Computational Methods for Local Regression

William S. Cleveland
E. Grosse

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Local regression is a nonparametric method in which the regression surface is estimated by fitting parametric functions locally in the space of the predictors using weighted least squares in a moving fashion similar to how a time series is smoothed by moving averages. Three computational methods for local regression are presented. (1) Fast surface evaluation is achieved by building a k-d tree in the space of the predictors, evaluating the surface at the corners of the tree, and then interpolating elsewhere by blending functions. (2) Surfaces are made conditionally-parametric in any proper subset of the predictors by a simple alteration of the weighting scheme. (3) Degree-of-freedom quantities that would be extremely expensive to compute exactly are approximated, not by numerical methods, but through a statistical model that predicts the quantities from the trace of the hat matrix, which can be computed easily.

1 Introduction

Local regression is a nonparametric approach to estimating regression functions, or surfaces (Macauley, 1931; Watson, 1964; Stone, 1977; Cleveland, 1979; Friedman, 1984; Hastie and Tibshirani, 1990; Cleveland and Grosse, to appear). Two examples are shown in Figures 1 and 2. In the first figure, there is one predictor, and the fitted function is the

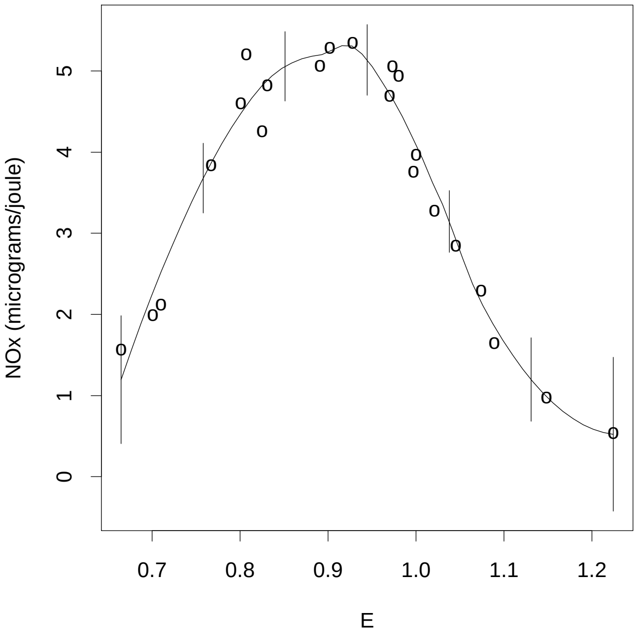

Figure 1: Local regression model with one predictor — fitted curve and 99% pointwise confidence intervals.

curve. In the second figure, there are two predictors, and the fitted surface is shown by a contour plot. These two examples will be explained in detail later.

We suppose the response and predictors are related by

$$y_i = g(x_i) + \epsilon_i,$$

for $i = 1$ to n, where y_i is the i-th observation of the response, x_i is the i-th observation of p predictors, g is the regression surface, and ϵ_i is a random error. If x is any point in the space of the predictors, $g(x)$ is the value of the surface at x. Predictors can be numerical variables or categorical. However, when categorical predictors are present, the observations are divided into subsets, one for each combination of levels of the categorical variables,

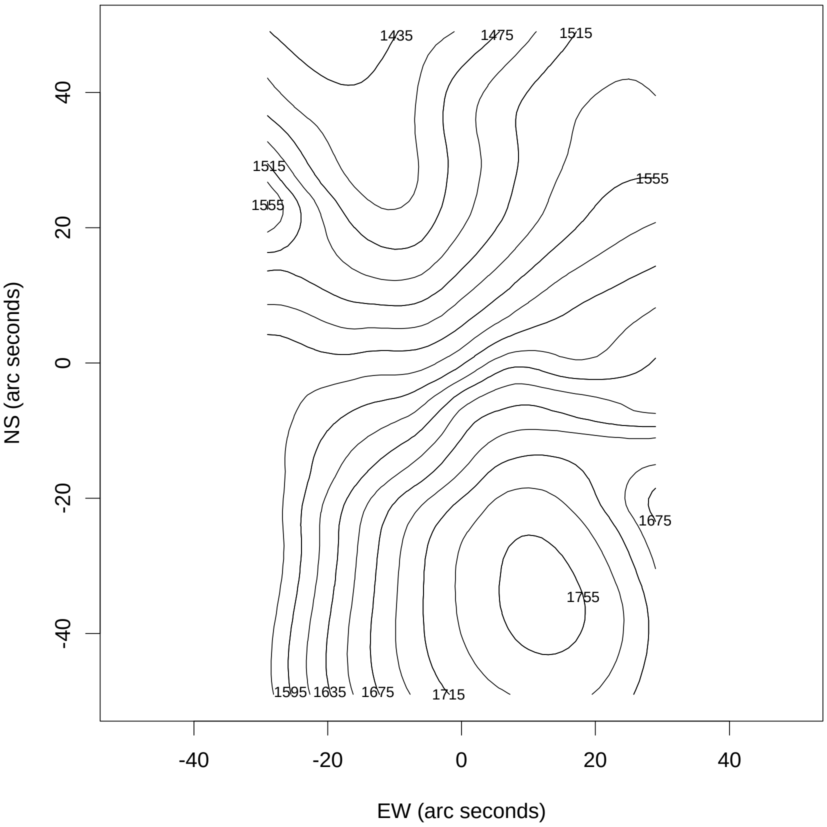

Figure 2: Local regression model with two predictors — contours of fitted surface.

and the response is fitted separately to the numerical predictors for each subset. Thus no additional computational issues arise as a result of categorical predictors, so in this paper we suppose all predictors are numerical.

In using local regression methods, we specify properties of g and ϵ_i, that is, we make assumptions about them. In practice, of course, it is important to verify these specifications using diagnostic methods (Cleveland, Grosse, and Shyu, 1991). There is a fundamental specification of g, however, that characterizes the approach as local regression; we suppose that for each x, the regression surface is well approximated in a certain neighborhood of x by a function from a parametric class. Here, we will allow the specification of one of two general classes of parametric functions: linear and quadratic polynomials. For example, suppose there are two predictors, u and v. If we specify linear, the class is made up of three monomials: a constant, u, and v. If we specify quadratic, the class is made up of five monomials: a constant, u, v, uv, u^2, and v^2. We will let λ denote the degree of the polynomial class and τ be the number of monomials.

The method for fitting local regression surfaces that is treated in this paper is *loess*, which is short for *local regression* (Cleveland, 1979; Cleveland, Devlin, and Grosse, 1988). Straightforward implementation of loess leads to an unrealistic computational burden. In this paper we present methods that make the method computationally practical, even in personal computer environments. In the remainder of this section we describe the computational problems and the contents of the paper.

1.1 Specifications, Fitting, and Inference

In Section 2 we describe specifications of local regression, the loess fitting method, and how inferences based on this fitting method are carried out. The details of the loess fitting method in a particular application are determined by the specifications of the surface and error terms, but the general approach is quite simple; g is estimated at x by fitting a linear or quadratic polynomial using weighted least squares, where the weight for the observation (x_i, y_i) decreases as the distance of x from x_i increases.

1.2 Surface Computation

In principle, loess fitting involves a weighted least-squares computation at each point where the surface is to be evaluated; for example, to make the contour plot in Figure 2, the loess estimate was evaluated at 5841 values. Typically, direct evaluation at all points is too expensive in the computing environments that most practitioners use today. In Section 3 we present a computational method for evaluating a loess fit. A set of points, typically small in number, is selected for direct computation using the loess fitting method, and a surface is provided by interpolation. The space of the predictors is divided into rectangular cells using an algorithm based on *k-d* trees (Friedman, Bentley, and Finkel, 1977), the loess fit is evaluated at the vertices of the tree, and then blending functions (Gordon, 1969) do the interpolation.

Suppose there are two or more predictors. One specification we can impose on g is that the surface be *conditionally-parametric* in a proper subset of the predictors (Cleveland, Grosse, and Shyu, 1991). This means that given the values of the predictors not in the subset, the surface is a member of a parametric class as a function of the subset. If we change the conditioning values, the surface is still a function in the same class, although the parameters might change. For example, suppose the predictors are u and v. Suppose $\lambda = 2$, and we specify the surface to be conditionally-parametric in u. Then given v, the surface is quadratic in u; the general form of the surface in this case is $h_0(v) + h_1(v)u + h_2(v)u^2$. It makes sense to specify a surface to be conditionally-parametric in one or more predictors if exploration of the data or *a priori* information suggests that the surface is globally a very smooth function of those predictors. Making such a specification when it is valid can result in a more parsimonious description of a regression surface.

If we have specified a surface to be conditionally-parametric, we need a computational procedure to ensure that the fitted surface is conditionally-parametric. In Section 3 we present a method for altering the basic loess computational method to accomplish this.

