COMPUTING NONPARAMETRIC FUNCTION ESTIMATES

Clive R. Loader
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA

Abstract: We discuss computational approaches to multivariate function estimation using strengths from the spline and local fitting worlds. In particular, we consider piecewise polynomial representations of the surface that can be easily evaluated at any point in the domain, and estimating parameters using local approaches. Ideas along these lines using rectangular partitions of the domain have been used previously by Silverman (1981) and Cleveland and Grosse (1991). In this paper we propose triangular partitions which may offer greater efficiency and adaptability in modeling surfaces.

1 Introduction

Figure 1 is a contour plot of a density estimate. The sample here consists of 225 observations from an equal mixture of three standard bivariate normal distributions.

This figure was constructed by direct application of the local likelihood density estimate, discussed in Loader (1993). In particular, for each point x on a 100×100 grid, the following system of equations was solved for a:

$$\frac{1}{n}\sum_{i=1}^{n} A(X_i - x)K\left(\frac{X_i - x}{h}\right) = \int A(u-x)K\left(\frac{u-x}{h}\right)e^{\langle a, A(u-x)\rangle}du \qquad (1)$$

Here, $A(v)$ is a vector of the ten cubic basis polynomials and a is a vector of unknown local coefficients. The density estimate is $\hat{f}(x) = \exp(\langle a, A(0)\rangle)$. As an estimate, the local likelihood method performs quite well; the three peaks are clearly separated and are reproduced almost to full height; the true peak height is about 0.053.

However, as a computational method direct application of the local likelihood method is wasteful and inefficient. The local likelihood method assumes the underlying density is smooth, and therefore it seems unnecessary to work independently at very close points.

Moreover, we only have the estimate evaluated on a discrete grid of points; for many purposes, this may be unsatisfactory. To use the fitted surface for classification or prediction requires us to be able to readily evaluate the surface at any point; not just the grid points. In the regression setting, a residual plot requires evaluation of the surface at the data points. Particularly in higher dimensions, one must visualize the surface by looking at lower dimensional sections. Or, one may be interested in looking more closely at part of the surface.

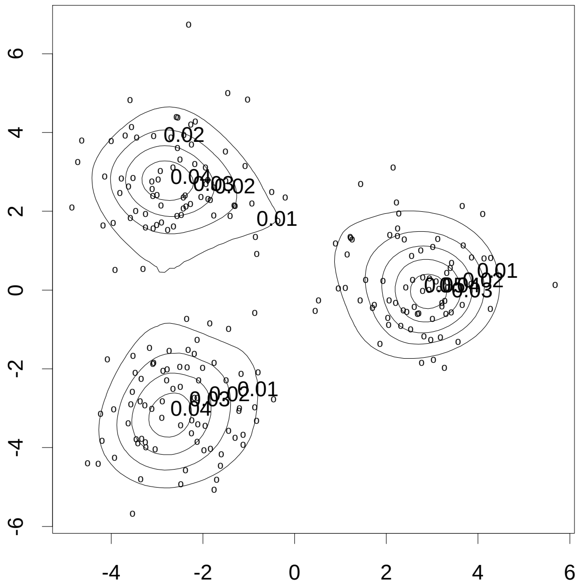

Figure 1: Density Estimate from 225 points. Direct application of the local likelihood method. Local cubic fitting was used with a variable bandwidth covering 113 data points.

The subject of this paper will be better ways to compute local fits. In particular, we require efficient ways to compute and represent the fitted surface. The examples presented are all density estimation, although very little is specific to this case.

2 What Computation is, and is not

The computation of local fits is often portrayed in the statistics literature as a race to evaluate the surface as fast as possible at a predetermined set of points. This represents a very narrow focus; while computation speed is certainly important, there are many other important considerations.

Fundamentally, the fit must summarize the data; not vice versa. The surface should be represented in a compact manner to allow prediction, classification, resampling, interactive visualization etc. Just computing the surface on a grid of points is in general not satisfactory.

An algorithm for local fitting generally consists of a number building blocks. On top of basic fast algorithms, one may want additional features, such as choices in the order of fitting; robustness; variable bandwidths and application to a wide variety of settings such as regression, density estimation, local likelihood fitting and multivariate problems. Ideally, the basic computational algorithms should be as flexible to as many of these settings as possible. Methods that require a fixed bandwidth, or only work in one dimension, are of very limited utility.

