Computational Statistics and Data Analysis 53 (2009) 932–940

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

A fast compact algorithm for cubic spline smoothing

Howard L. Weinert *

Electrical and Computer Engineering Department, Johns Hopkins University, Baltimore MD 21218, United States

ARTICLE INFO

Article history:
Received 15 February 2008
Received in revised form 27 October 2008
Accepted 27 October 2008
Available online 3 November 2008

ABSTRACT

An efficient algorithm is presented for computing discrete or continuous cubic smoothing splines with uniformly spaced and uniformly weighted measurements. The algorithm computes both the spline values and the generalized cross-validation score. Execution time and memory use are reduced by carefully exploiting the problem's rich structure. The frequency domain properties of the steady-state cubic spline smoother are also examined.

1. Introduction

Although computer speed and memory capacity are steadily increasing, any particular signal processing problem, such as upsampling or digital-to-analog conversion, may have a data set so large that either the solution cannot be produced quickly enough, or the memory capacity of the computer is exceeded. It is therefore imperative to use an algorithm that fully exploits matrix structure to reduce execution time and memory use. We will examine the problem of cubic spline smoothing in this regard, treating the discrete and continuous cases in a unified framework. In both cases, the coefficient matrix in the relevant system of equations has certain properties that have not yet been fully exploited. We will show how to use these properties to produce a faster, more compact algorithm.

In the discrete case, given inaccurate measurements $\{y_1, y_2, \ldots, y_n\}$ of every rth value of an unknown sequence, the objective is to estimate the entire sequence by finding the values $\{x_{1/r}, x_{2/r}, \ldots, x_{n+1-1/r}\}$ that minimize

$$\lambda \sum_{j=1}^{n} (y_j - x_j)^2 + \sum_{i=1}^{nr+r-3} (x_{(i+2)/r} - 2x_{(i+1)/r} + x_{i/r})^2, \tag{1.1}$$

where the positive real smoothing parameter λ controls the tradeoff between smoothness and fidelity to the data. The solution is a discrete cubic smoothing spline. If $r = 1$, the problem reduces to Whittaker–Henderson smoothing (no interpolation or extrapolation), for which a very efficient algorithm is given by Weinert (2007).

In the continuous case, given inaccurate, uniformly spaced samples $\{y_1, y_2, \ldots, y_n\}$ of an analog signal, and defining $x_\beta = x(\beta T)$ where T is the sampling period, we want to compute values $\{x_{1/r}, x_{2/r}, \ldots, x_{n+1-1/r}\}$ of the function $x(t)$ that minimizes

$$\lambda \sum_{j=1}^{n} (y_j - x_j)^2 + \int_a^b (x''(\tau))^2 \mathrm{d}\tau. \tag{1.2}$$

We assume that $[a, b] \supset [T, nT]$ and that $x(t)$ has a square-integrable second derivative on $[a, b]$. The function that minimizes (1.2) is a continuous cubic smoothing spline. Although fundamentally different, the discrete and continuous problems give identical results when $T = r \to \infty$.

* Corresponding address: Electrical and Computer Engineering, Johns Hopkins University, 3400 N. Charles St., 105 Barton Hall, Baltimore MD 21218, United States. Tel.: +1 410 516 2387; fax: +1 410 516 5566.
E-mail addresses: howard@jhu.edu, hweinert@gmail.com.

0167-9473/$ – see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2008.10.036

Spoerl (1943) was the first to formulate and solve the discrete problem. Later work by Duris (1977, 1980) and others was largely a rederivation of Spoerl's results. Recently, Weinert (2001) gave a state space algorithm for this problem, and Eilers (2003) presented an algorithm based on a generalization of the matrix equation of the Whittaker–Henderson problem. By using 0/1 weights, Eilers can deal with nonuniformly spaced, or missing, data. However, his approach is relatively inefficient for uniformly spaced data since his coefficient matrix has size $rn + r - 1$ instead of $n - 2$, and is neither Toeplitz nor persymmetric. For the much better-known continuous case, Reinsch (1967) gave a solution analogous to Spoerl's, and Weinert et al. (1980) and Kohn and Ansley (1989) presented state space algorithms. Even though these existing algorithms require just $O(n)$ floating point operations, we will show that execution time and memory use can be significantly reduced by eliminating unnecessary computations and arrays.

The algorithms mentioned in the previous paragraph require either a matrix factorization or the propagation of a Riccati equation. As discussed more fully in Weinert (2007), these two approaches are entirely equivalent, so with careful implementation the resulting algorithms should have the same execution time and memory use. In this paper we will use matrix factorization.