1.3 Computing Degrees-of-Freedom

Associated with a loess fit is an $n \times n$ matrix, L, that takes the vector of observations of the response, y_i, to the vector of fitted values, $\hat{y}_i = \hat{g}(x_i)$, where $\hat{g}$ is the fitted loess surface. Following the terminology for least squares fitting, we refer to L as the *hat matrix*. Inferences about g utilize two quantities,

$$\delta_k = \mathrm{tr}[(I - L)'(I - L)]^k$$

for $k = 1$ and 2. δ_1 and δ_2 are used in the computation of degrees of freedom for t-intervals for $g(x)$ and for F-tests to compare two loess fits. A supercomputer environment would be needed to do exact computations of the δ_k routinely. In Section 4 we present a method of approximation. Instead of using tools of numerical analysis, as we do to approximate the fitted surface, we take a statistical approach. We generated a large number of data sets, each with a response and one or more predictors, and computed the δ_k for each. Then we fitted a semi-parametric model to the δ_k based on predictors that can be computed cheaply. The central predictor is the trace of L.

1.4 Discussion

In the final section of the paper we cite other work on computational methods for nonparametric regression. Also, information is given on electronic access to Fortran routines that implement the computational methods of this paper.

2 Local Regression: Specifications, Fitting, and Inference

2.1 Local Regression

In carrying out local regression we specify properties of the surface and errors, that is we make certain assumptions about them. In all cases we suppose the ϵ_i are independent random variables with mean 0. We can specify a probability distribution for the ϵ_i. Here, we treat just one case; we assume the ϵ_i are independent normal random variables with variance σ^2. Computationally, other cases typically involve iterations of normal-error estimates.

As we stated earlier, we suppose that in a certain neighborhood of x, the regression surface is well approximated by linear or quadratic polynomials. Thus we must specify the degree, λ. Sometimes, it is useful to specify an intermediate model between $\lambda = 1$ and $\lambda = 2$ by discarding squares of some predictors (Cleveland, Grosse, and Shyu, 1991), but we do not consider this specification here. The overall sizes of the neighborhoods are specified by a parameter, α, that will be defined in Section 2.2. Size, of course, implies a metric, and we will use Euclidean distance, but a variety of metrics in the space of the raw data can be achieved by transformation of the original variables, which amounts to a change in the definition of the x_i. At the very least, we typically want to standardize the scales of the predictor observations. Finally, we can specify a subset of the predictors to be conditionally-parametric.

2.2 Loess

The loess fitting method is a definition of $\hat{g}(x)$, the estimate of g at a specific value of x. The smoothness of the loess fit depends on the specification of the neighborhood parameter, α, the specification of λ, and the specification of conditionally-parametric predictors.

Let $\Delta_i(x)$ be the Euclidean distance of x to x_i in the space of the predictors. Let $\Delta_{(i)}(x)$ be the values of these distances ordered from smallest to largest, and let

$$T(u) = \begin{cases} (1-u^3)^3 & \text{for } 0 \le u < 1 \\ 0 & \text{for } u \ge 1 \end{cases}$$

be the *tricube weight function.*

The neighborhood parameter, α, is positive. Suppose $\alpha \le 1$. Let q be αn truncated to an integer. We define a weight for (x_i, y_i) by

$$w_i(x) = T\left(\frac{\Delta_i(x)}{\Delta_{(q)}(x)}\right).$$

For $\alpha > 1$, the $w_i(x)$ are defined in the same manner but $\Delta_{(q)}(x)$ is replaced by $\Delta_{(n)}(x)\alpha^{1/p}$, where p is the number of predictors. The $w_i(x)$, which we will call the *neighborhood weights*, decrease or stay constant as x_i increases in distance from x.

If we have specified λ to be 1, a linear polynomial in x is fitted to y_i using weighted least squares with the weights $w_i(x)$; the value of this fitted polynomial at x is $\hat{g}(x)$. If λ is 2, a quadratic is fitted.

2.3 Statistical Inference

The normal-error loess estimate, $\hat{g}(x)$, is linear in y_i:

$$\hat{g}(x) = \sum_{i=1}^{n} l_i(x) y_i,$$

where the $l_i(x)$ do not depend on the y_i. Suppose diagnostic methods have revealed that the specifications of the surface have not resulted in appreciable lack of fit of $\hat{g}(x)$; we take this to mean that $E\hat{g}(x) - g(x)$ is small. Suppose further that diagnostic checking has verified the specifications of the error terms in the model. Then the linearity of $\hat{g}(x)$ results in distributional properties of the estimate that are very similar to those for classical parametric fitting. We will briefly review the properties.

Since $\hat{g}(x)$ is linear in y_i, the fitted value at x_i can be written

$$\hat{y}_i = \sum_{j=1}^{n} l_j(x_i) y_j.$$

Let L be the matrix whose (i,j)th element is $l_j(x_i)$ and let

$$\bar{L} = I - L$$

where I is the n by n identity matrix. For $k = 1$ and 2, let

$$\delta_k = \mathrm{tr}(\bar{L}'\bar{L})^k.$$

We estimate σ by the scale estimate

$$s = \sqrt{\frac{\sum_{i=1}^{n} \hat{\epsilon}_i^2}{\delta_1}}$$

from the residuals $\hat{\epsilon}_i = y_i - \hat{y}_i$. The standard deviation of $\hat{g}(x)$ is

$$\sigma(x) = \sigma \sqrt{\sum_{i=1}^{n} l_i^2(x)}.$$

We estimate $\sigma(x)$ by

$$s(x) = s \sqrt{\sum_{i=1}^{n} l_i^2(x)}.$$

Let

$$\rho = \frac{\delta_1^2}{\delta_2}.$$

The distribution of

$$\frac{\hat{g}(x) - g(x)}{s(x)}$$

is well approximated by a t-distribution with ρ degrees of freedom; we can use this result to form confidence intervals for $g(x)$ based on $\hat{g}(x)$.

We can use the analysis of variance to test a null loess fit against an alternative one. Let the parameters of the null fit be $\alpha^{(n)}$, $\lambda^{(n)}$, $\delta_1^{(n)}$, and $\delta_2^{(n)}$. Let the parameters of the alternative fit be α, λ, δ_1, and δ_2. For the test to make sense, the null model should be *nested* in the alternative (Cleveland, Grosse, and Shyu, 1991). Let rss be the residual sum of squares of the alternative model, and let $rss^{(n)}$ be the residual sum of squares of the null model. The test statistic, which is analogous to that for the analysis of variance in the parametric case, is

$$F = \frac{(rss^{(n)} - rss)/(\delta_1^{(n)} - \delta_1)}{rss/\delta_1}.$$

F has a distribution that is well approximated by an F distribution with denominator look-up degrees of freedom ρ, defined earlier, and numerator look-up degrees of freedom

$$\nu = \frac{(\delta_1^{(n)} - \delta_1)^2}{\delta_2^{(n)} - \delta_2}.$$

2.4 Examples

The data in Figure 1 are from an industrial experiment studying exhaust from an experimental one-cylinder engine (Brinkman, 1981). The dependent variable, NO_x, is the concentration of nitric oxide, NO, plus the concentration of nitrogen dioxide, NO_2, normalized by the amount of work of the engine. The predictor is the equivalence ratio, E, at which the engine was run; E is a measure of the richness of the air and fuel mixture. For the loess fit shown in the figure, λ is two and α is $2/3$.

The surface in Figure 2 was fitted to measurements of the radial velocity of the galaxy NGC7531 (Buta, 1987). Figure 3 shows locations where 323 measurements were made. The positions have been jittered slightly to reduce overplotting. The vertical and horizontal axes are north-south and east-west position, respectively, on the celestial sphere. For the loess fit shown in the figure, λ is two and α is 0.2.

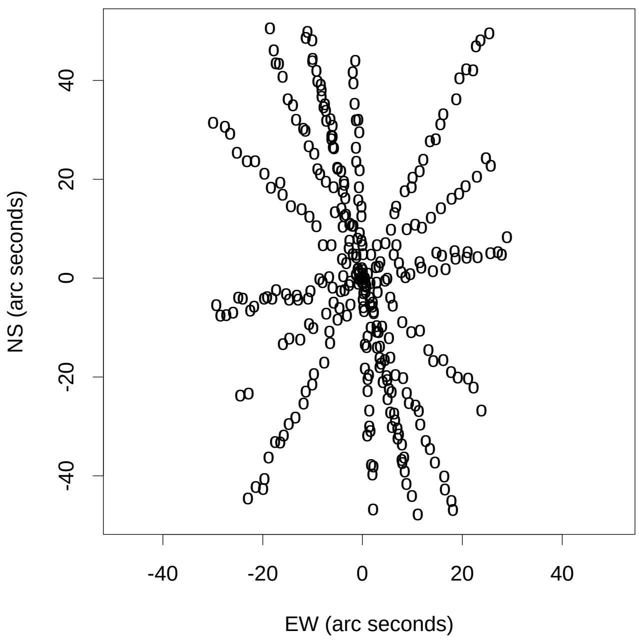

Figure 3: Galaxy data — locations of velocity measurements.

