Chapter 5

Algorithmic details

In this chapter we describe in more detail the various constituents of the principal curve and surface algorithms.

5.1. Estimation of curves and surfaces.

We described a simple smooth or local averaging procedure in chapter 4. There it was convenient to describe the smoother as a method of averaging in p space, although it has been pointed out that we can do the smoothing co-ordinate wise. That simplifies the treatment here, since we only need to discuss smoothers in their more usual regression context.

Usually a scatterplot smoother is regarded as an estimate of the conditional expectation $\mathbf{E}(Y \mid X)$, where Y and X are random variables. For our purposes X may be one or two dimensional. We will discuss one dimensional smoothers first, since they are easier to implement than two dimensional smoothers.

5.1.1. One dimensional smoothers.

The following subset of smoothers evolved naturally as estimates of conditional expectation, and are listed in order of complexity and computational cost.

5.1.1.1 Moving average smoothers.

The simplest and most natural estimate of $E(Y \mid X)$ is the *moving average* smoother. Given a sample (y_i, x_i), $i = 1, \ldots, n$, with the x_i in ascending order, we define

$$\mathbf{Smooth}_s(y \mid x_i) = \frac{1}{2k+1} \sum_{x_j \in [x_{i-k}, x_{i+k}]} y_j \tag{5.1}$$

where $k = [(ns-1)/2]$ and $s \in (0,1]$ is called the span of the smoother. An estimate of the conditional expectation at x_i is the average of the y_j for all those observations with x value equal to x_i. Since we usually only have one such observation, we average the y_j for all those observations with x value close to x_i. In the definition above, close is defined in the ordinal scale or in ranks. We can also use the interval scale or simply distance, but this is computationally more expensive. This moving average smoother suffers from a number of drawbacks. It does not produce very smooth fits and does not even reproduce straight lines unless the x_i are equispaced. It also suffers from bias effects on the boundaries.

5.1.1.2 Local linear smoothers.

An improvement on the moving average smoother is the *local linear* smoother of Friedman and Stuetzle (1981). Here the smoother estimates the conditional expectation at x_i by the fitted value from the least squares line fit of y on x using only those points for which $x_j \in (x_{i-k}, x_{i+k})$. This suffers less from boundary bias than the moving average and always reproduces straight lines exactly. The cost of computation for both of the above smoothers is $O(n)$ operations. Of course we can think of fitting local polynomials as well, but in practice the gain in bias is small relative to the extra computational burden.

5.1.1.3 Locally weighted linear smoothers.

Cleveland (1979) suggested using the local linear smoother, but also suggested weighting the points in the neighborhood according to their distance in x from x_i. This produces even smoother curves at the expense of an increased computation time of $O(kn)$ operations. (In the local linear smoother, we can obtain the fitted value at x_{i+1} from that at x_i by applying some simple updating algorithm to the latter. If local weighting is performed, we can no longer use updating formulae.)

5.1.1.4 Kernel smoothers.

The kernel smoother (Gasser and Muller, 1979) applies a weight function to every observation in calculating the fit at x_i. A variety of weight functions or kernels exist and a popular choice is the gaussian kernel centered at x_i. They produce the smoothest functions and are computationally the most expensive. The cost is $O(n^2)$ operations, although in practice the kernels have a bounded domain and this brings the cost down to $O(sn)$ for some s that depends on the kernel and the data.

In all but the kernel smoother, the span controls the smoothness of the estimated function. The larger the span, the smoother the function. In the case of the kernel smoother, there is a scale parameter that controls the spread of the kernel, and the larger the spread, the smoother the function. We will discuss the choice of spans in section 5.4.

For our particular application, it was found that the locally weighted linear smoother and the kernel smoother produced the most satisfactory results. However, when the sample size gets large, these smoothers become too expensive, and we have to sacrifice smoothness for computational speed. In this case we would use the faster local linear smoother.

5.1.2. Two dimensional smoothers.

There are substantial differences between one and two dimensional smoothers. When we find neighbors in two space, we immediately force some metric on the space in the way we define distance. In our algorithm we simply use the euclidean distance and assume the two variables are in the same scale.

