions $g(x)$ such that

$$\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\delta y_i} \right)^2 \leqq S, \quad g \in C^2[x_0, x_n]. \tag{2}$$

Here, $\delta y_i > 0$, $i=0, \ldots, n$ and $S \geqq 0$ are given numbers. The constant S is redundant and is introduced only for convenience. It allows for an implicit rescaling of the quantities δy_i which control the extent of smoothing. Recommended values for S depend on the relative weights δy_i^{-2}. If available, one should use for δy_i an estimate of the standard deviation of the ordinate y_i. In this case, natural values of S lie within the confidence interval corresponding to the left-hand side of (2):

$$N - (2N)^{\frac{1}{2}} \leqq S \leqq N + (2N)^{\frac{1}{2}}, \qquad N = n+1. \tag{3}$$

Choosing S equal to zero leads back to the problem of interpolation by cubic spline functions, or more generally, by spline functions of order $2k-1$, if we replace the second derivative in (1) by the derivative of order k. Here, the special case $k=2$ is mainly used, because it leads to a very simple algorithm for the construction of the function $f(x)$. Moreover, cubic splines are easily evaluated and

give in general satisfactory results. However, the minimum principle (1) influences the shape of the function $f(x)$ much more in smoothing ($S>0$) than in interpolating ($S=0$). Hence, it may be interesting to extend the treatment of (1), (2) to the case of higher derivatives in (1).

3. Cubic Splines as Solution of the Minimum Principle

The solution of (1), (2) may be obtained by the standard methods of the calculus of variations [2]. Thus, by introducing the auxiliary variable z together with the Lagrangian parameter p, we have to look for the minimum of the functional

$$\int_{x_0}^{x_n} g''(x)^2\,dx + p\left\{\sum_{i=0}^{n}\left(\frac{g(x_i)-y_i}{\delta y_i}\right)^2 + z^2 - S\right\}. \tag{4}$$

From the corresponding Euler-Lagrange equations, we determine the optimal function $f(x)$:

$$f''''(x) = 0, \qquad x_i < x < x_{i+1}, \qquad i = 0, \ldots, n-1, \tag{5}$$

$$f^{(k)}(x_i)_- - f^{(k)}(x_i)_+ = \begin{cases} 0 & \text{if } k=0,1 \quad (i=1,\ldots,n-1) \\ 0 & \text{if } k=2 \quad (i=0,\ldots,n) \\ 2p\,\dfrac{f(x_i)-y_i}{\delta y_i^2} & \text{if } k=3 \quad (i=0,\ldots,n), \end{cases} \tag{6}$$

with $f^{(k)}(x_i)_\pm = \lim\limits_{h\downarrow 0} f^{(k)}(x_i \pm h)$. For sake of a uniform notation, we use

$$f''(x_0)_- = f'''(x_0)_- = f''(x_n)_+ = f'''(x_n)_+ = 0.$$

(5) and (6) show that the extremal function $f(x)$ is composed of cubic parabolas

$$f(x) = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3, \quad x_i \leqq x < x_{i+1} \tag{7}$$

which join at their common endpoints such that f, f', and f'' are continuous. Hence, the solution is a cubic spline.

Inserting (7) into (6) we obtain relations among the spline coefficients. A short manipulation yields

from $k=2$:

$$c_0 = c_n = 0, \qquad d_i = (c_{i+1}-c_i)/(3h_i), \qquad i = 0, \ldots, n-1, \tag{8}$$

from $k=0$:

$$b_i = (a_{i+1}-a_i)/h_i - c_i h_i - d_i h_i^2, \qquad i = 0, \ldots, n-1, \tag{9}$$

from $k=1$:

$$Tc = Q^T a, \tag{10}$$

from $k=3$:

$$Qc = pD^{-2}(y-a). \tag{11}$$

Here, the following notation is used:

$$h_i = x_{i+1} - x_i,$$
$$c = (c_1, \ldots, c_{n-1})^T,$$

$y = (y_0, y_1, \ldots, y_n)^T$,

$a = (a_0, a_1, \ldots, a_n)^T$,

$D = \operatorname{diag}(\delta y_0, \ldots, \delta y_n)$,

T a positive definite, tridiagonal matrix of order $n-1$:
$t_{ii} = 2(h_{i-1} + h_i)/3$, $\quad t_{i,i+1} = t_{i+1,i} = h_i/3$,

Q a tridiagonal matrix with $n+1$ rows and $n-1$ columns:
$q_{i-1,i} = 1/h_{i-1}$, $\quad q_{ii} = -1/h_{i-1} - 1/h_i$, $\quad q_{i+1,i} = 1/h_i$.