For the statistician, it is important that procedures be implemented within the environments with which they are familiar, and that provide facilities for data management and interactive analysis and graphics and the like. A stand-alone C or Fortran program will certainly be much faster to execute than software incorporated in a statistical computing environment; however it is much less useful to the statistician because of the overhead required to manage data, interface with other systems for display etc.

We can now state why some of the 'fast' methods discussed in the statistics literature are essentially useless as general purpose computational algorithms. As particular examples, we focus on binning and updating methods, presented for example by Fan and Marron (1994). First we note the comparisons in that paper are essentially meaningless in light of the preceding discussion: Figure 8 of Fan and Marron (1994) compares times for local linear smoothing; one curve for a weighted smoother with robustness iterations running in the S environment while the methods promoted by Fan and Marron use a uniform kernel smoother without robustness iterations and compiled as a stand-alone C program. This comparison says nothing about the basic computational algorithms.

The binning method divides the predictor space into intervals or a grid, and assigns observations to grid points. For a large samples, this effectively reduces the sample size, thereby speeding up the computations. But the binning step is essentially a preliminary local constant fit; to avoid bias problems, large numbers of bins are required to avoid bias problems. This is particularly so in the multivariate case. Fan and Marron report the major speedups result from reducing the observations to an equally spaced grid, reducing the number of kernel evaluations. However, this only works with a constant bandwidth!

Updating methods are designed for polynomial weight functions, and expand sums such as those on the left hand side of (1) in powers of the X_i. The sums are then updated by moving to nearby x values. However, currently available implementations only address one dimensional problems, and the fastest implementations have serious stability problems.

Most importantly, both of these methods require solution of the optimization problem (1) for each fitting point. Hence, particularly for our density estimation setting, these methods are unlikely to be particularly fast. Also, the methods are geared towards evaluating the estimate on a predetermined set of points (for binning, on an equally spaced grid), which is not satisfactory for many statistical purposes such as prediction, classification, resampling and interactive graphics. Neither method provides a compact representation of the fitted surface.

3 Piecewise Polynomials

Suppose we have a sequence of vertices $v_0 < v_1 < \ldots < v_k$, and evaluations of a function $f(v_i)$ and its derivative $f'(v_i)$ at each vertex. There exists a unique C^1 function that matches the function values and derivatives at each vertex, and is piecewise cubic on each interval (v_i, v_{i+1}). This type of construction enables us to approximate smooth functions quite cheaply using Hermite polynomials. See De Boor (1978, Chapter 4) for further discussion of this scheme.

Another closely related method is the cubic spline approximation. In the most common use, the cubic spline method enforces continuity of the first and second derivatives; it is not required to match the true derivatives.

Several methods along these lines have been used in nonparametric function estimation. Penalized likelihood methods (Wahba, 1990) give rise to a cubic spline estimate with vertices at the data points. Regression splines, and the logspline density methods of Kooperberg and Stone (1991) estimate the parameters of a spline using criteria such as maximum likelihood; this enables the number of vertices to be substantially reduced. An alternative method is to estimate the parameters using local regression or likelihood methods. For this approach to be computationally efficient, it is important to make effective use of the vertices; for this reason the Hermite polynomial rather than cubic spline approach is preferable. This is the idea underlying the LOESS method (Cleveland and Grosse, 1991).

The computational advantage of local regression and likelihood appears in multidimensional cases. Global approaches to fitting generally involve the solution of a large scale optimization problem, which is generally very expensive. While sparse matrix techniques have been successfully applied to spline problems in one dimension, there use in multiple dimensions is much more difficult. By comparison, local regression methods solve a small optimization problem at each vertex.

Two fundamental problems remain: The construction of a suitable partition of a multidimensional domain, and the construction of interpolants over this partition. The most common types of partition involve rectangular cells or triangular cells.

A rectangular partition is often preferred in theoretical work, since it is often easier (or less difficult) to analyze, and define suitable polynomial approximations. The earliest work using local fits in this direction appears to be Silverman (1981) who constructed an estimate based on evaluation of a fixed bandwidth kernel estimate and its derivatives over a coarse grid, and used a piecewise quadratic scheme to interpolate over the cells of the grid.