It is also computationally harder to find neighbors in two dimensions than in one. The *k-d tree* (Friedman, Bently and Finkel, 1976) is an efficient algorithm and data structure for finding neighbors in k dimensions. The name arises from the data structure used to speed up the search time — a binary tree. The technique can be thought of as a multivariable version of the binary search routine. Friedman et al show that the computation required to build the tree is $O(kn \log n)$ and the expected search time for the m nearest neighbors of any point is $O(\log n)$.

5.1.3. The local planar surface smoother.

We wish to find **Smooth**$(y \mid \mathbf{x}_0)$ where $\mathbf{x}_0$ is a 2-vector not necessarily present in the sample. The following algorithm is analogous to the local linear smoother:

- Build the 2-d tree for the n pairs $(x_{11}, x_{21}), \cdots, (x_{1n}, x_{2n})$.
- Find the ns nearest neighbors of $\mathbf{x}_0$, and fit the least squares plane through their associated y values.
- The smooth at $\mathbf{x}_0$ is defined to be the fitted value at $\mathbf{x}_0$.

This algorithm does not allow updating as in the one-dimensional local linear smoother. The computation time for one fitted value is $O(\log n + ns)$. For this reason, we can include weights at no extra order in computation cost. We use gaussian weights with covariance $h^2 I$ and centered at $\mathbf{x}_0$, and h is another parameter of the procedure.

A simpler version of this smoother uses the (gaussian weighted) average of the y values for the ns neighbors. In the one dimensional case, we find that fitting local straight lines reduces the bias at the boundaries. In surface smoothing, the proportion of points on the boundary increases dramatically as we go from one to two dimensions. This provides a strong motivation for fitting planes instead of simple averages.

5.2. The projection step.

The other step in the principal curve and surface procedures is to *project* each point onto the current surface or curve. In our notation we require $\hat{\lambda}^{(j)}(x_i)$ for each i. We have already described the exact approach in chapter 3 for principal curves, which we repeat here for completeness.

5.2.1. Projecting by exact enumeration.

We project x_i *into* the line segment joining every adjacent pair of fitted values of the curve, and find the closest such projection. Into implies that when projecting we do not go beyond the two points in question. This procedure is exact but computationally expensive ($O(n)$ operations per search.) Nonetheless, we have used this method on the smaller data sets (≤ 150 observations.) There is no analogue for the principal surface routine.

5.2.2. Projections using the k-d tree.

At each of the n values of $\hat{\lambda}$ we have a fitted p vector. This is true for either the principal curve or surface procedure. We can build a p-d tree, and for each x_i, find its nearest neighbor amongst these fitted values. We then proceed differently for curves and surfaces.

- For curves we project the point *into* the segments joining this nearest point and its left neighbor. We do the same for the right neighbor and pick the closest projection.

- For surfaces we find the nearest fitted value as above. Suppose this is at $\hat{f}^{(j)}(\hat{\lambda}_k^{(j-1)})$. We then project x_i onto the plane corresponding to this fitted value and get a new value λ^*. (This plane has already been calculated in the smoothing step and is stored.) We then evaluate $\hat{f}^{(j)}(\lambda^*)$ and check if it is indeed closer. (This precautionary step is similar to projecting x_i *into* the line segments in the case of curves.) If it is, we set $\hat{\lambda}_i^{(j)} = \lambda^*$, else we set $\hat{\lambda}_i^{(j)} = \hat{\lambda}_k^{(j-1)}$. One could think of iterating this procedure, which is similar to a gradient search. Alternatively one could perform a Newton-Raphson search using derivative information contained in the least squares planes. These approaches are expensive, and in the many examples tested, made little or no difference to the estimate.

5.2.3. Rescaling the λ's to arc-length.

In the principal curve procedure, as a matter of practice, we always rescale the λ's to arc-length. The estimated λ's are then measured in the same units as the observations. Let $\hat{\lambda}_i^r$ denotes the rescaled $\hat{\lambda}_i^{(j)}$'s, and suppose $\hat{\lambda}_i^{(j)}$ are sorted. We define $\hat{\lambda}_i^r$ recursively as follows:

- $\hat{\lambda}_1^r = 0$.
- $\hat{\lambda}_i^r = \hat{\lambda}_{i-1}^r + \left\| \hat{f}^{(j)}(\hat{\lambda}_i^{(j)}) - \hat{f}^{(j)}(\hat{\lambda}_{i-1}^{(j)}) \right\|$.

