COMPUTING COX'S SMOOTHING SPLINE SCORE ESTIMATOR

Pin T. Ng

Department of Economics
University of Houston
Houston, TX 77204-5882

Key Words and Phrases: score function; spline; robustness; adaptive estimators.

ABSTRACT

We provide an efficient algorithm for computing the smoothing spline score estimator of Cox (1985). The algorithm exploits the banded structure of the linear algebra involved. Calls are made to the LAPACK, Level 1, 2 and 3 BLAS subroutine libraries designed to be efficient on a wide range of modern high-performance computers.

INTRODUCTION

The score function, defined as $\psi_0(x) = -\log{}' f_0(x) = -f_0'(x)/f_0(x)$, of a probability density function $f_0(x)$ plays an important role in many aspects of statistics. In the robustness literature, it is related to the constructions of L-, M- and R-estimators for location and scale model as well as related estimators for regression models. See Joiner and Hall (1983) for an excellent overview. Estimation of the score function is an integral part of various adaptive L-, M- and R-estimators which achieve the Cramer-Rao efficiency bounds asymptotically. See e.g. Koenker (1982). Estimation of Fisher information also involves estimation of the score function. See Ng (1995). In hypothesis testing, the score function plays a crucial role in making conventional testing procedures more robust to distribution misspecification as in Bickel (1978) and Bera and Ng (1992).

The fundamental contribution of the score function to statistics can, however, be best seen in the realm of exploratory data analysis. Figure 1 and Figure 2 present the probability density and the score functions of some common distributions.

We can see from the figures that the mode of a distribution is characterized by an upward crossing of the score function at the horizontal axis while an anti-mode is located at the point of downward crossing. An exponential distribution has a horizontal score. A tail thicker than the exponential has a negatively sloped score while a tail thinner than the exponential corresponds to an upward sloping score. A Gaussian distribution has a linear score function passing through the horizontal axis at its mean with a slope equals to the reciprocal of its variance. This particular feature was exploited in Bera and Ng (1995) to suggest an alternative to the popular probability or Q-Q plot for identifying potential departures from a Gaussian distribution in data analysis. An estimated score function with a redescending tail towards the horizontal axis indicates departure towards distributions with thicker tails than the normal distribution while a diverging tail suggests departure in the direction of thinner tailed distributions. An estimate of the density function may be recovered through exponentiating the negative integral of the estimated score function although this may seem to be a roundabout approach.

Most existing score estimators are constructed through computing the negative logarithmic derivative of some kernel based density estimators [see e.g. Stone (1975), Manski (1984), and Cox and Martin (1988)] while Csörgo and Révész (1983) suggested a nearest neighbor approach. Cox (1985) presented a totally different approach based on minimizing a penalized mean-squared error rule and giving rise to an estimator which was a variant of a cubic smoothing spline. Ng (1994) found that Cox's smoothing spline score estimator, which finds its theoretical justification from an explicit mean-squared-errors statistical decision criterion, is more robust than several *ad hoc* kernel estimators to distribution variations. This approach to score function estimation is also appealing because it is computationally quite tractable. Using the nonparametric curve fitting analogy, Ng (1994) provides a new characterization of the estimator and suggests an efficient computational strategy taking advantage of the banded structures of the linear algebra involved. In this paper, an implementation of such smoothing spline score estimator is presented.

THEORY

Cox (1985) suggested estimating the score function, ψ_0, of an unknown distribution, F_0, as the minimizer of

$$L[\psi] = \int \left(\psi^2 - 2\psi'\right) dF_n + \lambda \int \left(\psi''(x)\right)^2 dx, \tag{1}$$

over the Sobolev space, $H_2[a,b] = \{\psi : \psi, \psi'$ are absolutely continuous, and $\int_a^b [\psi''(x)]^2 dx < \infty\}$

The motivation for the estimator may be seen by viewing it as the (penalized) empirical analogue of the mean-squared error loss function

$$\int (\psi - \psi_0)^2 \, dF_0 = \int \left(\psi^2 - 2\psi'\right) dF_0 + \int \psi_0^2 dF_0,$$

where F_n, the empirical distribution function, replaces F_0.

Minimizing (1) provides a balance between "fidelity-to-data" represented by the mean-squared error term, the first integral in (1), and smoothness, represented by the second integral in (1), of the estimator $\widehat{\psi}$. The parameter λ controls the trade-off between these objectives, and is usually called the smoothing parameter. In kernel terms it may be viewed as a bandwidth parameters.