A further development in this direction is the k-d tree structure introduced by Friedman, Bentley and Finkel (1977) and applied to local regression by Cleveland and Grosse (1991). In this algorithm, the data is initially bounded by a box, and the cells are recursively split. The splits in the k-d tree algorithm always divide observations in the parent cell into two subsets of approximately equal size, and hence the resultant structure has most vertices in regions of high point density. This is commensurate with the nearest neighbor bandwidths used in LOESS; however, other split rules could be used to adapt to other situations.

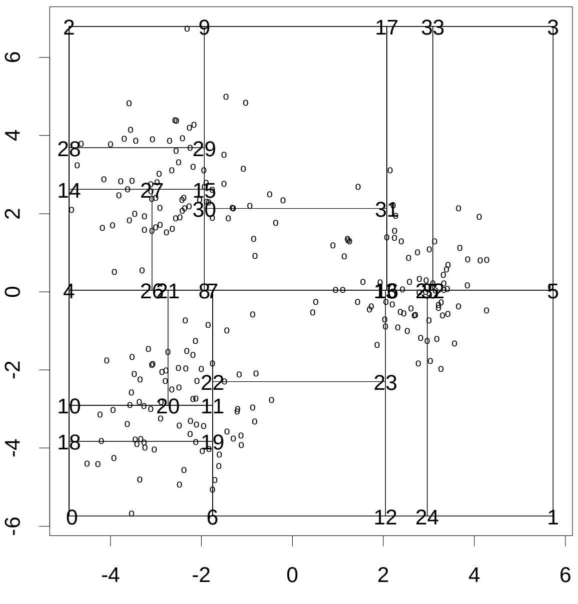

Figure 2: Division of 225 data points by a k-d tree with 34 vertices and 16 cells. Vertices are numbered in the order they are entered into the tree.

An example of a k-d tree is shown in Figure 2. As required, this partition has most vertices in regions of high density. One can also identify some weaknesses; in particular, occasionally vertices occur at very close points suggesting inefficiency. Also, construction of interpolants over a rectangle depends on more than just the corners; for example, interpolating over the rectangle $(10, 11, 18, 19)$ cannot ignore the evaluation at vertex 20. This problem becomes particularly complex in three or more dimensions.

An alternative to rectangular cells is partitions based on triangular cells. A triangle in d dimensions has $d+1$ vertices, against 2^d for cubes. This suggests there are *potential* savings in multiple dimensions. The word potential is stressed here, since there are both 'good' and 'bad' triangles.

Some examples of good and bad triangles are shown in Figure 3. The ideal triangulation would consist of equilateral triangles. In two dimensions, these can be tessalated; unfortunately this is non-adaptive and therefore may be wasteful in regions of low density where a large bandwidth must be used. The right angled triangle scores 'ok'; this triangle can be interpolated over with reasonable success, but we are unlikely to gain much over the use of rectangular grids. The tall isosceles triangle is poor, since some points may be far from the nearest vertex of the triangulation. Finally, the flat isosceles triangle scores bad; interpolating in the middle of the vertical edge will ignore the two side vertices.

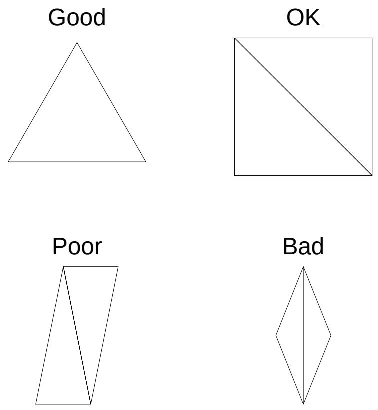

Figure 3: Examples of good and bad triangles in a triangulation.

In addition to requiring good triangles, there are several other competing criteria that must be considered when growing a triangulation. We wish to maintain adaptiveness; in particular, there should be more vertices in regions where small bandwidths are used. The triangles must be small enough for interpolants to work reasonably; however, the vertices should be sparse enough for efficiency. Finally, the triangulation must be fast to grow and search.

4 Recursive Partitioning

In this section, one possible triangulation scheme based on recursive partitioning is presented. First, draw a box around the data, and divide the box into two triangles by the addition of a diagonal. The triangulation is then grown recursively by adding vertices to existing edges. Suppose existing vertices are $v_1, \ldots, v_m$, and $h_1, \ldots, h_m$ are the bandwidths used at these vertices. For each edge (v_i, v_j) in the existing triangulation, assign a score:

$$\rho_{i,j} = \frac{\|v_i - v_j\|}{\min(h_i, h_j)}.$$

If $\rho_{i,j} > c$ split the edge. Let $\lambda = h_i/(h_i + h_j)$, and create a new vertex at $v_{m+1} = \lambda v_j + (1 - \lambda)v_i$. We then create the new edges joining points to v_{m+1}. This process is repeated until no more edges require splitting.