The results of Spoerl and Reinsch will now be summarized in a unified framework. Let $y^{\mathrm{T}} = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}$ be the measurement vector and $\hat{x}_s^{\mathrm{T}} = \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \cdots & \hat{x}_n \end{bmatrix}$ be the vector of spline values at the measurement locations. Let M be the $(n-2) \times n$ second differencing matrix; for example, when $n = 7$,

$$M = \begin{bmatrix} 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \end{bmatrix}.$$

Let S be the $(n-2) \times (n-2)$ symmetric, tridiagonal Toeplitz matrix with s_0 and s_1 on the diagonal and subdiagonal, respectively, where

$$s_0 = \frac{2r^2 + 1}{3r^2}, \qquad s_1 = \frac{r^2 - 1}{6r^2}, \tag{1.3}$$

in the discrete case, and

$$s_0 = \frac{2}{3}, \qquad s_1 = \frac{1}{6}, \tag{1.4}$$

in the continuous case. In both cases, $s_0 + 2s_1 = 1$. Let $c^{\mathrm{T}} = \begin{bmatrix} c_1 & c_2 & \cdots & c_{n-2} \end{bmatrix}$, where

$$c_{i-1} = (r\lambda)^{-1}(\hat{x}_{i-1/r} - 2\hat{x}_i + \hat{x}_{i+1/r}), \tag{1.5}$$

in the discrete case, and

$$c_{i-1} = (T\lambda)^{-1}\hat{x}''(iT), \tag{1.6}$$

in the continuous case. Then $\hat{x}_s$ and c are uniquely determined by

$$Ac = (\bar{\lambda}S + MM^{\mathrm{T}})c = My, \tag{1.7}$$

$$\hat{x}_s = y - M^{\mathrm{T}}c, \tag{1.8}$$

where $\bar{\lambda} = r^3\lambda$ in the discrete case, and $\bar{\lambda} = T^3\lambda$ in the continuous case. The coefficient matrix A is positive definite, pentadiagonal, and Toeplitz. When $n = 7$ for example,

$$A = \begin{bmatrix} 6 + \bar{\lambda}s_0 & -4 + \bar{\lambda}s_1 & 1 & 0 & 0 \\ -4 + \bar{\lambda}s_1 & 6 + \bar{\lambda}s_0 & -4 + \bar{\lambda}s_1 & 1 & 0 \\ 1 & -4 + \bar{\lambda}s_1 & 6 + \bar{\lambda}s_0 & -4 + \bar{\lambda}s_1 & 1 \\ 0 & 1 & -4 + \bar{\lambda}s_1 & 6 + \bar{\lambda}s_0 & -4 + \bar{\lambda}s_1 \\ 0 & 0 & 1 & -4 + \bar{\lambda}s_1 & 6 + \bar{\lambda}s_0 \end{bmatrix}.$$

The interpolated spline values, which lie on cubics, can be determined from $\hat{x}_s$ and c using

$$\hat{x}_{i+j/r} = \left(1 - \frac{j}{r}\right)\hat{x}_i + \frac{j}{r}\hat{x}_{i+1} - \frac{1}{6}\frac{j}{r}\left(1 - \frac{j}{r}\right)\bar{\lambda}\left(\left(1 + \frac{j}{r}\right)c_i + \left(2 - \frac{j}{r}\right)c_{i-1}\right), \tag{1.9}$$

where $i = 1, 2, \ldots, n-1$ and $j = 1, 2, \ldots, r-1$ and $c_0 = c_{n-1} = 0$. The extrapolated spline values, which lie on lines, can be computed from

$$\hat{x}_{j/r} = \hat{x}_1 + (r - j)(\hat{x}_1 - \hat{x}_{1+1/r}), \tag{1.10}$$

$$\hat{x}_{n+j/r} = \hat{x}_n + j(\hat{x}_n - \hat{x}_{n-1/r}), \tag{1.11}$$

where $j = 1, 2, \ldots, r-1$.

Neither Spoerl nor Reinsch proposed an automatic method of choosing λ. A good adaptive method is generalized cross-validation (GCV), in which λ is chosen to minimize the so-called GCV score (see Wahba (1990) and Green and Silverman (1994) for background). Noting from (1.7) and (1.8) that

$$y - \hat{x}_s = M^{\mathrm{T}}c = M^{\mathrm{T}}A^{-1}My,$$

the GCV score for cubic spline smoothing is

$$\frac{n^{-1}c^{\mathrm{T}}MM^{\mathrm{T}}c}{\left(n^{-1}\mathrm{trace}(M^{\mathrm{T}}A^{-1}M)\right)^2}. \tag{1.12}$$

Since the spline values and GCV score must be re-computed numerous times, algorithm efficiency is imperative.