Suppose $x_1 < x_2 < ... < x_n$ is an ordered random sample of size n from the unknown density function f_0. Using the so-called Dirac delta function $\delta_\alpha(x) = \delta_0(x - \alpha)$ which assigns mass one to the point α as in Ng (1994), we may rewrite (1) as

$$L[\psi] = \sum_{i=1}^{n} p_i \int \left(\psi^2(x) - 2\psi'(x)\right) \delta_{x_i}(x) \, dx + \lambda \int \left(\psi''(x)\right)^2 dx \tag{2}$$

and derive the Euler-Lagrange conditions,

$$\sum_{i=1}^{n} p_i \left[\psi(x) \, \delta_{x_i}(x) + \delta'_{x_i}(x)\right] + \lambda \psi^{(4)}(x) = 0 \tag{3}$$

Notice that instead of $1/n$, a more general weight p_i satisfying $\sum_{i=1}^{n} p_i = 1$ is assigned to each observation in (2). The general weighting scheme has the advantages of (i) avoiding singular matrix inversions discussed in the **Restrictions** section, and (ii) generating the derivative of the score function estimator, the *J-weight function* estimator, $\widehat{J}\left(F^{-1}(x)\right) = \widehat{\psi}'(x)$, that can be readily used to construct the adaptive L-estimator of Portnoy and Koenker (1989).

METHOD

It was shown in Ng (1994) that the solution, $\widehat{\psi}$, to the Euler-Lagrange condition (3) is piecewise cubic with the form,

$$\widehat{\psi}(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{4}$$

for $x \in [x_i, x_{i+1}]$, $i = 1, \cdots, n-1$, along with

$$\widehat{\psi}^{(k)}(x_i+) - \widehat{\psi}^{(k)}(x_i-) = \begin{cases} 0 & \text{if} \quad k = 0, 1 \\ -\frac{p_i}{\lambda} & \text{if} \quad k = 2 \\ -\frac{p_i \psi(x_i)}{\lambda} & \text{if} \quad k = 3 \end{cases} \tag{5}$$

in which $\widehat{\psi}^{(k)}(x_i\pm) = \lim_{h\to 0} \widehat{\psi}^{(k)}(x_i \pm h)$.

Comparing (5) with condition (6) in Reinsch (1967), we can see that a significant difference between the properties of our $\widehat{\psi}(x)$ and the conventional cubic spline regression estimator is the additional jumps in the second derivative caused by the term involving $\delta'_{x_i}(x)$ in (3). This complicates computation of the coefficients in the cubic spline variant somewhat but we will see that efficient algorithms can still be implemented.

There are $4n$ linear constraints in (5) and $4(n-1)$ unknowns in (4) plus the 4 unknowns a_0, b_0, a_n, b_n characterizing the estimate, $\widehat{\psi}$, outside the interval $[x_1, x_n]$. Note that $c_0 = d_0 = c_n = d_n = 0$, since, were they not, the penalty in (1) could be reduced without disturbing the mean-squared error term. Thus, there are $4n$ equations with $4n$ unknowns, a well posed system of linear equations.

Letting $h_i = x_{i+1} - x_i$ and using (5), as in Ng (1994) we have the following relationships among the vectors of parameters $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ of our cubic spline in (4):

$$Q'\mathbf{a} = R\mathbf{c} + \frac{1}{2\lambda} TP\mathbf{1} \tag{6}$$

$$\mathbf{b} = A\mathbf{a} - C\mathbf{c} - \frac{1}{2\lambda} UP\mathbf{1} \tag{7}$$

$$\mathbf{d} = D\mathbf{c} + \frac{1}{2\lambda} VP\mathbf{1} \tag{8}$$

where

$$\begin{aligned}
\mathbf{a} &= (a_{1,} \cdots, a_n)', \\
\mathbf{b} &= (b_1, \cdots, b_n)', \\
\mathbf{c} &= (c_2, \cdots, c_{n-1})', \\
\mathbf{d} &= (d_1, \cdots, d_{n-1})', \\
\mathbf{1} &= (1, \cdots, 1)' \text{ is a } n\text{x1 column vector of 1's,}
\end{aligned}$$

$$R = \begin{bmatrix}
\frac{2h_1}{3} + \frac{2h_2}{3} & \frac{h_2}{3} & 0 & \cdots & \cdots & 0 \\
\frac{h_2}{3} & \frac{2h_2}{3} + \frac{2h_3}{3} & \frac{h_3}{3} & \ddots & & \vdots \\
0 & \frac{h_3}{3} & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \frac{h_{n-3}}{3} & 0 \\
\vdots & & \ddots & \frac{h_{n-3}}{3} & \frac{2h_{n-3}}{3} + \frac{2h_{n-2}}{3} & \frac{h_{n-2}}{3} \\
0 & \cdots & \cdots & 0 & \frac{h_{n-2}}{3} & \frac{2h_{n-2}}{3} + \frac{2h_{n-1}}{3}
\end{bmatrix}$$