The use of bandwidths in determining the splits allows the algorithm to preserve adaptivity. If h_i is smaller than h_j,, then λ will be less than 0.5 and v_{m+1} is closer to v_i. In extreme cases this can over adapt, and so we restrict λ to the interval $(0.2, 0.8)$.

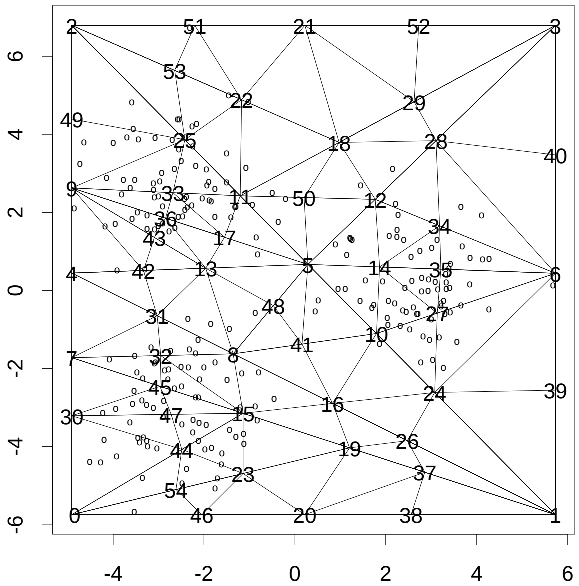

Figure 4: Recursive triangulation. Vertices are numbered in the order they were entered into the triangulation.

Figure 4 shows a recursive triangulation grown on the trimodal data in Figure 1. The vertices are numbered in the order they were entered into the triangulation. The bandwidth used is variable, covering 113 nearest neighbors for each fitting point. Some adaptivity can be seen in this picture: There is a greater density of vertices around the three peaks, where a smaller bandwidth is used for the fit.

The recursive partitioning presented here has both good and bad features. A big advantage is the triangulation can be stored in a tree-like structure; for any point x, it can then be rapidly determined which triangle contains x. The disadvantage is a recursive scheme scores only an 'OK' in terms of goodness of triangles, and so one can't expect substantial gains in efficiency. There are also some bad splits in the algorithm; for example, point 7 was used to split the triangle $(0, 1, 4)$; it would have been better for the algorithm to split the $(1, 4)$ edge first.

5 Finite Element Interpolants

The finite element method constructs interpolants over the cells of a partition. Only vertices on the boundary of a cell are used. This has computational advantages; the interpolants depend on only a small number of parameters and hence do not require solving large systems of equations.

The cell-based construction of finite elements leads to substantial difficulty in enforcing global smoothness conditions. For visualization purposes, we would like our surface to be continuous and differentiable. Also, the interpolant should be commensurate with our fitting procedure; for example, an interpolant that only reproduces linear polynomials is not adequate for use with a local quadratic or cubic fitting procedure.

A two dimensional element suitable for our purposes is the Clough-Tocher finite element (Clough and Tocher, 1965; Lancaster and Salkauskas, 1986). This method uses twelve pieces of information: The function values and derivatives at each vertex of the triangle, and the normal derivatives at the midpoint of each side. The Clough-Tocher finite element is then piecewise cubic over each of three sub-triangles, with continuity and differentiability conditions enforced at the interior seams. See Figure 5. A remarkable feature of the Clough-Tocher method is it produces a *globally* C^1 surface; when the method is applied independently on adjacent triangles, the resulting surface is differentiable at the common boundary!

The full twelve parameter Clough-Tocher method will reproduce a cubic polynomial. Unfortunately our local fitting procedure will not produce the normal derivatives at the midpoints of the sides; in practice, these are estimated by linear interpolation. This reduced nine parameter Clough-Tocher method reproduces all quadratic terms. A cubic reproducing scheme could be constructed by using second derivatives at the vertices and estimating normal derivatives using quadratic interpolation.