In the next section we present an efficient algorithm for computing the spline values and GCV score. In Section 3 we verify the accuracy of the algorithm and compare its performance to that of the algorithm in the MATLAB spline toolbox, which is based on Reinsch (1967). In Section 4 we examine the frequency response of the spline smoother in the steady-state case. Section 5 has concluding remarks and Section 6 has the MATLAB M-file that implements our continuous cubic smoothing spline algorithm.

2. The algorithm

To solve (1.7), first compute the entries in $w = My$ one at a time. Because of its simple structure, the coefficient matrix A does not need to be formed or stored. Factor A as

$$A = LDL^{\mathrm{T}}, \tag{2.1}$$

where L is unit lower triangular and banded (bandwidth $= 2$) and D is diagonal. Denoting the elements on the first and second subdiagonals of L as $\{-e_1, -e_2, \ldots, -e_{n-3}\}$ and $\{f_1, f_2, \ldots, f_{n-4}\}$, respectively, and the elements on the diagonal of D as $\{d_1, d_2, \ldots, d_{n-2}\}$, we have

$$a_0 = 6 + \bar{\lambda}s_0, \qquad a_1 = 4 - \bar{\lambda}s_1, \tag{2.2}$$

$$d_1 = a_0, \qquad f_1 = 1/d_1, \qquad \mu_1 = a_1, \qquad e_1 = \mu_1 f_1, \tag{2.3}$$

$$d_2 = a_0 - \mu_1 e_1, \qquad f_2 = 1/d_2, \qquad \mu_2 = a_1 - e_1, \qquad e_2 = \mu_2 f_2, \tag{2.4}$$

$$d_j = a_0 - \mu_{j-1}e_{j-1} - f_{j-2}, \qquad f_j = 1/d_j, \qquad \mu_j = a_1 - e_{j-1}, \qquad e_j = \mu_j f_j. \tag{2.5}$$

Store only the e_j and f_j values. At the same time as the entries in L and D are being computed, solve the triangular system $LDv = w$ using

$$v_1 = f_1 w_1, \qquad v_2 = f_2\left(w_2 + \mu_1 v_1\right), \tag{2.6}$$

$$v_j = f_j\left(w_j + \mu_{j-1}v_{j-1} - v_{j-2}\right). \tag{2.7}$$

Then solve the triangular system $L^{\mathrm{T}}c = v$, overwriting v with c, using

$$c_{n-2} = v_{n-2}, \qquad c_{n-3} = v_{n-3} + e_{n-3}c_{n-2}, \tag{2.8}$$

$$c_j = v_j + e_j c_{j+1} - f_j c_{j+2}. \tag{2.9}$$

Once c has been evaluated, compute the entries in $u = M^{\mathrm{T}}c$ one at a time. The numerator of the GCV score (1.12) can then be determined. To compute the trace appearing in the denominator of the GCV score, use the method introduced in the continuous case by Hutchinson and de Hoog (1985). If g_i, h_i, and q_i, respectively, denote entries on the diagonal and first and second superdiagonals of A^{-1}, one can show that

$$\mathrm{trace}(M^{\mathrm{T}}A^{-1}M) = 6\sum_{i=1}^{n-2} g_i - 8\sum_{i=1}^{n-3} h_i + 2\sum_{i=1}^{n-4} q_i. \tag{2.10}$$

To find the required entries in A^{-1} without determining the entire inverse, consider the identity

$$A^{-1} = D^{-1}L^{-1} + (I - L^{\mathrm{T}})A^{-1}. \tag{2.11}$$

Note that $D^{-1}L^{-1}$ is lower triangular with ith diagonal entry f_i, and $I - L^{\mathrm{T}}$ is upper triangular and banded (bandwidth $= 2$) with zeros on its diagonal and $\{e_1, e_2, \ldots, e_{n-3}\}$ and $\{-f_1, -f_2, \ldots, -f_{n-4}\}$ on its first and second superdiagonals, respectively. Consequently,

$$g_1 = f_{n-2}, \qquad h_1 = e_{n-3}g_1, \qquad g_2 = f_{n-3} + e_{n-3}h_1, \tag{2.12}$$

$$q_{j-2} = e_{n-j-1}h_{j-2} - f_{n-j-1}g_{j-2}, \qquad h_{j-1} = e_{n-j-1}g_{j-1} - f_{n-j-1}h_{j-2}, \tag{2.13}$$

$$g_j = f_{n-j-1} + e_{n-j-1}h_{j-1} - f_{n-j-1}q_{j-2}. \tag{2.14}$$

The sums in (2.10) can be accumulated as g_j, h_j, q_j are computed, eliminating the need to store these quantities. We can speed up the trace computation substantially by noting that since A^{-1} is persymmetric, only about half of the terms in each sum of (2.10) are unique, and only these need to be computed in (2.12)–(2.14).