is a $(n-2) \times (n-2)$ tridiagonal symmetric matrix,

$$Q = \begin{bmatrix}
\frac{1}{h_1} & 0 & \cdots & \cdots & \cdots & 0 \\
-\frac{1}{h_1} - \frac{1}{h_2} & \frac{1}{h_2} & \ddots & & & \vdots \\
\frac{1}{h_2} & -\frac{1}{h_2} - \frac{1}{h_3} & \ddots & \ddots & & \vdots \\
0 & \frac{1}{h_3} & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \frac{1}{h_{n-3}} & 0 \\
\vdots & & \ddots & \ddots & -\frac{1}{h_{n-3}} - \frac{1}{h_{n-2}} & \frac{1}{h_{n-2}} \\
\vdots & & & \ddots & \frac{1}{h_{n-2}} & -\frac{1}{h_{n-2}} - \frac{1}{h_{n-1}} \\
0 & \cdots & \cdots & \cdots & 0 & \frac{1}{h_{n-1}}
\end{bmatrix}$$

is a $n \times (n-2)$ banded matrix,

$$A = \begin{bmatrix} -\frac{1}{h_1} & \frac{1}{h_1} & 0 & \cdots & \cdots & 0 \\ 0 & -\frac{1}{h_2} & \frac{1}{h_2} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{1}{h_{n-2}} & \frac{1}{h_{n-2}} & 0 \\ \vdots & & \ddots & 0 & -\frac{1}{h_{n-1}} & \frac{1}{h_{n-1}} \\ 0 & \cdots & \cdots & 0 & -\frac{1}{h_{n-1}} & \frac{1}{h_{n-1}} \end{bmatrix}$$

is a $n \times n$ banded matrix,

$$C = \begin{bmatrix} \frac{h_1}{3} & 0 & \cdots & \cdots & 0 \\ \frac{2h_2}{3} & \frac{h_2}{3} & \ddots & & \vdots \\ 0 & \frac{2h_3}{3} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \frac{h_{n-3}}{3} & \\ \vdots & & \ddots & \frac{2h_{n-2}}{3} & \frac{h_{n-2}}{3} \\ \vdots & & & 0 & \frac{2h_{n-1}}{3} \\ 0 & \cdots & \cdots & 0 & -\frac{h_{n-1}}{3} \end{bmatrix}$$

is a $n \times (n-2)$ banded matrix,

$$D = \begin{bmatrix} \frac{h_1}{3} & 0 & \cdots & \cdots & 0 \\ -\frac{h_2}{3} & \frac{h_2}{3} & \ddots & & \vdots \\ 0 & -\frac{h_3}{3} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \frac{h_{n-3}}{3} & 0 \\ \vdots & & \ddots & -\frac{h_{n-2}}{3} & \frac{h_{n-2}}{3} \\ 0 & \cdots & \cdots & 0 & -\frac{h_{n-1}}{3} \end{bmatrix}$$

is a $(n-1) \times (n-2)$ banded matrix,

$$T = \begin{bmatrix} -\frac{h_1}{3} & \frac{2h_1}{3} & \frac{h_2}{3} & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \frac{2h_2}{3} & \frac{h_3}{3} & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \frac{2h_{n-3}}{3} & \frac{h_{n-2}}{3} & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & \frac{2h_{n-2}}{3} & \frac{h_{n-1}}{3} \end{bmatrix}$$

is a $(n-2) \times n$ banded matrix,

$$U = \begin{bmatrix} -\frac{2h_1}{3} & \frac{h_1}{3} & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \frac{h_2}{3} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \frac{h_{n-2}}{3} & 0 \\ \vdots & & & \ddots & 0 & \frac{h_{n-1}}{3} \\ 0 & \cdots & \cdots & \cdots & 0 & -\frac{2h_{n-1}}{3} \end{bmatrix}$$

is a $n \times n$ banded matrix,

$$V = \begin{bmatrix} \frac{h_1}{3} & \frac{h_1}{3} & 0 & \cdots & \cdots & 0 \\ 0 & 0 & \frac{h_2}{3} & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \frac{h_{n-2}}{3} & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & \frac{h_{n-1}}{3} \end{bmatrix}$$

is a $(n-1) \times n$ banded matrix and

$$P = \begin{bmatrix} p_1 & 0 & \cdots & \cdots & 0 \\ 0 & p_2 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & p_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & p_n \end{bmatrix}$$

is a $n \times n$ diagonal matrix.