3 Surface Computation

The strategy of our algorithm is to select a few places to do the loess calculation directly and interpolate elsewhere. Since the surface, $\hat{g}$, is smooth by construction, it should be plausible that one can directly calculate $\hat{g}$ on a coarse grid and interpolate cheaply to a finer grid for graphics or other purposes. It pays to use an adaptive grid that takes account of the location of the x_i, rather than a uniform rectangular one, because loess fitting adapts in such a way due to the nearest-neighbor feature. The adaptation is achieved by using *k-d* trees (Friedman, Bentley, and Finkel, 1977) to partition the domain into cells. Then blending functions (Lancaster and Šalkauskas, 1986) are fitted to the cells to give a C^1 surface. This algorithm is apparently new, although it resembles transfinite elements on irregular grids (Birkhoff, Cavendish, and Gordon, 1974; Cavendish, 1975) and regression trees (Freidman, 1979) in some respects. The algorithm may be viewed as belonging to the category of two-stage methods (Schumaker, 1976), in which typically one maps scattered data onto a rectangular grid and then uses a grid-based interpolation scheme.

In this section, we first discuss certain numerical and algorithmic issues that need careful attention to evaluate the surface at a single point, then we describe the *k-d* tree and blending methods, then we treat performance, and finally we describe a procedure to modify the loess fitting method to produce a conditionally-parametric surface.

3.1 Least Squares

To get the value of the loess surface, $\hat{g}(x)$, at a particular point x, the computational steps are the following: identify neighbors of x by calculating n distances, sort, then solve a q by τ linear least squares problem. Note that, unlike some other nonparametric methods such as smoothing splines (Wahba, 1978), evaluation of a loess surface implies independent computation for distinct x, allowing large speed-up on parallel computers. For each x, the dominant cost is a classic linear algebra problem for which well-vectorized subroutines are widely available.

Even with an efficient $O(n \log n)$ procedure, the sorting to get $\Delta_{(i)}(x)$ can become a bottleneck, slowing down the overall computation noticeably. A little time can be saved by avoiding the square roots and instead sorting $\Delta_i^2(x)$. On some machines, a little more time can be saved by arranging the distance calculation so that it vectorizes. A much more significant speedup is achieved by applying a classic algorithm (Floyd and Rivest, 1975) for selecting the q-th largest in an unordered list.

If we have found the neighborhood for a point x and next need the neighborhood for a nearby point x', can we do better than starting from scratch? One possibility would be to somehow take advantage of the triangle inequality, only updating the distances for points near the first neighborhood boundary. We have pursued another approach. During the partial sort we update a permutation vector instead of explicitly reordering the distance vector. By using the final permutation for x rather than the identity when starting the partial sort for x', fewer comparisons and swaps may be needed.

What happens if there are several points on the boundary of the neighborhood? That is to say, suppose there are distinct i and j such that $\Delta_{(j)}(x)=\Delta_{(i)}(x)=\Delta_{(q)}(x)$. The procedure just described breaks ties arbitrarily and hence selects at random just enough points on the boundary to bring the total to q. Since the tricube weight function drops to 0 at the boundary, this is justified; boundary points make no difference in the least squares local fit. In implementing a uniform weight function, in contrast, more care would be needed to ensure that all points on the boundary receive equal treatment.

Badly distributed observations may not uniquely determine all τ monomial terms in the local regression. Consider locally-quadratic fitting in one predictor. If, because of multiplicities, there are only two distinct sample locations inside a neighborhood, then a quadratic polynomial is not uniquely determined. In more variables, something similar can occur if there are patterns in the distribution of predictors. The most common such situation involves a discrete variable, so that a neighborhood that otherwise seems large enough contains just points lying on one or two lines. Again there is no unique local quadratic. In such a situation, we use the pseudoinverse to produce stable values for $\hat{g}(x)$. Since q is typically much larger than τ, a preliminary factorization $X = QR$ into a τ by τ matrix R and a q by τ matrix Q with $Q'Q = I_\tau$ followed by SVD of R allows the pseudoinverse to be computed efficiently (Chan, 1982). The interpolation scheme to be described later uses not only $\hat{g}(x)$, but also the slope of the local model at x. The pseudoinverse has an effect like dropping the square term in the transverse predictor. This is the best that can be automatically done with the available local information, but does imply the interpolation has to work with less information. It is wise in such circumstances to make a surface plot and verify that no undesired flat spots result. Of course, the data analyst can eliminate the use of the pseudoinverse by adding rows to X (increasing α) or removing columns from X

(reducing λ or specifying conditionally-parametric predictors).

It is possible to save linear algebra cost for multiple data sets at the same sample points with the same α, at the cost of increased workspace requirements, by saving the pseudoinverse coefficients. In the implementation of our computational methods, we provide this option. Associated with each vertex in the k-d tree is a τ by q coefficient matrix and a length-q integer vector.

3.2 k-d Trees

The k-d tree is a particular data structure (Friedman, Bentley, and Finkel, 1977) for partitioning space by recursively cutting cells in half by a hyperplane orthogonal to one of the coordinate axes. It was originally designed for answering nearest-neighbor queries but we use it in a different way. (Ironically, the tree is not of assistance in identifying the neighborhoods. The trouble is that α is commonly large enough that each neighborhood intersects most of the cells in the k-d tree.)

Let C be a rectangular cell containing the x_i and let h be a number between 0 and 1. To build the k-d tree, call *partition*(C), the recursive function described informally by

partition(C)

  $j :=$ maximizer-of$(\max_{x \in C}[x]_j - \min_{x \in C}[x]_j)$; component with greatest spread

  $\mu :=$ median$\{[x]_j : x \in C\}$;

  cut $C =: L \cup R$ by the hyperplane that intersects j-th axis at μ;

    left subcell $L \subseteq \{x \in C : [x]_j \leq \mu\}$;

    right subcell $R \subseteq \{x \in C : [x]_j > \mu\}$;

  if size$(L) > nh$

    partition(L);

  if size$(R) > nh$

    partition(R);

Thus h controls how far to refine the partition; if more than hn points are inside the cell, we refine. Larger values make the method run faster by generating fewer vertices; smaller values yield a blending surface closer to the direct surface. Also, as h increases, the tree tends to change only at values of h around a power of 1/2, provided the number of points on cell boundaries is small, because the number of points inside a cell tends to be around $n/2^k$ after the k-th round of partitioning.

The procedure is illustrated in Figure 4 where the circles graph x_i with two predictors, with $n = 100$, and with $h = 0.05$. Consider the top panel of the figure. First a bounding rectangle is drawn which encloses the sample points. This first cell is taller than it is wide, so it is bisected at the median of the second predictor; this results in the horizontal line inside the box. Then the two cells are considered. Each is wider than it is tall, so each is bisected at median values of the first predictor. The result at this point, is the four cells shown in the top panel. The final tree is shown in the bottom panel.

For readers more familiar with quadtrees, which bisect simultaneously at the midpoint in all the coordinates, it is worth noting that the *k-d* cells do not all have the same shape. In compensation, the cells do not vary widely in the number of points they contain; a quadtree can contain mostly empty cells if the samples are non-uniformly distributed.

An additional stopping criterion is needed in a Fortran implementation. Only a fixed size array is available for storing vertices, so there may be no room to continue refining the tree. For this reason, we refine in breadth-first order so that stopping at any time spreads the error uniformly. Ordinarily, *k-d* tree algorithms have been implemented in depth-first order, but breadth-first is no harder.

Once the *k-d* tree is built, $\hat{g}(x)$ is computed directly at the vertices. By "vertex", we just mean a corner of a cell; "vertex" seems a better term because a corner of one cell typically lies in the middle of a side of an adjacent cell. In addition to computing $\hat{g}(x)$ at the vertices, derivatives of $\hat{g}$ are estimated there by taking the slopes of the locally-linear or locally-quadratic fit. These slopes are a natural by-product of the least squares computation and cost nothing extra to obtain.

The k-d tree is designed to adapt to very non-uniformly distributed data. Nevertheless, if the samples are scattered in an elliptical cloud, we could help the tree along in the early stages by first rotating the points to bring the principal axes into alignment with the coordinate axes. (It is apparent from the definition that $\hat{g}$ itself is rotationally invariant.) An even more sophisticated rotation would align principal axes of clusters in the data (Art, Gnanadesikan, and Kettenring, 1982). Rarely do we bother with such rotations.

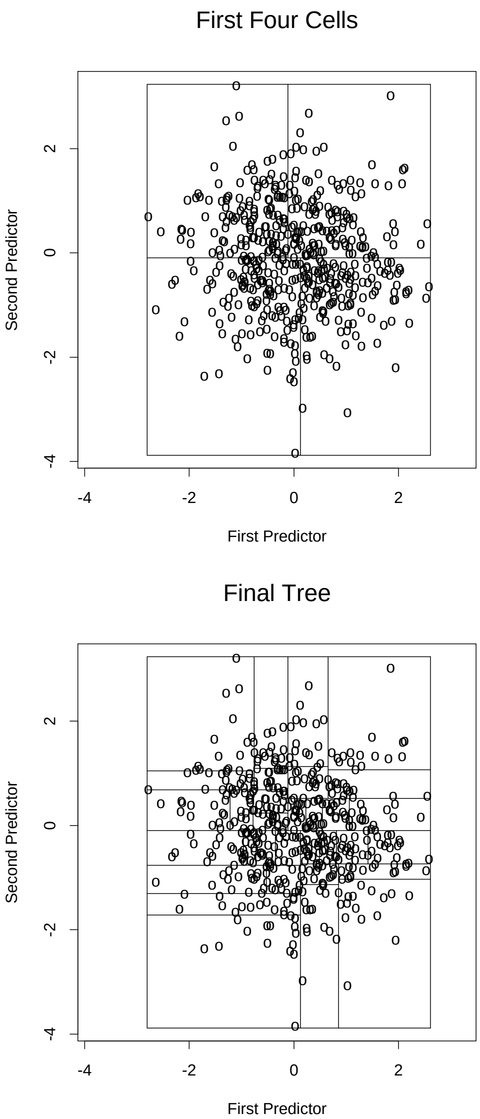

Figure 4: A k-d tree.