We can further reduce execution time and memory use by exploiting a little-known result about convergence along the diagonals of L and A^{-1}. Bauer (1954, 1955, 1956) showed that as $j, n \to \infty$, $e_j \to e$ and $f_j \to f$, where e and f satisfy

$$z^4 + (\bar{\lambda}s_1 - 4)z^3 + (\bar{\lambda}s_0 + 6)z^2 + (\bar{\lambda}s_1 - 4)z + 1 = \frac{1}{f}(z^2 - ez + f)(fz^2 - ez + 1). \tag{2.15}$$

He also showed that

$$\left|f_j - f\right| \le \gamma \rho^{2j}, \tag{2.16}$$

where $\gamma > 0$, and ρ is the magnitude of the root of the polynomial (2.15) that lies inside, and nearest to, the unit circle. The e_j sequence converges at the same rate, and so do the sequences in (2.12)–(2.14), with limits denoted by g, h, q.

Formulas can be determined for these limits. Since the polynomial on the left in (2.15) has symmetric coefficients, its roots occur in reciprocal pairs. Let z_1, z_2, z_3, z_4 denote the roots, with $z_1 z_2 = z_3 z_4 = 1$. If $\alpha_1 = z_1 + z_2$ and $\alpha_2 = z_3 + z_4$, then $\alpha_1 + \alpha_2 = 4 - \bar{\lambda}s_1$ and $\alpha_1\alpha_2 = 4 + \bar{\lambda}s_0$. Hence

$$\alpha_1 = 0.5\left(4 - \bar{\lambda}s_1 + \sqrt{\bar{\lambda}(\bar{\lambda}s_1^2 - 4)}\right), \tag{2.17}$$

$$\alpha_2 = 0.5\left(4 - \bar{\lambda}s_1 - \sqrt{\bar{\lambda}(\bar{\lambda}s_1^2 - 4)}\right), \tag{2.18}$$

and therefore

$$z_1 = 0.5\left(\alpha_1 + \sqrt{\alpha_1^2 - 4}\right), \qquad z_2 = 0.5\left(\alpha_1 - \sqrt{\alpha_1^2 - 4}\right), \tag{2.19}$$

$$z_3 = 0.5\left(\alpha_2 + \sqrt{\alpha_2^2 - 4}\right), \qquad z_4 = 0.5\left(\alpha_2 - \sqrt{\alpha_2^2 - 4}\right). \tag{2.20}$$

If $\bar{\lambda} < 4/s_1$, the roots are distinct and complex with positive real parts. Furthermore, z_2 and z_4 are the roots inside the unit circle and

$$f = z_2 z_4, \qquad e = z_2 + z_4, \qquad \rho = |z_2|. \tag{2.21}$$

If $\bar{\lambda} = 4/s_1$, the roots are distinct and purely imaginary with z_2 and z_3 inside the unit circle and

$$f = z_2 z_3, \qquad e = z_2 + z_3, \qquad \rho = |z_2|. \tag{2.22}$$

When $4/s_1 < \bar{\lambda} < 4/s_1^2$, the roots are distinct and complex with negative real parts. When $\bar{\lambda} = 4/s_1^2$, there is a repeated pair of negative real roots, and when $\bar{\lambda} > 4/s_1^2$, there are four distinct negative real roots. In these three cases, z_1 and z_3 are inside the unit circle and

$$f = z_1 z_3, \qquad e = z_1 + z_3, \qquad \rho = |z_1|. \tag{2.23}$$

Using (2.13) and (2.14) we can express g, h, q in terms of e and f:

$$g = \frac{f(1+f)}{(1-f)\left((1+f)^2 - e^2\right)}, \qquad h = \frac{eg}{1+f}, \qquad q = eh - fg. \tag{2.24}$$

With the above facts, we can reduce execution time and memory use by computing the sequences in (2.5), (2.13) and (2.14) only until they are sufficiently close to their limiting values. Ideally, we would want to find the smallest integer N such that

$$\left|\frac{f_j - f}{f}\right| < 10^{-J}, \quad j \ge N, \tag{2.25}$$

and similarly for the other sequences, where the error exponent J is chosen by the user. For programming purposes, however, it is more efficient to estimate the number of required iterations ahead of time. In light of (2.16) and (2.25), we will compute the estimate $\hat{N}$ using

$$\hat{N} = \text{ceil}\left(\frac{\log_{10} f - J}{2\log_{10}\rho}\right). \tag{2.26}$$

As long as $\hat{N} < \text{ceil}\,(n/2) - 1$, we evaluate the quantities in (2.5), (2.13) and (2.14) only for $j \le \hat{N}$, and then set

$$f_j = f, \qquad e_j = e, \qquad g_j = g, \qquad h_j = h, \qquad q_j = q, \qquad j \ge \hat{N} + 1. \tag{2.27}$$

Table 1
Effect of iteration truncation.