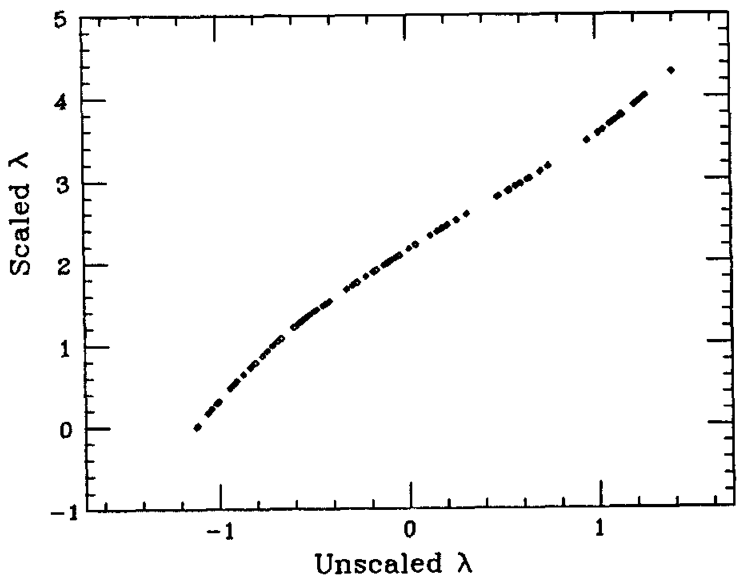

Figure (5.1) A λ plot for the circle example. Along the vertical axis we plot the final values for $\hat{\lambda}_i$, after rescaling the $\hat{\lambda}$'s at every iteration in the principal curve procedure. Along the horizontal axis we have the final $\hat{\lambda}$'s using the principal curve procedure with no rescaling.

In general there is no analogue of rescaling to arc-length for surfaces. Surface area is the corresponding quantity. We can adjust the parameters locally so that the area of a small region in parameter space has the same area as the region it defines on the surface. But this adjustment will be different in other regions of the surface having the same values for one of the parameters. The exceptions are surfaces with zero gaussian curvature. (These are surfaces that can be obtained by ***smoothly denting*** a hyperplane to form something like a corrugated sheet. One can imagine that such a rescaling is then possible).

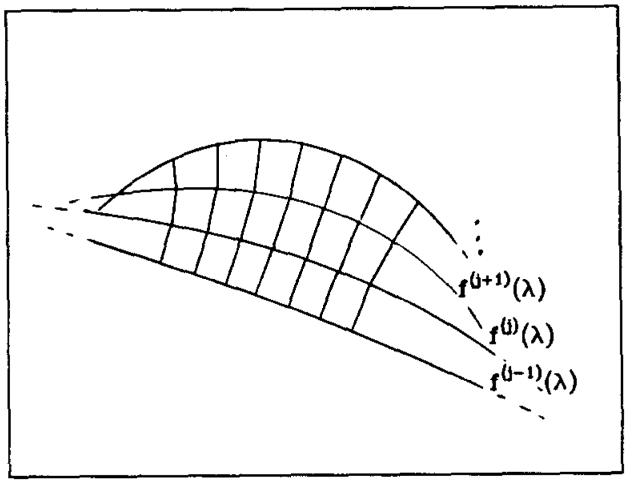

Figure (5.2) Each iteration approximately preserves the metric from the previous one. The starting curve is unit speed, and so the final curve is approximately so, up to a constant.

Even though it is not possible to do such a rescaling for surfaces, it would be comforting to know that our parametrization remains reasonably consistent over the surface as we go through the iterations.

Figure 5.1 demonstrates what happens if we use the principal curve procedure on the circle example, and do not rescale the parameter estimates at each iteration. The metric gets preserved, up to a scalar. Figure 5.2shows why this is so. The original metric gets transferred from one iteration to the next. As long as the curves do not change dramatically from one iteration to the next, there will not be much distortion.