We hope the number of vertices will be much smaller than n. This is at least true asymptotically, because the number of cells needed to properly fit the data depends on the smoothness of $g(x)$, not n. In Figure 4 there are 66 vertices, so we solve just 66 least squares problems instead of, say, the thousands necessary to draw a contour plot.

Because of symmetries in the sample locations, it may happen that when building the k-d tree, two adjacent cells are cut in the same direction and at the same value. This introduces a double vertex on the shared cell side. A good rule of thumb in computational geometry is never to compute anything twice. Consistency rather than optimization is the crucial reason, and in this case we also wish to avoid double counting when computing $\mathrm{tr}(L)$ later. So before introducing a vertex we check the neighboring cell and use a pointer to an existing vertex if possible.

3.3 Blending

We now sketch the scheme used to interpolate based on the computation of $\hat{g}$ at the cells. We use a method called *blending* or *Coon's patch* or *transfinite interpolation* developed in the automobile industry (Gordon, 1969) for handling cross sections of clay models of a car. It is a method for generating smooth surfaces given function data along grid lines.

It is helpful to think of the case $p = 2$. Each cell boundary consists of four or more segments that meet at vertices. On each segment, function values are interpolated using the unique cubic polynomial determined by the function and derivative data at the vertices. Normal derivatives are interpolated linearly along the segment. In this way, we obtain function and derivative values all along the cell boundaries. Values are consistent from segment to segment and from cell to cell.

Finally, blending functions interpolate across the cell proper. This technique takes a certain combination of univariate interpolants in each variable separately to build, in our application, a surface with continuous first derivative. Although the calculation is not actually done this way, each cell is implicitly subdivided and on each piece a cubic polynomial is constructed.

We estimate partial derivatives at the vertices by slopes of the local model. This is quick but not the same as taking true derivatives of the underlying direct loess surface. It is interesting to note that with just a few more backsolves but no extra matrix factorizations, the correct derivatives can be obtained (Lancaster and Šalkauskas, 1986, §10.4).

Our implementation continues to use slopes rather than the true derivatives because we have not found any examples where the approximation leads to a poor smooth. In fact, as presently defined, the direct smooth $\hat{g}$ will have discontinuous derivatives resulting from the discontinuous nature of the neighborhood radius. The radius would need to be smoothed somehow before exact derivative formulas would be useful.

A related issue is that we implicitly use cubic interpolation with zero values for the cross derivatives $\partial_{ij} f$ at the rectangle corners. This is satisfactory for our purposes but produces flat spots that might be visible in a carefully rendered image of the surface, particularly if there are reflection lines that happen to pass near the corners. For a discussion of this phenomenon, see (Barnhill, 1977; Farin, 1990). If desired, it would be possible to provide correct cross derivatives.

A special problem arises for higher dimensions that does not occur for $p \leq 2$. Introducing vertices only on the sides of a cell can lead to adjacent cells in which sides introduced by independent cuts intersect. This is illustrated in Figure 5. It is necessary to add all such

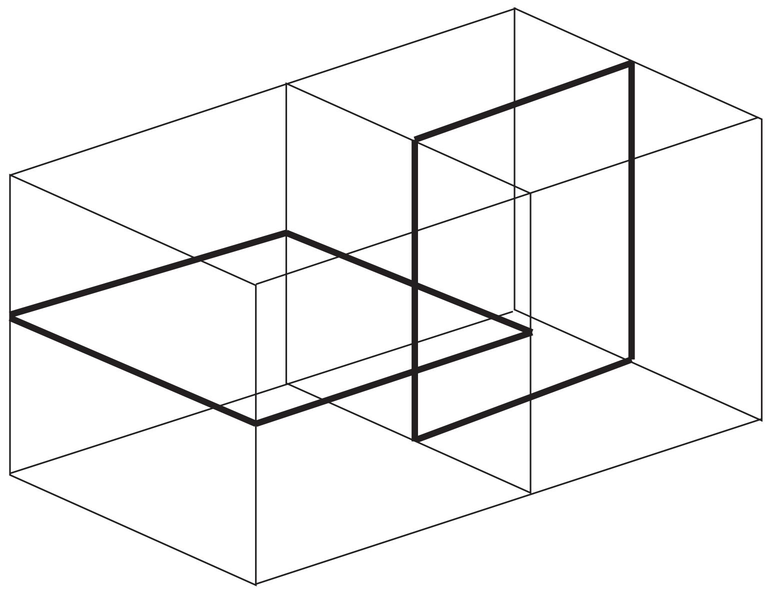

Figure 5: In three dimensions, extra vertices are introduced by division of adjacent cells.

points of intersection as vertices; otherwise, independent interpolation along the two edges gives inconsistent values at the common point.

The blending method is based on cubic polynomial interpolation and it is well-known that polynomial extrapolation is dangerous. So our computer implementation refuses to evaluate $\hat{g}(x)$ for x outside the bounding rectangle of the observations. To avoid nuisances caused by rounding, the bounding box is expanded by 1% in each variable.

3.4 Statistical Quantities

To carry out the computational method of the next section we need the diagonal of the hat matrix, L. The linear dependence of blending coefficients at each vertex on q nearby data values can be computed from least squares system, $X\beta = Wy$. Since the blending interpolant at an arbitrary point x ultimately is just a linear combination of vertex data, it is only a modest amount of extra effort to accumulate any element of L.

If we have just smoothed at a point x_i then we have a singular value decomposition $X = U\Sigma V'$. The smooth value $\hat{y}_i$ is just β_1, the coefficient of the constant term, and this is linearly related to the data y via the i-th row of L

$$L_{i,\cdot} = V_{1,\cdot}\Sigma^{+}U'W. \tag{1}$$

A few backsolves and an inner product suffice to compute this row.

The calculation for the interpolation method is only slightly more complicated. Note that smoothing at vertex j uses a linear combination of data values which we may write as $\beta_j = F_j y$ and blending uses a linear combination of vertex data. So there is a matrix F composed of the F_j and a (sparse) matrix B such that $L = BF$. A diagonal element, $L_{ii} = e_i' BFe_i$, may be computed by using the same interpolation algorithm on "vertex data" $F_j e_i$ and summing over j.

3.5 Performance

Since the scheme is local and reproduces quadratics, it can be shown that m leaf cells can achieve overall error $O(m^{-3})$. Hence as $n \to \infty$ and for a fixed error tolerance, the number of vertices is bounded. Partial sorting and linear algebra are each $O(n)$ and blending at each sample point involves a tree walk and fixed arithmetic. So overall runtime is proportional to the number of data points. Since the constant involves the number of k-d tree vertices, one needs to be cautious in trying to use this result as the basis for comparison with other computational methods. However, it is a basis for optimism when attempting to analyze huge data sets.

Profiling a typical computation revealed that 30% of the time was spent in linear algebra, 30% in building the k-d tree and interpolating by blending, 20% in sorting, and 20% in reading the input data. Total time for smoothing 600 points with $p = 2$, $\lambda = 2$, $\alpha = .5$ is under ten seconds on current workstations.

We have presented the loess interpolation method as an approximation to the direct loess method. But the interpolation method is a perfectly sensible nonparametric regression procedure in its own right. If we choose the cell-size parameter, h, to be small, then we get a close approximation of the direct fit; more specifically, in our experimentation we have found that taking h to be $\alpha/5$ typically results in a close approximation. As h increases, the interpolation surface typically becomes smoother and departs more and more from the direct surface. But for practice, the important question is how well the interpolation surface fits the data, not how well the interpolation surface approximates the direct one; thus diagnostic methods target the question of surplus and lack of fit of the interpolation surface. Also, the interpolation surface has a hat matrix, L, and inferences, of course, are based on it and not the hat matrix of the direct surface. Still, it is comforting that the interpolation surface can give a good approximation of the direct, since the direct method has been extensively investigated and its properties are well understood.

We will use the galaxy data and NO_x data sets discussed in Sections 1 and 2 to illustrate the relationship of the direct and interpolation surfaces as h changes. For the galaxy data, the spindly configuration of the points in the space of the predictors presents a challenge to the interpolation method to approximate the direct surface. For the fit shown in Figure 2, $\lambda = 2$, $\alpha = 0.2$, and $h = 0.2/5 = 0.04$, the standard value. Let us consider values of h of the form $0.95\ (1/2)^k$ for $k = 1$ to 6. This spans a large range; the largest value, 0.475, results in a tree with just 4 cells, and the smallest value, 0.015, is less than the standard value of $h = 0.04$. Also, any value of h in this range yields a tree that is the same as that for one for the six values we have selected. On the top panel of Figure 6, the maximum absolute difference of the direct and interpolation surfaces evaluated at the x_i is graphed against $\log_2 h$. The vertical dashed line segment shows the standard value of h. For this value, the absolute difference is 16 km/sec, which is an insignificant departure in this application. Most importantly, diagnostics showed the standard surface fitted the data.