β	Optimal λ	Max. relative difference
1	5.8	5.6×10^{-8}
10^{-2}	4.7×10^{2}	3.5×10^{-9}
10^{-4}	1.6×10^{5}	2.4×10^{-10}

Table 2
Execution time and memory use.

n, r	Our algorithm	MATLAB spline toolbox
$10^5, 2$	36 ms/4.8 MB	1.00 s/19.2 MB
$10^6, 2$	367 ms/48 MB	12.2 s/192 MB
$10^5, 10$	117 ms/11.2 MB	1.51 s/25.6 MB
$10^6, 10$	1.19 s/112 MB	17.8 s/256 MB

The value of λ that minimizes the GCV score can be determined by trial and error or, more efficiently, by using a golden section search together with parabolic interpolation (Brent, 1973). Once the optimal λ has been determined, compute the spline values at the measurement locations using (see (1.8))

$$\hat{x}_s = y - u, \tag{2.28}$$

and the interpolated and extrapolated spline values using (1.9)–(1.11).

3. Algorithm performance

To assess how much accuracy is lost when the iterations in (2.5), (2.13) and (2.14) are truncated, measurements were produced by taking 100,000 samples of the analog signal

$$s(t) = 10 + \cos(t) + \cos(1.97t) + \cos(3.38t),$$

at a rate of 1 kHz ($T = 10^{-3}$), and then adding noise with the MATLAB expression beta*randn(1, 100,000), where $\beta = 1, .01, .0001$. For $r = 2$ and $J = 6$, values of the continuous cubic smoothing spline were computed with and without truncation, using the optimal value for λ. Table 1 shows the maximum relative difference between the two sets of results for each noise level β. Truncation clearly produces a negligible difference when $J = 6$, a choice that should be more than adequate for most applications.

Table 2 compares the execution time and memory use of our continuous cubic smoothing spline algorithm ($J = 6$) to that of *csaps* in MATLAB's spline toolbox. Compared to MATLAB's, our algorithm is about 30 times faster and uses only 25% of the memory when $r = 2$, and is about 15 times faster and uses 44% of the memory when $r = 10$. All tests were run on a 1.73 GHz (Centrino) Windows laptop using MATLAB 7. We note that *csaps* does not compute the GCV score.

4. Steady-state cubic spline smoothers

For finite n, the discrete and continuous cubic spline smoothers are time-varying linear filters and thus are not amenable to standard frequency domain analysis. However, for sufficiently large n, the smoothers behave like time-invariant linear filters except near the ends of the data record. These boundary effects can be eliminated by letting the number of measurements become infinite, in which case (1.7) and (1.8) become difference equations:

$$c_i + (\bar{\lambda}s_1 - 4)c_{i+1} + (\bar{\lambda}s_0 + 6)c_{i+2} + (\bar{\lambda}s_1 - 4)c_{i+3} + c_{i+4} = y_{i+2} - 2y_{i+3} + y_{i+4}, \tag{4.1}$$

$$\hat{x}_i = y_i - c_{i-2} + 2c_{i-1} - c_i. \tag{4.2}$$

Taking (bilateral) z-transforms, we see that

$$\left(z^4 + (\bar{\lambda}s_1 - 4)z^3 + (\bar{\lambda}s_0 + 6)z^2 + (\bar{\lambda}s_1 - 4)z + 1\right) C(z) = z^2(z-1)^2 Y(z), \tag{4.3}$$

$$\hat{X}_s(z) = Y(z) - z^{-2}(z-1)^2 C(z), \tag{4.4}$$

and therefore the transfer function (from the measurements to the spline values at the measurement locations) is

$$H(z) = \frac{\hat{X}_s(z)}{Y(z)} = \frac{\bar{\lambda}z^2(s_1 z + s_0 + s_1 z^{-1})}{(z-1)^4 + \bar{\lambda}z^2(s_1 z + s_0 + s_1 z^{-1})}. \tag{4.5}$$

The poles of $H(z)$ are given in (2.19) and (2.20).

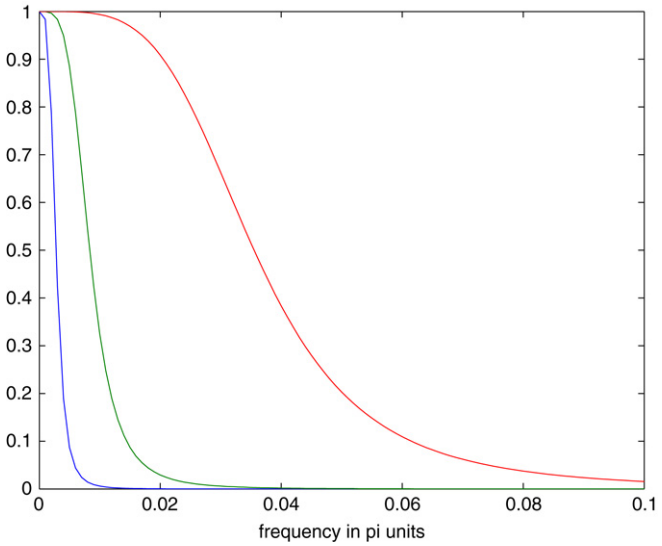

Fig. 1. Frequency response.