Using these relationships, we can express, as in Ng (1994), the penalized loss function (1) as

$$\begin{aligned} L[\psi] &= (\mathbf{a}'P\mathbf{a} - 2\mathbf{1}'P\mathbf{b}) + 2\lambda\mathbf{c}'R\mathbf{c} + 2\mathbf{1}'P'T'\mathbf{c} + K \\ &= \mathbf{a}'\left(P + 2\lambda QR^{-1}Q'\right)\mathbf{a} - 2\mathbf{a}'\left(A - CR^{-1}Q'\right)'P\mathbf{1} + K \end{aligned}$$

where K is a constant independent of the smoothing spline parameters. As a standard quadratic optimization exercise, the solution yields

$$\widehat{\mathbf{a}} = \left(I + 2\lambda P^{-1}QR^{-1}Q'\right)^{-1} P^{-1}\left(A - CR^{-1}Q'\right)' P\mathbf{1} \tag{9}$$

With $\mathbf{\hat{a}}$, estimates for the rest of the coefficients $\mathbf{b}, \mathbf{c}$, and $\mathbf{d}$ can be obtained using (6) —(8). From (6), (9) and writing $\mathbf{y} = P^{-1}(A - CR^{-1}Q')' P\mathbf{1}$, we get

$$\widehat{\mathbf{c}} = \left(R + 2\lambda Q'P^{-1}Q\right)^{-1} Q'\mathbf{y} - \frac{1}{2\lambda}R^{-1}TP\mathbf{1} \tag{10}$$

Similarly, $\mathbf{\hat{a}}$ can alternatively be expressed as

$$\widehat{\mathbf{a}} = \left(I - 2\lambda P^{-1}Q\left(R + 2\lambda Q'P^{-1}Q\right)^{-1} Q'\right)\mathbf{y} \tag{11}$$

The $\mathbf{y}$ in (10) and (11), called the *pseudo* y in Ng (1994), plays the role of the dependent variable in nonparametric curve fitting.

IMPLEMENTATION

The banded structure of the linear systems to be solved in (10) and (11) are much preferable to (9). Efficient algorithms can be written using subroutines tailored specifically for banded matrices. The public domain LAPACK (Linear Algebra PACKage) is one of those designed to be efficient on a wide range of modern high-performance computers (e.g. vector processors, high-performance "super-scalar" workstation, and shard memory multiprocessors) by calling the Level 1, 2 and 3 BLAS (Basic Linear Algebra Subprograms). The banded nature of the linear algebra in (7) — (8) and (10) —(11) can also be exploited by using the public domain Level 1 and 2 BLAS subroutines. As machine code versions of the BLAS are written by more and more machine vendors to fully exploit specific machine architecture, we can achieve good performance in a portable way over a large class of modern computers by calling the Level 1, Level 2 BLAS and LAPACK subprograms.

The subroutine *splscr* first calls *cox* to compute the coefficient vectors **a, b, c,** and **d** of the smoothing spline score estimator. The coefficient vectors are returned to *splscr* through the matrix **f**. *splscr* then calls *hat* with the matrix **f** to (i) allocate a bin number to each evaluation point, at which estimate of the score is desired, (ii) compute the differences between the evaluation points and the lower bounds of their respective bins, which are the n sample points of the random sample of size n from the unknown density function, f_0 and (iii) compute the estimated values of the score function at the evaluation points using the formula given by (4). As a by product, the pseudo $\mathbf{y}$ and the L-score function, which is the derivative of the score function, are also returned in *splscr*.

STRUCTURE

subroutine splscr(nx,x,p,nz,z,exlam,big,w,iw,ift,psi,lscore,suy)
Formal parameters:

nx	Integer	input:	number of observations
x	Double(nx)	input:	random sample of size nx from F_0
p	Double(nx)	input:	weights associated with x
nz	Integer	input:	number of evaluation points
z	Double(nz)	input:	evaluation points
$exlam$	Double	input:	pre-chosen smoothing parameter, $\lambda = 10^{exlam}$
big	Double	input:	a machine-dependent largest finite floating-point number or any number greater than the largest value in x
w	Double(37*nx-18+nz)	work:	double precision work array of length at least $37 * nx - 18 + nz$
iw	Integer(nx+nz)	work:	integer work array of legnth at least $nx + nz$
ift	Integer	output:	premature exit code:

			0 – OK
			i – the reciprocal condition number during the ith call to linear system solution routine *dptsvx* or *dpbsvx* is less than the machine precision
psi	Double(nz)	output:	estimated scores at the evaluation points
lscore	Double(nz)	output:	estimated L-scores at the evaluation points
suy	Double(nx)	output:	the pseudo y

subroutine cox(nx,x,p,exlam,h,c,r,q,pq,a,pa,w1,w2,w3,w4,w5,w6,w7,w8,iw, ift,f,suy)