The bottom panel shows a similar analysis for the NO_x data, where $\alpha = 2/3$, $\lambda = 2$, and $h = \alpha/5$. For the standard surface, the maximum absolute difference is 0.022 μ/j. This is 0.4% of the range of the fitted values, so if we added the direct curve to Figure 1, the two curves would not be visually distinguishable.

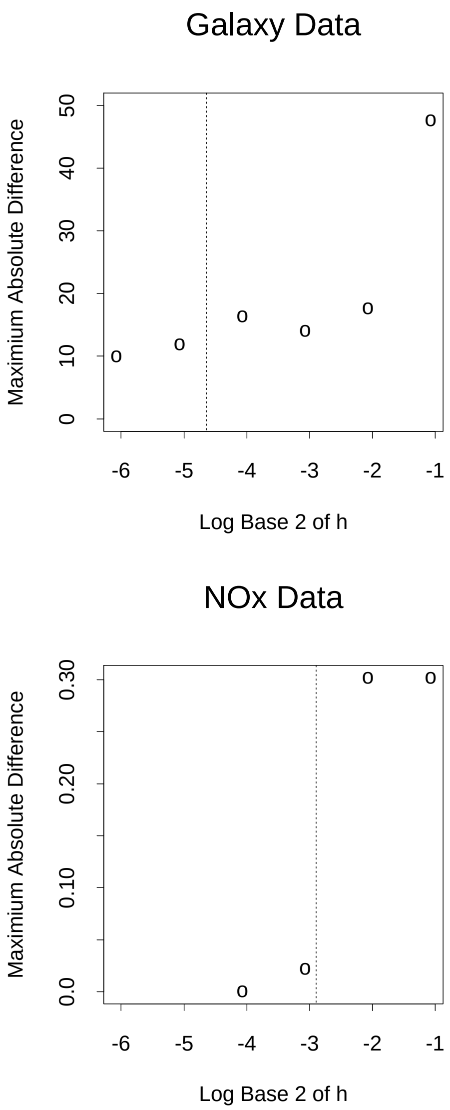

Figure 6: Comparison of direct and interpolation methods for galaxy data.

3.6 Conditionally-Parametric Surfaces

The method for making a loess fit conditionally-parametric in a proper subset of the predictors is simple. The subset is ignored in computing the Euclidean distances that are used in the definition of the neighborhood weights, $w_i(x)$. Let us use an example to explain why this produces a conditionally-parametric surface. Suppose there are two predictors, u and v, and $\lambda = 2$. Suppose we specify u to be a conditionally-parametric predictor. Since the weight function ignores the variable u, the i-th weight, $w_i(u, v)$, for the fit at (u, v), is the same as the i-th weight, $w_i(u + t, v)$, for the fit at $(u + t, v)$. Thus the quadratic polynomial that is fitted locally for the fit at (u, v) is the same as the quadratic polynomial that is fitted locally for the fit at $(u+t, v)$, whatever the value of t. Thus for the fixed value of v, the surface is exactly this quadratic as a function of the first predictor.

4 Computing Degrees-of-Freedom

In this section we present a method for computing δ_1 and δ_2 approximately. To make notation and formulas somewhat simpler, we will work with $\gamma_k = n - \delta_k$.

We could have attempted to develop a numerical method to compute the γ_k approximately. Instead, because it seemed more promising, we took a statistical approach. We randomly generated predictor data sets; in other words, we generated values of x_i. For each data set, values of α and λ were assigned, and γ_1 and γ_2 were computed. We developed a semi-parametric model with γ_1 and γ_2 as a single response, γ, and with several predictors. The central predictor is $\mathrm{tr}(L)$, which can be computed cheaply. The other predictors are p, λ, n, and k, a categorical variable with two levels that describes whether γ_1 or γ_2 is being predicted. We fitted the model to the data, and the fitted model is used in applications to compute predicted values of the γ_k based on the values of $\mathrm{tr}(L)$, p, λ, n, and k.

4.1 The Predictor tr(L)

The model for γ was identified by a combination of empirical study, which consisted of generating pilot data sets and studying them graphically, and of theory. The central idea of the model, however, using $\mathrm{tr}(L)$ to predict γ, arose from theoretical considerations, which we now describe.

Let τ be the number of monomial terms in the parametric class being fitted locally, including the constant term. If $\lambda = 1$, $\tau = p+1$, and if $\lambda = 2$, $\tau = (p+1)(p+2)/2$. Suppose, instead of fitting a loess surface, we carried out a least squares fit of the τ polynomial terms. Then L is the hat matrix of the fit and

$$\gamma_1 = \gamma_2 = \mathrm{tr}(L) = \tau \ .$$

This equality is the limiting case for loess as $\alpha \to \infty$ since the loess fit tends to the least squares polynomial fit. Let us consider the other extreme. Suppose there are no ties among the x_i and α is sufficiently small that the only positive neighborhood weight for the fit at x_i is $w_i(x_i) = 1$. Then $\hat{g}(x_i) = y_i$, L is the identity matrix, and

$$\gamma_1 = \gamma_2 = \mathrm{tr}(L) = n \ .$$

In between these extremes, γ_i and $\mathrm{tr}(L)$ depend in a highly complex way on x_i and on the specification of the surface, that is, the choices of α, λ, and which predictors, if any, are conditionally parametric; the complexity defies characterizing the dependence.

We can, however, look at a special case where the x_i have a regular structure, and gain some insight. Suppose there is one predictor and the x_i are equally-spaced. Also, suppose $\lambda = 1$. As before, let q be αn truncated to an integer, that is, the the number of points in a neighborhood. For fixed q, as n gets large, the operator matrix becomes well approximated by a circulant matrix. (In fact, if the points are equally spaced on a circle, and we take the distance between two points to be the minimum distance around the circle, the operator matrix is exactly circulant.) Let W be an altered form of the tricube weight function:

$$W = \begin{cases} c(1 - |2u|^3)^3 & \text{for } |u| < 1/2 \\ 0 & \text{otherwise,} \end{cases}$$

where c is a constant that makes W integrate to one. Thus we have simply rescaled the domain and range of T. Let W^* be the Fourier transform of W. Then for fixed q and large n,

$$\mathrm{tr} L^k \approx \alpha^{-1} \int_{-\infty}^{\infty} W^{*k}(w) dw \ .$$

Using this result we have

$$\begin{aligned} \mathrm{tr}(L) &\approx 1.72 \ \alpha^{-1} \\ \gamma_1 &\approx 2.04 \ \alpha^{-1} \\ \gamma_2 &\approx 2.06 \ \alpha^{-1} \ . \end{aligned}$$

Now, instead of equality of these three quantities we have

$$\log \frac{\gamma_1}{\mathrm{tr}(L)} \approx 0.17$$
$$\log \frac{\gamma_2}{\mathrm{tr}(L)} \approx 0.17 \ .$$

One thing this does is to dispel the notion that we can predict the γ_i simply by taking the value of $\mathrm{tr}(L)$; an error of 17% is larger than we can tolerate. In fact, as we shall see shortly, the error would be far greater for $p > 1$. But the above log ratio will be helpful in selecting and interpreting the model for γ_i.

4.2 Model Selection

We initially generated a variety of pilot data sets, explored them with graphical methods, and on the basis of this exploration and the properties of γ described earlier, chose the form of the model. The model response is

$$r = \log \ \frac{\gamma}{\mathrm{tr}(L)} \ .$$

It made sense to study percentage deviations of γ from $\mathrm{tr}(L)$ because of the near equality of the two for large and small values of α. The central predictor, $\mathrm{tr}(L)$, was transformed by the inverse square root since it resulted in a relatively simple structure for the dependence of γ on the predictors. It was convenient to scale the transformed variable so that it varied between 0 and 1; since $\mathrm{tr}(L)$ varies between τ and n, the resulting transformation was

$$z = \frac{\frac{1}{\sqrt{\mathrm{tr}(L)}} - \frac{1}{\sqrt{n}}}{\frac{1}{\sqrt{\tau}} - \frac{1}{\sqrt{n}}}.$$

We selected three other predictors that were treated as categorical in the fitting of the selected model. The first is p, the number of predictors in the data set; for the data that we generated, p ranged from 1 to 4 since applications of local regression with more than four predictors are rare due to dimensionality problems. The second is λ, the degree of the parametric class being fitted locally, so λ is 1 or 2. The third, k, is 1 if γ_1 is being predicted and is 2 if γ_2 is being predicted. In effect, these three predictors divide the model up into $4 \times 2 \times 2 = 16$ cases. Let r_i be an observation of the response and let z_i be the corresponding observation of z. The model we selected is semi-parametric:

$$r_i = \theta_{p\lambda k} z_i^{\alpha_{p\lambda k}} (1 - z_i)^{\beta_{p\lambda k}} f(z_i) \epsilon_i \ ,$$

for $p = 1$ to 4, $\lambda = 1$ to 2, and $k = 1$ to 2, where f is a nonparametric function of z, and the ϵ_i are i.i.d. random variables. To make f identifiable, we add the boundary conditions $f(0) = f(1) = 1$. This form was chosen since plots of r against z for the pilot data sets showed patterns that looked like beta functions. This is not surprising since, from the results of Section 4.1, as $\alpha \to \infty$, we have $z \to 1$ and $r \to 0$, and as α gets small, we have $z \to 0$ and $r \to 0$.