A left-hand multiplication of (11) by $Q^T D^2$ separates the variable c:

$$(Q^T D^2 Q + p\,T)\,c = p\,Q^T y, \tag{12}$$

$$a = y - p^{-1} D^2 Q c. \tag{13}$$

Thus, if the value of the Lagrangian parameter p is given, we obtain the vector c from (12) and the vector a from (13). The remaining coefficients b_i and d_i are now computed from (8) and (9). Note that the matrix corresponding to (12) has band form with five non-zero diagonals. This matrix is positive definite, if $p \geqq 0$.

4. Determination of the Lagrangian Parameter

The expression (4) has to be minimized also with respect to z and p, leading to the conditions

$$p\,z = 0, \tag{14}$$

$$\sum_{i=0}^{n} \left(\frac{f(x_i) - y_i}{\delta y_i}\right)^2 = S - z^2. \tag{15}$$

By application of (12) and (13) the left-hand side of (15) is easily expressed as $F(p)^2$ where

$$F(p) = \|D\,Q\,(Q^T D^2 Q + p\,T)^{-1} Q^T y\|_2. \tag{16}$$

From (14) we conclude that either $p = 0$ or $z = 0$.

The first case is only possible if $F(0) \leqq S^{\frac{1}{2}}$. This applies, if the given data points are such that the straight line fitted to them by usual least squares prinsiple satisfies condition (2). From (12) we obtain $c = 0$ and from (13) by a limiting process

$$a = y - D^2 Q\,(Q^T D^2 Q)^{-1} Q^T y.$$

Thus, the cubic spline reduces to this straight line.

If $F(0) > S^{\frac{1}{2}}$, we have $p \neq 0$ and $z = 0$. Hence, equality holds in (2) and p must be determined from the equation

$$F(p) = S^{\frac{1}{2}}. \tag{17}$$

It is easy to show that $F(p)$ is a strictly decreasing, convex function if $p \geqq 0$, where $F(p) \to 0$ as $p \to \infty$. Hence, there exists one and only one positive root of (17). There is also at least one negative root. However, it can be shown that the value of the expression (1) is greater for each negative root than for the positive root.

We use NEWTON's method to compute this positive root. If we start with $p=0$, the convexity of $F(p)$ guarantees global convergence. Let us introduce the abbreviation

$$u=p^{-1}c=(Q^TD^2Q+pT)^{-1}Q^Ty.$$

Then we have $F(p)^2=u^TQ^TD^2Qu$ and

$$\begin{aligned} F\,dF/dp &= u^TQ^TD^2Q\,(du/dp) \\ &= p\,u^TT(Q^TD^2Q+pT)^{-1}Tu-u^TTu. \end{aligned} \tag{18}$$

For the computation of u we need the Cholesky decomposition R^TR of the positive definite band matrix Q^TD^2Q+pT. This symmetric decomposition may be used also for the computation of the right-hand side of (18). Hence, we obtain F from the triangular decomposition of Q^TD^2Q+pT, a forward substitution with R^T, and a backward substitution with R. A multiplication with the tridiagonal matrix T and a second forward substitution with R^T yields FdF/dp.

Now the algorithm is:

Start with $p=0$.

(i) Cholesky decomposition R^TR of Q^TD^2Q+pT.

(ii) Compute u from $R^TRu=Q^Ty$ and $v=DQu$, accumulate $e=v^Tv$.

(iii) *If e greater S*
Compute $f=u^TTu$ and $g=w^Tw$ where $R^Tw=Tu$,
replace p by $p+(e-(Se)^{\frac{1}{2}})/(f-p\times g)$,
restart with step (i),

Otherwise

(iv) Compute $a=y-Dv$, $c=pu$ and b_i, d_i according to (8), (9).

Another way to determine the Lagrangian parameter is to apply NEWTON's method to the function $F(1/\bar{p})-S^{\frac{1}{2}}$, starting with $\bar{p}=0$ and producing the reciprocal value of p. Proceeding along the same lines, this alternative way needs only minor modifications of the outlined algorithm. When tested with few examples, convergence was always reached with a slightly reduced number of iterations.