Figure 6 shows the Clough-Tocher method applied to the data from Figure 1 using the triangulation in Figure 4. Qualitatively the picture looks very similar to the direct fit; the three peaks are kept separate and reproduced

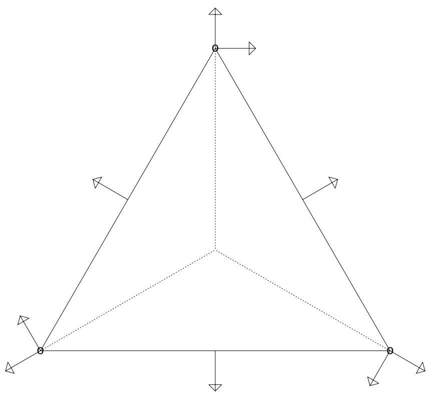

Figure 5: Clough-Tocher method. The interpolant uses function values at the three vertices of the triangle, and the nine directional derivatives indicated. The interpolant is made up of three piecewise cubics over the interior triangles.

to a similar height in both figures. Some differences are visible, particularly in the 0.01 contours.

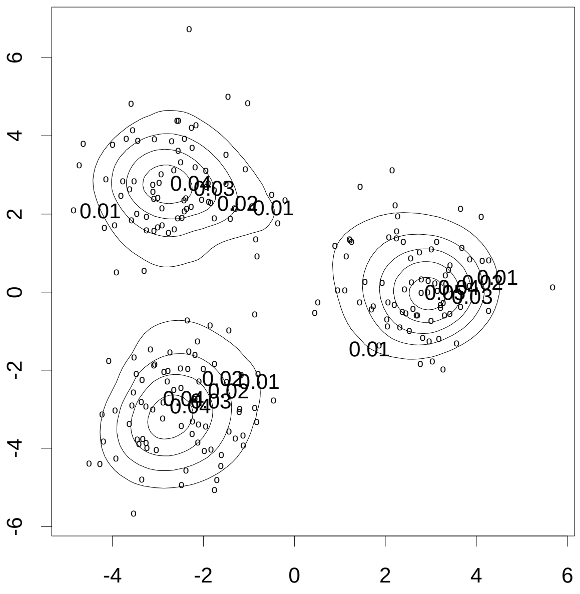

Figure 6: Density estimate using Clough-Tocher interpolation over a triangulation. Local cubic fitting.

Alternative constructions of interpolants can be based on piecewise quadratic, rather than cubic, polynomials. This has advantages noted by Silverman (1981) for producing contour plots since level sets can be readily found. Another advantage is in locating local maxima of the estimate. The difficulty is that a quadratic has fewer degrees of freedom, and additional internal seams must be introduced to enforce boundary constraints. For example, Silverman divides each cell into sixteen triangles. For a construction of a C^1 quadratic surface on a triangular partitions, see section 6.2 of Chui (1988).

	No. Vertices	$d(\tilde{f}, \hat{f})$	$d(\tilde{f}, f)$
Direct	225	-	0.261
Triangulation	55	0.0296	0.269
k-d tree	34	0.0638	0.282
k-d tree	66	0.0345	0.264

Table 1: Comparison of direct fitting with triangulation and k-d tree based interpolation schemes.

6 Concluding Remarks

The triangulation approach is competing with the k-d tree as an approximation method. A natural question is to try to make an objective comparison of their performance. We consider 'comparable' estimates to consist of the same or similar numbers of vertices. The real question of course is how well do the interpolated estimates approximate the true estimate. In practice we cannot measure this, so we also consider how well the interpolated estimate approximates the direct. We consider the criterion

$$d(\tilde{f}, \hat{f}) = \frac{1}{n} \sum_{i=1}^{n} |\log \tilde{f}(X_i) - \log \hat{f}(X_i)|$$

where $\hat{f}(x)$ is the direct density estimate and $\tilde{f}(x)$ is the interpolated density estimate. where $\hat{f}$ is either density or log-density; $\hat{f}_0$ is direct and $\hat{f}_1$ is triangulation or kdtree estimate. This is also an approximation to L_1 distance $\int |\tilde{f}(x) - \hat{f}(x)| dx$ between the two estimates.

Table 1 shows the results for our triangulation and two different sizes of k-d tree. As we would hope, the 34 vertex k-d tree is substantially beaten. The triangulation also slightly beats the 66 vertex k-d tree compared to the direct estimate, but loses slightly compared to the true density. Of course, not too much should be concluded from one example selected by the author; however we believe this certainly gives grounds for optimism.