4.3 Data for Fitting the Model

To fit the model, we generated data different from the data used in the pilot studies to identify the model. For each combination of p, λ, and k, we generated 32 data sets, selected a value of α for each, and computed values of r and z. For each data set, the sample size, n, was 300, and the x_i were generated from a normal distribution with covariance matrix equal to the identity matrix. We wanted the resulting 32 values of z to be as close to uniformly distributed as possible. Thus we invoked the result of Section 4.1 that $\mathrm{tr}(L)$ behaves like α^{-1} times a constant as α changes, and chose 30 values of α so that $\alpha^{1/2}$ was equally spaced from 1 to a value sufficiently small that z would be sufficiently close to 0, but not so small that we would encounter numerical problems; then to make z sufficiently close to 1, we added two values of α, 1.5 and 3.

Figure 7 is a graph of the values of r against the values of z for the sixteen combinations of the categorical variables. Notice the extreme regularity of the dependence of r on z for the 16 cases, and the small amount of noise about the overall pattern; this is the phenomenon that allows our method to work.

4.4 Fitting the Model

The model was fitted by the backfitting algorithm (Friedman and Stuetzle, 1981; Hastie and Tibshirani, 1990), with each iteration consisting of two steps. In one step, f was estimated by a loess fit with $\alpha = 1/4$ and $\lambda = 2$. In the second step, the α and β parameters were estimated.

The estimation, to make it simpler, was done on the log scale. That is, we fitted the model

$$\log(r_i) = \theta^{\sharp}_{p\lambda k} + \alpha_{p\lambda k} \log(z_i) + \beta_{p\lambda k} \log(1 - z_i) + f^{\sharp}(z_i) + \epsilon^{\sharp}_i,$$

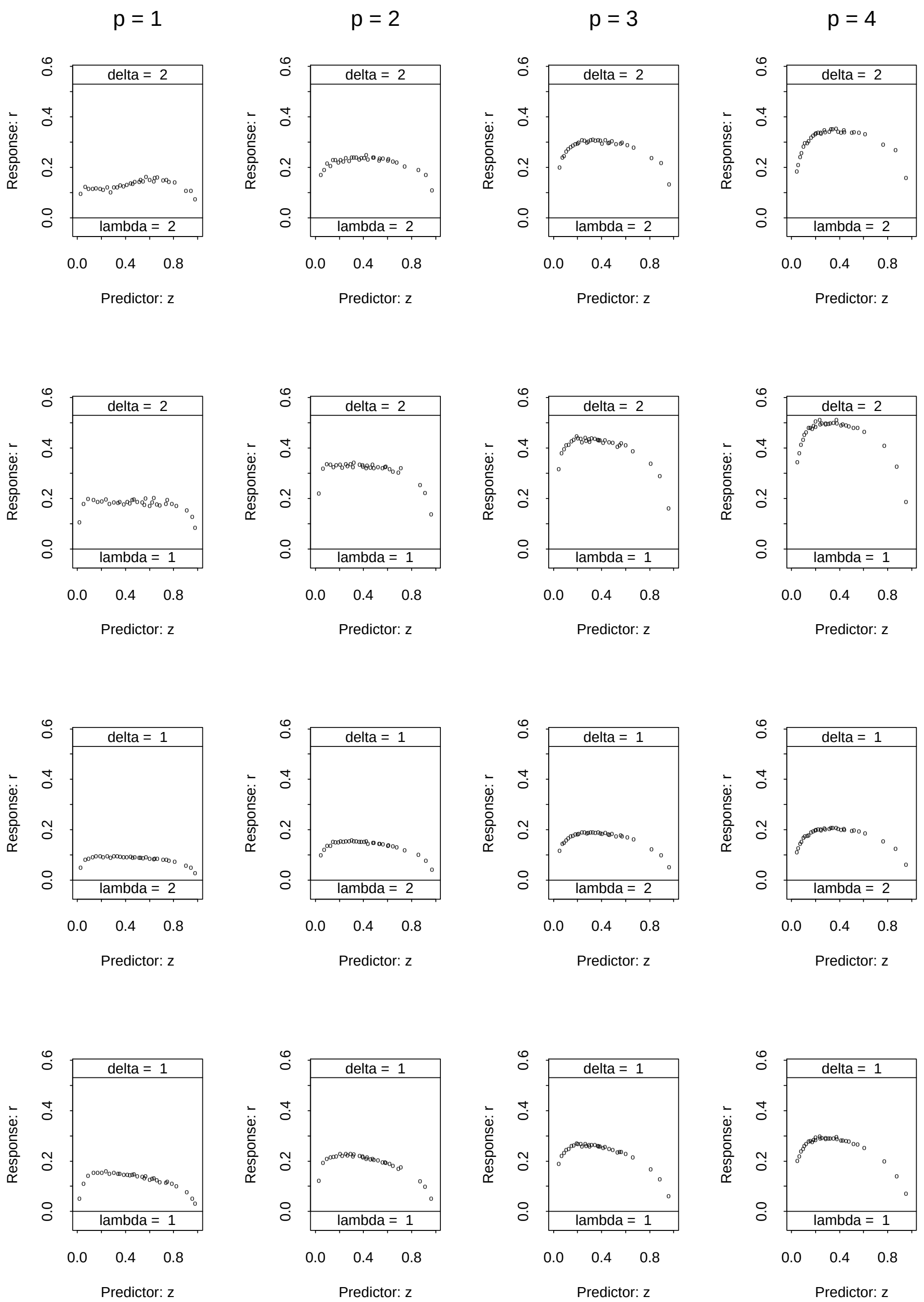

Figure 7: Data generated for fitting of semi-parametric model.

where $\theta^{\sharp}_{p\lambda k} = \log\theta_{p\lambda k}$, $f^{\sharp} = \log(f)$, and $\epsilon^{\sharp}_i = \epsilon_i$. $f^{\sharp}$ was estimated by fitting loess to

$$\log(r_i) - \hat{\theta}^{\sharp}_{p\lambda k} - \hat{\alpha}_{p\lambda k}\log(z_i) - \hat{\beta}_{p\lambda k}\log(1 - z_i)$$

as a function of z_i, where $\hat{\theta}^{\sharp}_{p\theta k}$, $\hat{\alpha}_{p\lambda k}$, and $\hat{\beta}_{p\lambda k}$ were the current estimates of the parameters, and the parameter estimates were updated by least squares regression of $\log(r_i) - f^{\sharp}$ on a constant, $\log(z_i)$, and $\log(1 - z_i)$. Thus we fitted a model which — given p, λ, and n — is a generalized additive model with a parametric part and a nonparametric part (Hastie and Tibshirani, 1990). The successive estimates of f in the backfittings were made to satisfy the boundary constraints at $z = 0$ and $z = 1$. This was done by adding the values $(0, 1)$ and $(1, 1)$ to the (z_i, r_i) values, and giving these two points prior weight 100 and the remaining points prior weight 1; the effect of this is that any neighborhood weight for these two points is multiplied by 100, forcing the fit to go through 1 at the ends. Taking the log of r, which itself is a log of $\gamma/\mathrm{tr}(L)$, is not as outlandish as it might first seem. The generated values of $\gamma/\mathrm{tr}(L)$ were all greater than 1, which is not surprising since we showed that this is so for the equally-spaced circular case in Section 4.1. (*N.B.*, we do not allow values of α that can actually yield a z of 0 or 1.)

4.5 The Fitted Model

The solid curves in Figure 8 show the fitted model, which consists of 16 univariate functions. The dashed curves show the terms

$$\hat{\theta}_{p\lambda k}(1 - z)^{\hat{\alpha}_{p\lambda k}} z^{\hat{\beta}_{p\lambda k}}.$$

Figure 9 graphs $\hat{f}(z)$. The fitted model agrees with the numerical values we derived in Section 4.1 for $\lambda = 1$ and $p = 1$. There we saw that

$$\log\left(\frac{\gamma_k}{\mathrm{tr}(L)}\right) \approx 0.17$$

for $k = 1$ and 2. Figure 8 shows the maximum values of $\log(\gamma_k|\mathrm{tr}(L))$ are in fact in the vicinity of 0.17 for $p = 1$.