5. Testresults and ALGOL Program

Interpolation is frequently used in connection with numerical differentiation of a function which is given only incompletely by a table. Splines have been used to reduce the error due to the inherent discretization. It is clear, however, that all methods based on interpolation give results which reflect the accuracy of the data. The error part introduced by incorrect ordinates depends on the order of the derivative, the stepwidth of the table, and the accuracy of its entries. If it is dominant, smoothing should replace interpolation.

For demonstration we used a table of $sin(x)$, $x=0^0\,(1^0)\,180^0$, rounded to four decimal places ($\delta y=5_{10}-5/\sqrt{3}$). Approximations to the derivatives of order $k=1, 2, 3$ were computed from difference quotients, interpolation and smoothing by cubic splines. In the table, we list the corresponding mean-square errors obtained from comparison with the true values.

Table. *Mean-square error of numerical derivatives of sin(x), computed either by interpolation* (1), (2) *or by smoothing* (3)

Order	(1) Difference quotient	(2) Cubic spline $S=0$	(3) Cubic spline $S=180$
0	$3.0_{10}-5$	$3.0_{10}-5$	$1.3_{10}-5$
1	$2.6_{10}-3$	$3.4_{10}-3$	$0.21_{10}-3$
2	0.26	0.67	0.0042
3	28	74	0.16

The application of smoothing for curve plotting is demonstrated in Fig. 1.

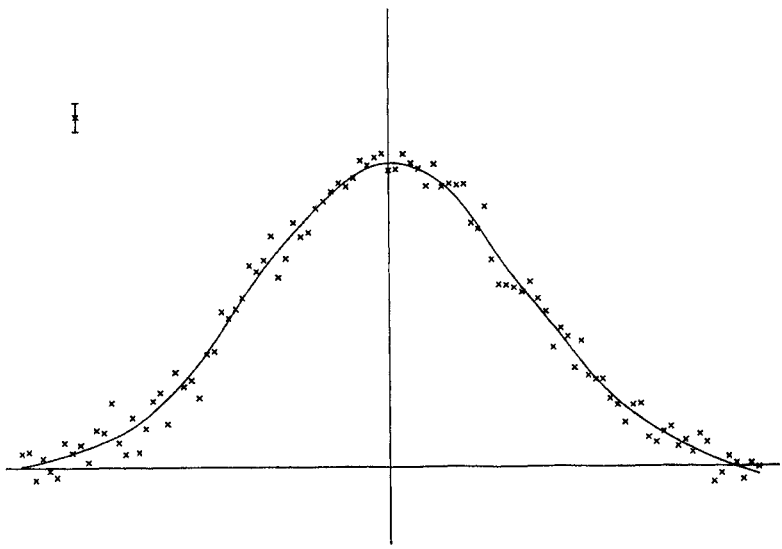

Fig. 1. Smoothing of data points (×) by a cubic spline, $2\delta y_i$ is marked in the upper left-hand corner, $n=100$, $S=90$

The results were produced by the following ALGOL program which corresponds to the modified variant of the algorithm. A rational version of the Cholesky decomposition is used.

Input:

n1, n2 number of first and last data point, *n2*>*n1*.

x, y, dy arrays with *x*[*i*], *y*[*i*], *dy*[*i*]↑(−2) as abscissa, ordinate and relative weight of *i*-th data point, respectively, (*i*=*n1* (1) *n2*). The components of the array *x* must be strictly increasing.

s a non-negative parameter which controls the extent of smoothing: the spline function *f* is determined such that

$$\sum_{i=n1}^{n2}((f(x[i])-y[i])/dy[i])\uparrow 2 \leqq s,$$

where equality holds unless *f* describes a straight line.