The triangulation used in section 4 is based on recursive partitioning. This enables the triangulation to be stored as a tree type structure, which facilitates rapid searching; in particular, one can rapidly determine which triangle contains an arbitrary point in the domain. There are however some disadvantages; in particular, we have noted the resulting triangulation will generally only score 'OK' on the scale of Figure 3. We have also given up another major advantage of triangulations; namely the ability to well approximate fairly arbitrary nonlinear boundaries and domains.

There are alternative methods of growing triangulation, any of which may have advantages and disadvantages in our application. An obvious alternative is a sequential scheme, beginning with a seed triangle in the center of the data, and adding neighboring triangles until the domain is filled. This loses the tree structure of our recursive scheme. Some advantages are that 'good' triangulations may result more readily, and easier adaptability to non-rectangular boundaries.

Another step that might be considered is to attempt to optimize the triangulation in some sense. While a full blown optimization is the type of expensive problem we seek to avoid, some improvement may be obtained by moving vertices around or constructing triangulations on a given set of vertices. Some discussion of such schemes can be found in Barnhill (1977).

There are a number of directions in which this work can be extended. Perhaps the most obvious is beyond two dimensions, where as already noted there is potential for triangulations to be much more efficient than rectangular grids.

An obvious extension of this work (indeed the original motivation) is to higher dimensions, where the potential saving of triangular cells over rectangular by reducing the number of vertices seems much greater. Of course, any nonparametric fitting becomes much more difficult beyond two dimensions due to data sparseness. However, the construction of good triangulations is more difficult; also difficult is the problem of enforcing global differentiability in finite element interpolants.

There are also some theoretical questions that could be pursued. One obvious question is to analyze how well interpolated methods perform as estimates; in particular, we hope to preserve good properties of local polynomial fitting. Such an analysis may also help suggest refinements to the triangulation; in particular, how deep should the triangulation be taken? Another question is how to estimate derivatives. The present implementation uses the local slopes from the local polynomial fits; this is convenient since the slopes are available at no extra computation costs. Theoretical analysis suggests these slopes have good properties as derivative estimates; however, not always at the same bandwidth as for estimating the function itself. The author has found this problem severe in some cases for local quadratic fitting; hence the preference for local cubic fitting in the examples in this paper.

Finally, we mention some ongoing work in nonparametric function estimation related to the topics discussed in this paper. Eric Grosse is considering alternative constructions of Loess estimates based on the k-d tree partition; in particular, placing vertices at the centers of the cells, rather than the corners. This solves the problem of nearby vertices mentioned in relation to Figure 2, and should improve efficiency of the scheme. Mark Hansen and Charles Kooperberg are using triangulation methods in a global likelihood estimation scheme, considering generalized vertex splines (Chui, 1988) and other methods to construct estimates.

REFERENCES

Barnhill, R. E. (1977). Representation and approximation of surfaces. In *Mathematical Software III*, Ed. J. Rice. Academic Press, New York.

Chui, C. K. (1988). *Multivariate Splines.* SIAM, Philadelphia.

Cleveland, W. S. and Grosse, E. (1991). Computational methods for for local regression. *Statistics and Computing* **1**, 47-62.

Clough, R. W. and Tocher, J. L. (1965). Finite element stiffness matrices for analysis of plates in bending. *Proc. Conf. Matrix Methods in Structural Mechanics*, Wright-Patterson A.F.B., Ohio.

de Boor, C. (1978). *A Practical Guide to Splines.* Springer Verlag, New York.

Fan, J. and Marron, J. S. (1994). Fast implementations of nonparametric curve estimators. *J. Comp. & Graph. Statist.* **3**, 35-56.

Friedman, J. H., Bentley, J. L. and Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software* **3**, 209-226.

Kooperberg, C. and Stone, C. J. (1991). A study of logspline density estimation. *Comp. Statist. & Data Anal.* **12**, 327-347.

Lancaster, P. and Salkauskas, K. (1986). *Curve and Surface Fitting: An Introduction.* Academic Press, London.

Loader, C. (1993). Local Likelihood Density Estimation. Available by ftp from `netlib.att.com` in the file `netlib/att/stat/doc/93/31.ps.Z`.

Silverman, B. W. (1981). Density Estimation for Univariate and Bivariate Data. In *Interpreting Multivariate Data*, 37-53. Ed. V. Barnett. Jon Wiley & Sons, Chichester.

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