The model provides an excellent fit to the data. This is illustrated in Figure 10, which graphs the sample distribution function of $\log\hat{\gamma}/\gamma$, where $\hat{\gamma}$ is the prediction of γ resulting from the the fitted model. All values lie between ± 0.03 and most lie between ± 0.01. Since the natural log transformation is used in the figure, and since $\log(x) \approx 1 + x$ for $|x| \leq 0.25$,

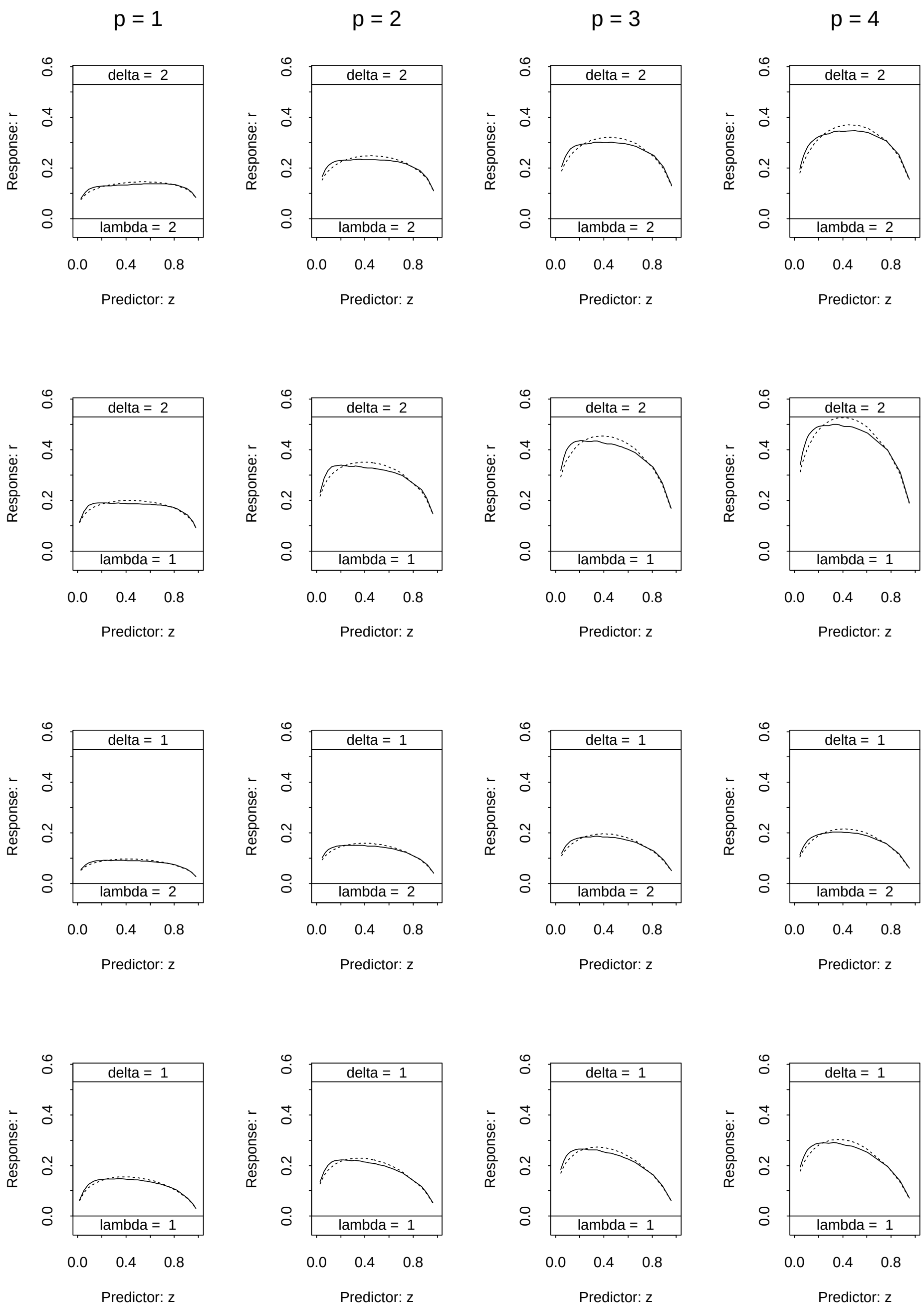

Figure 8: Fitted semi-parametric model.

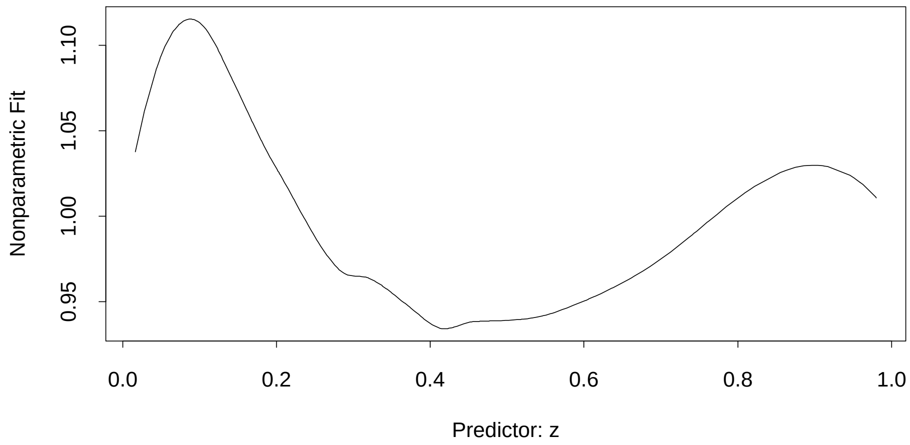

Figure 9: Loess estimate of nonparametric function in the semi-parametric model.

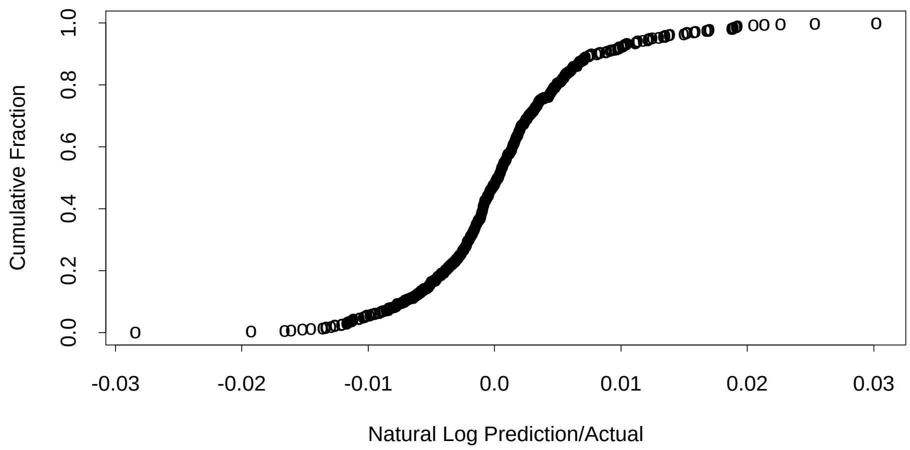

Figure 10: Deviations of predicted values of γ from the actual values.

the graph shows that the model predicts the observed values of γ to within $\pm 3\%$ in all cases and to within $\pm 1\%$ in most. This is more than enough accuracy for the purpose at hand, which is to compute degrees-of-freedom for t-intervals and F-tests.

4.6 From the Laboratory to the Field

The computer-generated data used in the pilot studies and model fitting are not representative of the entire population of data sets that users of loess fitting encounter. For one thing, the predictors were generated from normal distributions. Second, while we changed n during the pilot studies, it was equal to 300 for the data to which we fitted the model. Nor is it realistic to suppose that it would be possible to generate a representative collection. For this reason we make no attempt to use statistical inference based on the model fitting to characterize the errors of our method in practice. Instead, we simply went into the field, tried the predictive model on a large number of data sets, and compared with the actual values of γ_1 and γ_2, obtained by explicitly forming $\bar{L}$. The resulting errors, not surprisingly, were larger, as a collection, than the errors for the generated data, but were still well within the limits of acceptable accuracy. Figure 11 shows results for the NO_x data and the galaxy data studied in earlier sections. Values of $\log(\hat{\gamma}/\gamma)$ are graphed against α, where $\hat{\gamma}$ is the predicted value and γ the actual value. For the NO_x data, deviations are within $\pm 4.5\%$, and for the galaxy data, deviations are within $\pm 12.4\%$. Both are acceptable accuracies for degree-of-freedom computations.