The frequency response can be obtained from the transfer function (4.5) by replacing z with $e^{j\omega}$, in which case

$$H(\omega) = \frac{\bar{\lambda}\,(1 - 2s_1(1 - \cos\omega))}{4(1 - \cos\omega)^2 + \bar{\lambda}\,(1 - 2s_1(1 - \cos\omega))}. \tag{4.6}$$

So for large data sets, the cubic spline smoother is approximately a zero-phase lowpass filter with frequency response (4.6). Fig. 1 shows plots of this frequency response in the continuous case ($s_1 = 1/6$) as a function of the normalized frequency ω/π for $\bar{\lambda} = 5.8 \times 10^{-9}, 4.7 \times 10^{-7}, 1.6 \times 10^{-4}$, respectively, from left to right.

The 3 dB cutoff frequency of $H(\omega)$ is

$$\omega_c = \cos^{-1}\left(1 + \frac{1}{4}\alpha\bar{\lambda}s_1 - \frac{1}{4}\sqrt{\alpha\bar{\lambda}(4 + \alpha\bar{\lambda}s_1^2)}\right), \quad \alpha = \sqrt{2} - 1, \tag{4.7}$$

which is well-defined as long as

$$\bar{\lambda} \le \frac{16}{\alpha\,(1 - 4s_1)}, \tag{4.8}$$

in which case we have the inverse formula

$$\bar{\lambda} = \frac{4\,(1 - \cos\omega_c)^2}{\alpha - 2\alpha s_1\,(1 - \cos\omega_c)}. \tag{4.9}$$

Clearly, the filter bandwidth is a monotonically increasing function of $\bar{\lambda}$. Also,

$$\lim_{\bar{\lambda}\to\infty} H(\omega) = 1, \qquad \lim_{\bar{\lambda}\to 0} H(\omega) = \begin{cases} 1, & \omega = 0 \\ 0, & \omega \neq 0. \end{cases} \tag{4.10}$$

Since $H(0) = 1, H^{(i)}(0) = 0,\ i = 1, 2, 3$, and $H^{(4)}(0) \neq 0$, the steady-state smoother will reproduce polynomial inputs of degree three or less. See Unser et al. (1993) and Unser and Blu (2007) for related work on steady-state spline smoothing.

Finally, we note that in steady-state the denominator of the GCV score (1.12) can be computed using

$$\lim_{n\to\infty} n^{-1}\mathrm{trace}(M^{\mathrm{T}}A^{-1}M) = 6g - 8h + 2q. \tag{4.11}$$

De Nicolao et al. (2000) presented a different method for computing the left side of (4.11).

5. Concluding remarks

The key to efficient computation is special-purpose code that takes advantage of matrix structure and the characteristics of the programming language. We have produced a fast compact algorithm for discrete and continuous cubic spline smoothing problems with measurements that have been collected and stored beforehand. If the user wants to begin

processing before all the data have been collected, he can apply these algorithms to individual data blocks that overlap to some degree to minimize boundary effects.

If the smoothing parameter λ is small enough to make the coefficient matrix A ill-conditioned with respect to machine precision, a more accurate solution may result from a QR factorization of the matrix

$$\begin{bmatrix} \sqrt{\lambda}B^{\mathrm{T}} \\ M^{\mathrm{T}} \end{bmatrix}$$

whose condition number is the square root of the condition number of A. Here B is a Cholesky factor of S.