Output:

a, b, c, d arrays, collecting the coefficients of the cubic spline f such that with $h = xx - x[i]$ we have $f(xx) = ((d[i] \times h + c[i]) \times h + b[i]) \times h + a[i]$ if $x[i] \leqq xx < x[i+1]$, $i = n1, \ldots, n2-1$. Furthermore, $a[n2] = f(x[n2])$ and $c[n2] = 0$ while $b[n2]$ and $d[n2]$ are left undefined.

```
procedure smooth(n1, n2) data: (x, y, dy, s) result: (a, b, c, d);
    value n1, n2, s;
    integer n1, n2;
    real s;
    array x, y, dy, a, b, c, d;
begin
    integer i, m1, m2;  real e, f, f2, g, h, p;
    array r, r1, r2, t, t1, u, v [n1-1: n2+1];
    m1 := n1-1;  m2 := n2+1;
    r[m1] := r[n1] := r1[n2] := r2[n2] := r2[m2] :=
    u[m1] := u[n1] := u [n2] := u [m2] := p := 0;
    m1 := n1+1;  m2 := n2-1;
    h := x[m1]-x[n1];  f := (y[m1]-y[n1])/h;
    for i := m1 step 1 until m2 do
    begin
        g := h;  h := x[i+1]-x[i];
        e := f;  f := (y[i+1]-y[i])/h;
        a [i] := f-e;  t[i] := 2×(g+h)/3;  t1[i] := h/3;
        r2[i] := dy[i-1]/g;  r[i] := dy[i+1]/h;
        r1[i] := -dy[i]/g-dy[i]/h
    end i;
    for i := m1 step 1 until m2 do
    begin
        b[i] := r[i]×r[i]+r1[i]×r1[i]+r2[i]×r2[i];
        c[i] := r[i]×r1[i+1]+r1[i]×r2[i+1];
        d[i] := r[i]×r2[i+2]
    end i;
    f2 := -s;
next iteration:
    for i := m1 step 1 until m2 do
    begin
        r1[i-1] := f×r[i-1];  r2[i-2] := g×r[i-2];
        r[i] := 1/(p×b[i]+t[i]-f×r1[i-1]-g×r2[i-2]);
        u[i] := a[i]-r1[i-1]×u[i-1]-r2[i-2]×u[i-2];
        f := p×c[i]+t1[i]-h×r1[i-1];  g := h;  h := d[i]×p
    end i;
    for i := m2 step -1 until m1 do
        u[i] := r[i]×u[i]-r1[i]×u[i+1]-r2[i]×u[i+2];
    e := h := 0;
    for i := n1 step 1 until m2 do
```

```
    begin
        g := h;  h := (u[i+1] - u[i])/(x[i+1] - x[i]);
        v[i] := (h - g) × dy[i] × dy[i];  e := e + v[i] × (h - g)
    end i;
    g := v[n2] := -h × dy[n2] × dy[n2];  e := e - g × h;
    g := f2;  f2 := e × p × p;
    if f2 ≥ s ∨ f2 ≤ g then go to fin;
    f := 0;  h := (v[m1] - v[n1])/(x[m1] - x[n1]);
    for i := m1 step 1 until m2 do
    begin
        g := h;  h := (v[i+1] - v[i])/(x[i+1] - x[i]);
        g := h - g - r1[i-1] × r[i-1] - r2[i-2] × r[i-2];
        f := f + g × r[i] × g;  r[i] := g
    end i;
    h := e - p × f;  if h ≤ 0 then go to fin;
    p := p + (s - f2)/((sqrt(s/e) + p) × h);  go to next iteration;
comment Use negative branch of square root, if the sequence of abscissae x[i]
        is strictly decreasing;
fin:
    for i := n1 step 1 until n2 do
    begin
        a[i] := y[i] - p × v[i];
        c[i] := u[i]
    end i;
    for i := n1 step 1 until m2 do
    begin
        h := x[i+1] - x[i];
        d[i] := (c[i+1] - c[i])/(3 × h);
        b[i] := (a[i+1] - a[i])/h - (h × d[i] + c[i]) × h
    end i
end smooth;
```

Acknowledgement. The author is greatly indebted to Dr. R. BULIRSCH and Dr. J. STOER for their stimulating interest and valuable comments.

References

[1] SCHOENBERG, I. J.: On interpolation by spline functions and its minimal properties, p. 109. On Approximation Theory. Proceedings of the Conference held in the Mathematical Research Institute at Oberwolfach, Black Forest, August 4—10, 1963. Basel-Stuttgart: Birkhäuser 1964.

[2] FUNK, P.: Variationsrechnung und ihre Anwendung in Physik und Technik. Berlin-Göttingen-Heidelberg: Springer 1962.

Mathematisches Institut der Technischen Hochschule
8000 München 2
Arcisstr. 21