5 Discussion

5.1 Other Work on Computational Methods for Nonparametric Regression

Much work has been reported on computational methods for nonparametric regression procedures of all types, particularly local regression procedures, smoothing splines, regression splines, kernel estimates, and projection pursuit. Since Hastie and Tibshirani (1990) have recently provided a comprehensive review, we cite here just a few examples. Projection pursuit regression (Friedman and Stuetzle, 1981; Friedman, Grosse, and Stuetzle, 1983) is a highly computer-intensive approach that required many new computational methods including a very fast updating method for local regression fitting for one predictor. Hastie and Tibshirani (1990) developed a local scoring algorithm for fitting generalized additive

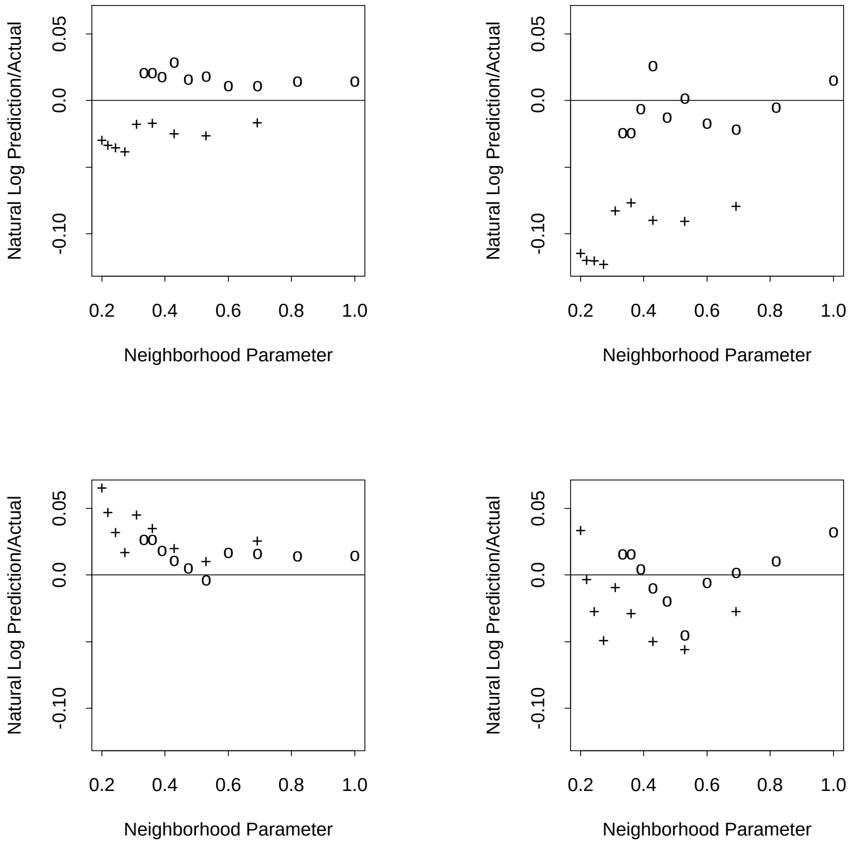

Figure 11: Deviations of predicted values of γ from the actual values. The circles graph results for the NO_x data, and the plus signs graph results for the galaxy data.

models. Bates, Lindstrom, Wahba, and Yandell (1987) and Gu and Wahba (1988) discuss methods for carrying out generalized cross-validation for smoothing spline fits. Friedman and Silverman (1989), Friedman (1991), and Breiman (to appear) have investigated adaptive fitting methods using regression splines; a number of novel computational methods have been devised to carry out knot selection in an automated way. For pointers to the large numerical approximation literature, see (Franke and Schumaker, 1987; Grosse, 1990).

5.2 Future Work

Although we have focused in this paper on k-d trees and blending to accelerate loess smoothing, it should be apparent that the same strategy could be pursued in several other directions. Other partitioning and interpolation techniques might be used, such as subsampling from the sample points and using finite element interpolation on a Delaunay triangulation. More radically, one could try k-d trees and blending with other local smoothing procedures, or abandon the local computations altogether and solve for coefficients as a single, sparse linear least squares problem. In our method, the k-d tree determined strictly by the sample locations and ignoring the values of the smooth surface. This is the method we currently use in our implementation, and is necessary if linear algebra is to be shared across multiple data sets. It is a natural choice because commonly more samples will be taken in regions where more variation is anticipated. A better tactic might be to refine based on the remaining interpolation error. When making a cut, immediately smooth at the new vertex and compare with the interpolated smooth value there based on the parent cell. If the difference is below some tolerance, no further refinement is done. In some sense this would make the process nonlinear, but because there is a nearby fixed (and linearly determined) surface, the statistical procedures remain asymptotically valid.

5.3 Fortran Routines

Fortran routines are available that implement the methods described here. They may be obtained electronically by sending the message `send dloess from a` to the address `netlib@research.att.com`. To get the single precision version, replace `dloess` by `loess` in the netlib request. If you wish to learn more about the mechanics of this distribution scheme, consult (Dongarra and Grosse, 1987).

References

Art, D., Gnanadesikan, R., and Kettenring, J. R. (1982) Data-based metrics for hierarchical cluster analysis. *Utilitas Mathematica*, **21A**, 75-99.

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

Bates, D. M., Lindstrom, M. J., Wahba, G., and Yandell, B. S. (1987) GCVPACK-routines for generalized cross-validation. *Comm. Stat.-Simula.*, **16**, 263-297.

Birkhoff, G., Cavendish, J. C., and Gordon, W. J. (1974) Multivariate approximation by locally blended univariate interpolants. *Proc. National Academy of Sciences USA*, **71**, 3423-3425.

Breiman, L. (to appear) The Π method for estimating multivariate functions from noisy data. *Technometrics.*

Brinkman, N. D. (1981) Ethanol fuel — a single-cylinder engine study of efficiency and exhaust emissions. *SAE Transactions*, **90**, No. 810345, 1410-1424.

Buta, R. (1987) The structure and dynamics of ringed galaxies, III: surface photometry and kinematics of the ringed nonbarred spiral NGC7531. *The Astrophysical J. Supplement Ser.*, **64**, 1-37.

Cavendish, J. C., (1975) Local mesh refinement using rectangular blended finite elements. *J. Comp. Physics*, **19**, 211-228.

Chan, T. F.-C. (1982) Algorithm 581: An improved algorithm for computing the singular value decomposition. *ACM Transactions on Mathematical Software*, **8**, 84-88.

Cleveland, W. S. (1979) Robust locally-weighted regression and smoothing scatterplots. *J. Am. Statist. Ass.*, **74**, 829-836.

Cleveland, W. S. and Grosse, E. (to appear) *Fitting Functions to Data.* Pacific Grove, California: Wadsworth.

Cleveland, W. S., Devlin, S. J., and Grosse, E. (1988) Regression by local fitting: methods, properties, and computational algorithms. *J. Econometrics*, **37**, 87-114.

Cleveland, W. S., Grosse, E., Shyu, W. M. (1991) Local regression models. *Statistical Models in S* edited by Chambers, J. M. and Hastie, T. Pacific Grove, California: Wadsworth.

Dongarra, J. J. and Grosse, E. (1987) Distribution of mathematical software via electronic mail. *Comm. ACM*, **30**, 403-407.

Farin, G. (1990) *Curves and Surfaces for Computer Aided Design: A Practical Guide, 2nd ed.* New York: Academic Press.

Floyd, R. W. and Rivest, R. L. (1975) Expected time bounds for selection. *Communications of the ACM*, **18** (1975), 165-172.

Franke, R. and Schumaker, L. L. (1987) A bibliography of multivariate approximation. *Topics in Multivariate Approximation* edited by Chui, C. K., Schumaker, L. L., and Utreras, F. New York: Academic Press.

Friedman, J. H. (1979) A tree-structured approach to nonparametric multiple regression. *Smoothing Techniques for Curve Estimation* edited by Gasser, T. and Rosenblatt, M. Springer.

Friedman, J. H. (1984) A variable span smoother. *Tech. Rep.*, **LCS5**. Dept. of Statistics, Stanford.

Friedman, J. H. (1991) Multivariate adaptive regression splines (with discussion) *Ann. of Statist.*, **19**, 1-141.

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.

Friedman, J. H., Grosse, E. H., and Stuetzle, W. (1983) Multidimensional additive spline approximation. *SIAM J. Sci. Stat. Comp.*, **4**, 291-301.

Friedman, J. H. and Silverman, B. W. (1989) Flexible parsimonious smoothing and additive modelling (with discussion) *Technometrics*, **31**, 3-39.

Friedman, J. H. and Stuetzle, W. (1981) Projection pursuit regression. *J. Am. Statist. Ass.*, **76**, 817-23.

Gordon, W. J. (1969) Distributive lattices and the approximation of multivariate functions. *Proceedings of the Symposium on Approximation with Special Emphasis on Splines* edited by Schoenberg, I. J. Univ. of Wisconsin, New York: Academic Press.

Grosse, E. (1990) A catalogue of algorithms for approximation. *Algorithms for Approximation II* edited by Mason, J. and Cox, M. London: Chapman and Hall.

Gu, C. and Wahba, G. (1988) Minimizing GCV/GML scores with multiple smoothing parameters via the Newton method. *Tech. Rep. 847*, Dept. of Statist., Univ. of Wisconsin.

Hastie, T. and Tibshirani, R. (1990) *Generalized Additive Models*, London: Chapman and Hall.

Lancaster, P. and Šalkauskas, K. (1986) *Curve and Surface Fitting: An Introduction.* New York: Academic Press.

Macauley, F. R. (1931) *The Smoothing of Time Series* New York: National Bureau of Economic Research.

McLain, D. H. (1974) Drawing contours from arbitrary data points. *Computer J.*, **17**, 318-324.

Schumaker, L. L. (1976) Fitting surfaces to scattered data. *Approximation Theory II* edited by Lorentz, G. G., Chui, C. K., and Schumaker, L. L. New York: Academic Press.

Stone, C. J. (1977) Consistent nonparametric regression. *Ann. of Stat.*, **5**, 595-620.

Wahba, G. (1978) Improper priors, spline smoothing, and the problem of guarding against model errors in regression. *J. Roy. Stat. Soc. B*, **40**, 364-372.

Watson, G. S. (1964) Smooth regression analysis. *Sankhya* A, **26**, 359-372.