Formal parameters:

nx	Integer	input:	as in *splscr*
x	Double(nx)	input:	as in *splscr*
p	Double(nx)	input:	as in *splscr*
exlam	Double	input:	as in *splscr*
h	Double(nx-1)	work:	stores the ordered spacing of x
c	Double(3,nx-2)	work:	stores C in banded form
r	Double(2,nx-2)	work:	stores R in banded form
q	Double(3,nx-2)	work:	stores Q in banded form
pq	Double(3,nx-2)	work:	stores $P^{-1}Q$ in banded form
a	Double(3,nx)	work:	stores A in banded form
pa	Double(3,nx)	work:	stores $P^{-1}A'$ in banded form
w1	Double(nx)	work:	

w2	Double(nx)	work:	
w3	Doublenx	work:	
w4	Double(nx)	work:	
w5	Doublenx	work:	
w6	Double(3,nx)	work:	
w7	Double(3,nx	work:	
w8	Double(3*nx)	work:	
iw	Integer(nx)	work:	
ift	Integer	output:	as in *splscr*
f	Double(nx+1,4)	output:	coefficients of the smoothing spline score estimator: f(i+1,1) = a_i f(i+1,2) = b_i f(i+1,3) = c_i f(i+1,4) = d_i
suy	Double(nx)	output:	as in *splscr*

subroutine hat(nx,x,p,nz,z,iz,dz,xx,f,big,psi,lscore)

Formal parameters:

nx	Integer	input:	as in *splscr*
x	Double(nx)	input:	as in *splscr*
p	Double(nx)	input:	as in *splscr*
nz	Integer	input:	as in *splscr*
z	Double(nz)	input:	as in *splscr*
iz	Integer(nz)	work:	vector of bin numbers of the evaluation points
dz	Double(nz)	work:	vector of distances between

			the evaluation points and the lower bounds of their respective bins
xx	Double(nx+1)	work:	
f	Double(nx+1,4)	input:	output from *cox*
big	Double	input:	as in *splscr*
psi	Double(nz)	output:	as in *splscr*
lscore	Double(nz)	output:	as in *splscr*

Auxiliary algorithms:

The following subroutines from the Level 1 BLAS are called:

subroutine dcopy(n, x, incx, y, incy) — copies x to y

subroutine daxpy(n, alpha, x, incx, y, incy) — constant times a vector plus a vector

The following subroutine from the Level 2 BLAS is called:

subroutine dgbmv(trans, m, n, kl, ku, alpha, a, lda, x, incx, beta, y, incy) — general band matrix-vector product

The following subroutines from the LAPACK are called:

subroutine dptsvx(fact, n, nrhs, d, e, df, ef, b, ldb, x, ldx, rcond, ferr, berr, work, info) — solves a symmetric positive definite tridiagonal system of linear equations with estimate of the condition number and error bounds of the solution

subroutine dpbsvx(fact, uplo, n, kd, nrhs, ab, ldab, afb, ldafb, equed, s, b, ldb, x, ldx, rcond, ferr, berr, work, iwork, info) — solves a symmetric positive definite banded system of linear equations with estimate of the condition number and error bounds of the solution

RESTRICTIONS

The vector of random sample x of size nx from distribution F_0 passed to subroutine *cox* is assumed to be pre-sorted in ascending order. It is also essential that the random sample x does not have equal valued elements, which will theoretically never occur for random sample drawn from continuous distribution, that can result in floating point overflows in *cox* due to division by zero values of h_i for any $i = 1, \cdots, nx$. Close values of adjacent elements in the sorted x will still, nevertheless, be very likely to result in a premature exit from *cox* caused by calls to *dptsvx* or *dpbsvx* from within *cox.* When such premature exit occur, an alternative will be to replace the close values with a single value, e.g. the average, having weight equals to the sum of their respective weights.

PRECISION

We suggest using double precision on 32 bit machines. On 64 bits machines, single precision would be adequate. This can easily be done through changing the "double precision" statements to "real" in subroutines *splscr, cox* and *hat*, changing the constant definitions in the "parameter" statements from double to single, and calling the single precision versions of the BLAS1, BLAS2 and LAPACK subroutines *scopy, saxpy, sgbmv, sptsvx* and *spbsvx* in *cox* and *hat.*

ACKNOWLEDGEMENT

I wish to express my appreciation to Roger Koenker and Dennis Cox for their helpful discussions. Computations are performed on equipment supported by National Science Foundation Grants SES 89-22472 and SBR 93-20555.

BIBLIOGRAPHY

Anderson, E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorensen, (1992), *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia.

Bera, A.K., and P.T. Ng, (1992), Robust Tests for Heteroskedasticity and Autocorrelation Using Score Function, working paper.