6. MATLAB M-file

What follows is the M-file for our continuous cubic smoothing spline algorithm. The inputs are the measurement *row* vector y, the interpolation parameter r, the sampling period T, the truncation error exponent J, and the smoothing parameter λ. The outputs are the estimate row vector x and the GCV score.

```
function[x,score]=cspline(y,r,T,J,lam)
n=length(y); nc=ceil(n/2); Tcu=T^3; rn=r*n;
x=zeros(1,rn+r-1); c=zeros(1,n); w=zeros(1,n-2); z=zeros(1,n);
for j=1:n-2
   w(j)=y(j)-2*y(j+1)+y(j+2);
end
lamr=lam*Tcu;
a0=6+lamr*2/3; a1=4-lamr/6; a2=sqrt(a1^2-4*(a0-2));
x1=(a1+a2)/2; x2=-(a2-a1)/2;
if lamr>24
   alpha=0.5*(x1+sqrt(x1^2-4)); beta=0.5*(x2+sqrt(x2^2-4));
elseif lamr<24
   alpha=0.5*(x1-sqrt(x1^2-4)); beta=0.5*(x2-sqrt(x2^2-4));
else
   alpha=0.5*(x1-sqrt(x1^2-4)); beta=0.5*(x2+sqrt(x2^2-4));
end

if J>log10(alpha*beta)-(nc-1)*2*log10(abs(alpha))

                      %Use untruncated algorithm since N > nc-2

   %Factor coefficient matrix, solve triangular systems, find trace

   e=zeros(1,n-2); f=zeros(1,n-2);
   d=a0; f(1)=1/d; c(2)=f(1)*w(1); mu=a1; e(1)=mu*f(1);
   d=a0-mu*e(1); f(2)=1/d; c(3)=f(2)*(w(2)+mu*c(2));
   mu=a1-e(1); e(2)=mu*f(2);
   for j=3:n-2
       d=a0-mu*e(j-1)-f(j-2); f(j)=1/d;
       c(j+1)=f(j)*(w(j)+mu*c(j)-c(j-1));
       mu=a1-e(j-1); e(j)=mu*f(j);
   end
   c(n-2)=c(n-2)+e(n-3)*c(n-1);
   for j=n-4:-1:1
      c(j+1)=c(j+1)+e(j)*c(j+2)-f(j)*c(j+3);
   end
   g2=f(n-2); tr1=g2; h=e(n-3)*g2; tr2=h;
   g1=f(n-3)+e(n-3)*h; tr1=tr1+g1; tr3=0;
   for k=n-4:-1:n-nc
      q=e(k)*h-f(k)*g2; tr3=tr3+q;
      h=e(k)*g1-f(k)*h; tr2=tr2+h; g2=g1;
      g1=f(k)*(1-q)+e(k)*h; tr1=tr1+g1;
   end
   q=e(n-nc-1)*h-f(n-nc-1)*g2; tr3=tr3+q;
   h=e(n-nc-1)*g1-f(n-nc-1)*h; tr2=tr2+h;
```

```
   tr1=6*(2*tr1-rem(n,2)*g1);
   tr2=-8*(2*tr2-(1+rem(n,2))*h);
   tr3=2*(2*tr3-rem(n,2)*q);
   tr=(tr1+tr2+tr3)/n;

else                    %Use truncated algorithm since N < nc-1

   %Factor coefficient matrix, solve triangular systems, find trace

   flim=alpha*beta; elim=alpha+beta;
   glim=flim*(1+flim)/((1-flim)*((1+flim)^2-elim^2));
   hlim=elim*glim/(1+flim); qlim=elim*hlim-flim*glim;
   N=ceil((log10(flim)-J)/(2*log10(abs(alpha))));
   e=zeros(1,N); f=zeros(1,N);
   d=a0; f(1)=1/d; c(2)=f(1)*w(1); mu=a1; e(1)=mu*f(1);
   d=a0-mu*e(1); f(2)=1/d; c(3)=f(2)*(w(2)+mu*c(2));
   mu=a1-e(1); e(2)=mu*f(2);
   g2=flim; tr1=g2; h=elim*g2; tr2=h;
   g1=flim+elim*h; tr1=tr1+g1; tr3=0;
   for j=3:N
      d=a0-mu*e(j-1)-f(j-2); f(j)=1/d;
      c(j+1)=f(j)*(w(j)+mu*c(j)-c(j-1));
      mu=a1-e(j-1); e(j)=mu*f(j);
      q=elim*h-flim*g2; tr3=tr3+q;
      h=elim*g1-flim*h; tr2=tr2+h; g2=g1;
      g1=flim*(1-q)+elim*h; tr1=tr1+g1;
   end
   tr1=tr1+(nc-N-1)*glim; tr2=tr2+(nc-N)*hlim;
   tr3=tr3+(nc-N)*qlim; tr1=6*(2*tr1-rem(n,2)*glim);
   tr2=-8*(2*tr2-(1+rem(n,2))*hlim); tr3=2*(2*tr3-rem(n,2)*qlim);
   tr=(tr1+tr2+tr3)/n; mu=a1-elim;
   for j=N+1:n-2
    c(j+1)=flim*(w(j)+mu*c(j)-c(j-1));
   end
   c(n-2)=c(n-2)+elim*c(n-1);
   for j=n-3:-1:N+2
      c(j)=c(j)+elim*c(j+1)-flim*c(j+2);
   end
   for j=N:-1:1
      c(j+1)=c(j+1)+e(j)*c(j+2)-f(j)*c(j+3);
   end
end

%Compute GCV score

   z(1)=c(2); z(2)=c(3)-2*c(2);
   for j=3:n-2
       z(j)=c(j-1)-2*c(j)+c(j+1);
   end
   z(n-1)=c(n-2)-2*c(n-1); z(n)=c(n-1);
   sq=(z*z')/n; score=sq/tr^2;

%Compute estimates

x(r:r:rn)=y-z;
if r<8
   fac1=x(2*r)-x(r)-lamr*c(2)/6; fac2=x(rn)-x(rn-r)+lamr*c(n-1)/6;
   for j=1:r-1
     j1=j/r; j2=1-j1; v=lamr*j1*j2/6; j3=v*(1+j1); j4=v*(2-j1);
     for i=1:n-1
```

```
        ir=i*r;
        x(ir+j)=j2*x(ir)+j1*x(ir+r)-j3*c(i+1)-j4*c(i);
     end
     x(j)=x(r)-j2*fac1; x(rn+j)=x(rn)+j1*fac2;
   end
 else
    lamc=lamr/(6*r^3); r2=2*r; rsq=r^2;
    for i=1:n-1
      ir=i*r; tmp1=x(ir)/r; tmp2=x(ir+r)/r; tmp3=lamc*c(i+1); tmp4=lamc*c(i);
      for j=1:r-1
         x(ir+j)=(r-j)*tmp1+j*tmp2-j*(rsq-j*j)*tmp3-j*(r-j)*(r2-j)*tmp4;
      end
    end
    tmp1=x(r)-x(r+1); tmp2=x(rn)-x(rn-1);
    for j=1:r-1
       x(j)=x(r)+(r-j)*tmp1; x(rn+j)=x(rn)+j*tmp2;
    end
  end
```