Bera, A.K., and P.T. Ng, (1994), Tests for Normality Using Estimated Score Function, *Journal of Statistical Computation and Simulation,* 52, 273-287.

Bickel, P.J., (1978), Using Residuals Robustly I: Tests for Heteroscedasticity, Nonlinearity, *The Annals of Statistics*, 6, 266-291.

Cox, Dennis D., (1985), A Penalty Method for Nonparametric Estimation of the Logarithmic Derivative of a Density Function, *Annals of the Institute of Statistical Mathematics,* 37, 271-288.

Cox, Dennis D. and D.R. Martin, (1988), Estimation of Score Function, working paper.

Csörgo, M. and P. Révész, (1983), An N.N-estimator for the Score Function, Seminarbericht Nr.49, Proceedings of the First Easter Conference on Model Theory, Sektion Mathematik.

Dongarra, J.J., J. Du Croz, I.S. Duff and S. Hammarling, (1990), A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Transaction on Mathematical Software*, 16, 1-17.

Dongarra, J.J., J. Du Croz, S. Hammarling and R.J. Hanson, (1988), An Extended Set of FORTRAN Basic Linear Algebra Subprograms, *ACM Transaction on Mathematical Software*, 14, 1-17.

Joiner, B.L. and D.L. Hall, (1983), The Ubiquitous Role of f'/f in Efficient Estimation of Location, *The American Statistician*, 37, 128-133.

Koenker, R.W., (1982), Robust Methods in Econometrics, *Econometric*

Reviews, 1, 213-255.

Lawson, C.L., R.J. Hanson, D.R. Kincaid and F.T. Krogh, (1979), Basic Linear Algebra Subprograms for Fortran Usage, *ACM Transactions on Mathematical Software*, 5, 308-323.

Manski, C., (1984), Adaptive Estimation of Non-linear Regression Models, *Econometric Reviews*, 3, 145-194.

Ng, P.T., (1994), Smoothing Spline Score Estimator, *SIAM, Journal on Scientific Computing*, 15, 1003-1025.

Ng, P.T., (1995), Finite Sample Properties of Adaptive Regression Estimators, *Econometric Reviews*, 14, 267-297.

Portnoy, S. and Koenker, R., (1989), Adaptive L-Estimation of Linear Models, *The Annals of Statistics*, 17, 362-381.

Reinsch, C., (1967), Smoothing by Spline Functions, *Numerische Mathematik*, 10, 177-183.

Stone, C., (1975), Adaptive Maximum Likelihood Estimation of A Location Parameter, *The Annals of Statistics*, 3, 267-284.

```
      subroutine splscr(nx,x,p,nz,z,exlam,big,w,iw,ift,psi,
     & lscore,suy)
c
c     Algorithm to compute the estimated score function of an
c     unknown distribution using smoothing spline with
c     pre-chosen smoothing parameter
c     Subroutines called:
c         "cox","hat"
c
      integer nx,nz,ift,iw(1),nx3,ic,ir,iq,ipq,ia,ipa
      integer iw1,iw2,iw3,iw4,iw5,iw6,iw7,iw8,ixx,idz,iif,iiz
      double precision x(1),p(1),z(1),exlam,psi(1),lscore(1)
```

```
      double precision big,w(1),suy(1)
      nx3 = 3*nx
      ic = nx
      ir = ic+nx3-6
      iq = ir+2*(nx-2)
      ipq = iq+nx3-6
      ia = ipq+nx3-6
      ipa = ia+nx3
      iw1 = ipa+nx3
      iw2=iw1+nx
      iw3=iw2+nx
      iw4=iw3+nx
      iw5=iw4+nx
      iw6=iw5+nx
      iw7=iw6+nx3
      iw8=iw7+nx3
      ixx=iw8+nx3
      idz=ixx+nx+1
      iif=idz+nz
      iiz=nx+1
      call cox(nx,x,p,exlam,w(1),w(ic),w(ir),w(iq),w(ipq),w(ia),
     & w(ipa),w(iw1),w(iw2),w(iw3),w(iw4),w(iw5),w(iw6),w(iw7),
     & w(iw8),iw(1),ift,w(iif),suy)
      call hat(nx,x,p,nz,z,iw(iiz),w(idz),w(ixx),w(iif),big,psi,
     & lscore)
      return
      end


      subroutine cox(nx,x,p,exlam,h,c,r,q,pq,a,pa,w1,w2,w3,w4,w5,
     & w6,w7,w8,iw,ift,f,suy)
c
c     Subroutine to compute the smoothing spline coefficients
c       using equation (6), (7), (9) and (10)
c     BLAS1 routines called:
c        "dcopy","daxpy"
```

```
c     BLAS2 routines called:
c        "dgbmv"
c     LAPACK routines called:
c        "dptsvx","dpbsvx"
c
      character*1 equed
      integer i,nx,ift,nxm1,nxm2,nxm3,nxp1,info,iw(1)
      double precision x(1),p(1),exlam,f(nx+1,1)
      double precision zero,one,two,three,onethd,twothd,lambda
      double precision h(1),c(3,1),r(2,1),q(3,1),pq(3,1),a(3,1)
      double precision pa(3,1),w1(1),w2(1),w3(1),w4(1),w5(1)
      double precision w6(3,1),w7(3,1),w8(1),suy(1),rcond
      double precision ferr,berr
      parameter (zero = 0.0d1, one = 0.1d1, two = .2d1,
     & three = .3d1, onethd = .1d1/.3d1, twothd = .2d1/.3d1)
c
      ift = 0
      nxm1 = nx-1
      nxm2 = nx-2
      nxm3 = nx-3
      nxp1 = nx+1
      lambda = 1.d1**exlam
c
c     Compute ordered spacings
c
      do 10 i=1,nxm1
         h(i) = x(i+1)-x(i)
 10   continue
c
c     Pack the C matrix into c, T into w7, Q into q,
c       P^-1*Q into pq, A' into a, and P^-1*A' into pa
c     Note: in loop 20, flow dependencies are traded for
c       duplicated computations so as to achieve parallelism
c
      call dcopy(nxm2,zero,0,c(3,1),3)
      call dcopy(nxm2,zero,0,w7(3,1),3)
```

```
      call dcopy(nxm1,zero,0,a(1,1),3)
      call dcopy(nxm1,zero,0,pa(1,1),3)
      do 20 i = 1,nxm2
         c(1,i) = onethd*h(i)
         c(2,i) = twothd*h(i+1)
         w7(1,i+2) = onethd*h(i+1)
         w7(2,i+1) = twothd*h(i)
         q(1,i) = one/h(i)
         q(2,i) = -(one/h(i)+one/h(i+1))
         q(3,i) = one/h(i+1)
         pq(1,i) = one/h(i)/p(i)
         pq(2,i) = -(one/h(i)+one/h(i+1))/p(i+1)
         pq(3,i) = one/h(i+1)/p(i+2)
         a(2,i) = -one/h(i)
         a(3,i) = one/h(i)
         pa(2,i) = -one/h(i)/p(i)
         pa(3,i) = one/h(i)/p(i+1)
 20   continue
      c(3,nxm2) = -onethd*h(nxm1)
      w7(3,1) = -c(1,1)
      a(2,nx) = one/h(nxm1)
      a(2,nxm1) = -a(2,nx)
      a(3,nxm1) = a(2,nx)
      a(3,nx) = zero
      a(1,nx) = a(2,nxm1)
      pa(2,nx) = a(2,nx)/p(nx)
      pa(2,nxm1) = -a(2,nx)/p(nxm1)
      pa(3,nxm1) = pa(2,nx)
      pa(3,nx) = zero
      pa(1,nx) = pa(2,nxm1)
c
c     Pack the R matrix into r
c
      do 30 i = 2,nxm3
         r(1,i) = c(2,i-1)+c(2,i)
         r(2,i) = c(1,i+1)
```

```
 30   continue
      r(1,1) = c(2,1)+twothd*h(1)
      r(1,nxm2) = c(2,nxm3)+c(2,nxm2)
      r(2,1) = c(1,2)
c
c     Compute pseudo y = P^-1*A'*P*1 - P^-1*Q*R^-1*C'*P*1
c
c     Create C'*P*1
c
      call dgbmv("t",nx,nxm2,2,0,one,c,3,p,1,zero,suy,1)