References

Bauer, F.L., 1954. Beitrage zur Entwicklung numerischer Verfahren fur programmgesteuerte Rechenanlagen I. Sitzungsberichte Math.-Naturwissen. Klasse Bayerischen Akad, Wissen. Munchen, 275–303.

Bauer, F.L., 1955. Ein direktes Iterationsverfahren zur Hurwitz–Zerlegung eines Polynoms. Archiv Elektrischen Ubertragung 9, 285–290.

Bauer, F.L., 1956. Beitrage zur Entwicklung numerischer Verfahren fur programmgesteuerte Rechenanlagen II. Sitzungsberichte Math.-Naturwissen. Klasse Bayerischen Akad, Wissen. Munchen, 163–203.

Brent, R.P., 1973. Algorithms for Minimization without Derivatives. Prentice-Hall, Englewood Cliffs.

De Nicolao, G., Ferrari-Trecate, G., Sparacino, G., 2000. Fast spline smoothing via spectral factorization concepts. Automatica 36, 1733–1739.

Duris, C.S., 1977. Discrete interpolating and smoothing spline functions. SIAM J. Numer. Anal. 14, 686–698.

Duris, C.S., 1980. Fortran routines for discrete cubic spline interpolation and smoothing. ACM Trans. Math. Software 6, 92–103.

Eilers, P.H.C., 2003. A perfect smoother. Analyt. Chem. 75, 3631–3636.

Green, P.J., Silverman, B.W., 1994. Nonparametric Regression and Generalized Linear Models. Chapman and Hall, London.

Hutchinson, M.F., de Hoog, F.R., 1985. Smoothing noisy data with spline functions. Numer. Math. 47, 99–106.

Kohn, R., Ansley, C.F., 1989. A fast algorithm for signal extraction, influence and cross-validation in state space models. Biometrika 76, 65–79.

Reinsch, C.H., 1967. Smoothing by spline functions. Numer. Math. 10, 177–183.

Spoerl, C.A., 1943. Difference-equation interpolation. Trans. Actuarial Soc. Amer. 44, 289–325.

Unser, M., Aldroubi, A., Eden, M., 1993. B-spline signal processing: Part II-efficient design and applications. IEEE Trans. Signal Process. 41, 834–848.

Unser, M., Blu, T., 2007. Self-similarity: Part I-splines and operators. IEEE Trans. Signal Process. 55, 1352–1363.

Wahba, G., 1990. Spline Models for Observational Data. SIAM, Philadelphia.

Weinert, H.L., 2001. Fixed Interval Smoothing for State Space Models. Springer, New York.

Weinert, H.L., 2007. Efficient computation for Whittaker–Henderson smoothing. Comput. Statist. Data Anal. 52, 959–974.

Weinert, H.L., Byrd, R.H., Sidhu, G.S., 1980. A stochastic framework for recursive computation of spline functions: Part II-smoothing splines. J. Optim. Theory Appl. 30, 255–268.