      call dcopy(nxm2,r(1,1),2,w1,1)
      call dcopy(nxm2,r(2,1),2,w2,1)
c
c     Make R^-1*C'*P*1; result stored in w5
c
      call dptsvx("n",nxm2,1,w1,w2,w3,w4,suy,nxm2,w5,nxm2,rcond,
     & ferr,berr,w6,info)
      if (info .gt. 0) then
         ift = 1
         return
      endif
c
c     Create P^-1*Q*R^-1*C'*P*1
c
      call dgbmv("n",nx,nxm2,2,0,one,pq,3,w5,1,zero,suy,1)
c
c     Psuedo y stored in suy
c
      call dgbmv("n",nx,nx,1,1,one,pa,3,p,1,-one,suy,1)
c
c     Compute R^-1*T*P*1, the second part of the c
c       coefficient vector and store temporarily in w5
c
      call dgbmv("n",nxm2,nx,0,2,one,w7,3,p,1,zero,w5,1)
      call dptsvx("f",nxm2,1,w1,w2,w3,w4,w5,nxm2,w8,nxm2,rcond,
     & ferr,berr,w6,info)
```

```
      if (info .gt. 0) then
         ift = 2
         return
      endif
      call dcopy(nxm2,w8,1,w5,1)
c
c     Compute (R+lambda*Q'*P^-1*Q)^-1*Q'*y, the first part of
c       the c coefficient vector of smoothing spline
c     Solution returned in w4
c
c     Create R+lambda*Q'*P^-1*Q and store in w6
c     Create Q'*y and store in w2 and w3
c
      do 40 i = 1,nxm2
         w6(1,i) = (q(1,i)*pq(1,i)+q(2,i)*pq(2,i)+q(3,i)
     &    *pq(3,i))*lambda+r(1,i)
         w6(2,i) = (q(1,i+1)*pq(2,i)+q(2,i+1)*pq(3,i))
     &    *lambda+r(2,i)
         w6(3,i) = (q(1,i+2)*pq(3,i))*lambda
 40   continue
      call dgbmv("t",nx,nxm2,2,0,one,q,3,suy,1,zero,w2,1)
      call dcopy(nxm2,w2,1,w3,1)
      call dpbsvx("e","l",nxm2,2,1,w6,3,w7,3,equed,w1,w2,nxm2,
     & w4,nxm2,rcond,ferr,berr,w8,iw,info)
      if (info .gt. 0) then
         ift = 3
         return
      endif
c
c     Store the c coefficient vector in f(*,3)
c
      call dcopy(nxm2,w4,1,w1,1)
      call daxpy(nxm2,-one/lambda,w5,1,w1,1)
      call dcopy(nxm2,w1,1,f(3,3),1)
      f(1,3) = zero
      f(2,3) = -p(1)/lambda
```

```
      f(nxp1,3) = zero
c
c     Store the a coefficient vector in f(*,1)
c
      call dcopy(nx,suy,1,w1,1)
      call dgbmv("n",nx,nxm2,2,0,-lambda,pq,3,w4,1,one,w1,1)
      call dcopy(nx,w1,1,f(2,1),1)
      f(1,1) = f(2,1)
c
c     Store the b coefficient vector in f(*,2) and the d
c       coefficient in f(*,4)
c
      do 50 i = 1,nxm1
         f(i+1,4) = (f(i+2,3)-f(i+1,3)+p(i+1)/lambda)
     &    /(h(i)*three)
         f(i+1,2) = (f(i+2,1)-f(i+1,1))/h(i)-(two*f(i+1,3)
     &    +f(i+2,3)+p(i+1)/lambda)*h(i)*onethd
 50   continue
      f(1,2) = f(2,2)
      f(nxp1,2) = (f(nxp1,1)-f(nx,1))/h(nxm1)+h(nxm1)*(f(nx,3)+
     & two*p(nx)/lambda)*onethd
      f(1,4) = zero
      f(nxp1,4) = zero
      return
      end


      subroutine hat(nx,x,p,nz,z,iz,dz,xx,f,big,psi,lscore)
c
c     Subroutine to compute the score estimates at the
c       evaluation points, z(i), using equation (3).  As a by
c       product, the L-score estimate at each z(i) is also
c       returned
c     BLAS1 routines called:
c        "dcopy"
c
```

```
      integer nx,nz,iz(1)
      double precision zero,half,one,two,three,big
      parameter (zero = 0.0d0, half = 0.5d0, one = 0.1d1,
     &  two = 0.2d1, three = .3d1)
      double precision x(1),p(1),z(1),dz(1),xx(1),f(nx+1,4)
      double precision psi(1),lscore(1)
c
c     Assign bin numbers, iz(i), to evaluation points, z(i)
c
      call dcopy(nx,x,1,xx,1)
      xx(nx+1) = big
      do 10 i=1,nz
         do 20 j=1,nx+1
            if (z(i) .lt. xx(j)) then
               iz(i)=j-1
               goto 10
            endif
 20      continue
 10   continue
c
c     Computes the differences between evaluation points and
c      the lower bounds of their respective bins
c
      call dcopy(nx,x,1,xx(2),1)
      xx(1) = xx(2)
      do 30 i=1,nz
         dz(i)=z(i)-xx(iz(i)+1)
 30   continue
c
c     Compute the score and L-score estimates
c
      do 40 i = 1,nz
         psi(i) = f(iz(i)+1,1)+f(iz(i)+1,2)*dz(i)+f(iz(i)+1,3)*
     &   dz(i)**2+f(iz(i)+1,4)*dz(i)**3
         lscore(i) = f(iz(i)+1,2)+two*f(iz(i)+1,3)*dz(i)+three*
     &    f(iz(i)+1,4)*dz(i)**2
```

```
40    continue
      return
      end
```