5.3. Span selection.

We consider there to be two categories of spans corresponding to two distinct stages in the algorithm.

5.3.1. Global procedural spans.

The first guess for f is a straight line. In many of the interesting situations, the final curve will not be a function of the arc length of this initial curve. The final curve is reached by successively *bending* the original curve. We have found that if the initial spans of the smoother are too small, the curve will bend too fast, and may get lost! The most successful strategy has been to initially use large spans, and then to decrease them slowly. In particular, we start with a span of $0.5n$, and let the procedure converge. We then drop the span to $0.4n$ and converge again. Finally the same is done at $0.3n$ by which time the procedure has found the general shape of the curve. We then switch to mean square error (MSE) span selection mode.

5.3.2. Mean squared error spans.

The procedure has converged to a self consistent curve for the span last used. If we reduce the span, the average distance will decrease. This situation arises in regression as well. In regression, however, there is a remedy. We can use cross-validation (Stone 1977) to select the span. We briefly outline the idea.

5.3.2.1 Cross-validation in regression.

Suppose we have a sample of n independent pairs (y_i, x_i) from the model $Y = f(X) + \epsilon$. A nonparametric estimate of $f(x_0)$ is $\hat{f}_s(x_0) = \textbf{Smooth}_s(y \mid x_0)$. The expected squared prediction error is

$$EPE = \mathbf{E}(Y - \hat{f}(X))^2 \tag{5.2}$$

where the expectation is taken over everything random (i.e. the sample used to estimate $\hat{f}(\cdot)$ and the future pairs (X,Y)). We use the residual sum of squares,

$$RSS(s) = \sum_{i=1}^{n} (y_i - \hat{f}_s(x_i))^2,$$

as the natural estimate of EPE. This is however, a biassed estimate, as can be seen by letting the span s shrink down to 0. The smooth then estimates y_i by itself, and RSS is zero. We call this ***bias due to overfitting*** since the bias is due to the influence y_i has in forming its own prediction. This also shows us that we cannot use RSS to help us pick the span. We can, however, use the cross-validated residual sum of squares (CVRSS). This is defined as

$$CVRSS(s) = \sum_{i=1}^{n} (y_i - \textbf{Smooth}_s^{(i)}(y \mid x_i))^2, \tag{5.3}$$

where $\textbf{Smooth}_s^{(i)}(y \mid x_i)$ is the smooth calculated from the data with the pair (y_i, x_i) removed, and then evaluated at x_i. It can be shown that this estimate is approximately unbiassed for the true prediction error. In minimizing the prediction error, we also minimize the integrated mean square error EMSE given by

$$EMSE(s) = \mathbf{E}(\hat{f}_s(X) - f(X))^2$$

since they differ by a constant. We can decompose this expression into a sum of a variance and bias terms, namely

$$\begin{aligned} EMSE(s) &= \mathbf{E}[\,\mathbf{Var}(\hat{f}_s(X)] + \mathbf{E}[(\,\mathbf{E}(\hat{f}_s(X)\,|X) - f(X))^2] \\ &= VAR(s) + BIAS^2(s). \end{aligned}$$

As s gets smaller the variance gets larger (averaging over less points) but the bias gets smaller (width of the neighborhoods gets smaller), and vice versa. Thus if we pick s to minimize CVRSS(s) we are trying to minimize the true prediction error or equivalently to find the span which optimally mixes bias and variance.

Getting back to the curves, one thought is to cross-validate the orthogonal distance function. This, however, will not work because we would still tend to use span zero. (In general we have more chance of being close to the interpolating curve than any other curve). Instead, we cross-validate the co-ordinates separately.

5.3.2.2 Cross-validation for principal curves.

Suppose f is a principal curve of h, for which we have an estimate $\hat{f}$ based on a sample $x_1, \ldots, x_n$.

A natural requirement is to choose s to minimize $EMSE(s)$ given by

$$\begin{aligned} EMSE(s) &= \mathbf{E}_h \left\| f(\lambda_f(X)) - \hat{f}_s(\lambda_f(X)) \right\|^2 \\ &= \sum_{j=1}^{p} \mathbf{E}_{h_\lambda}(\,\mathbf{Var}(\hat{f}_s(\lambda_f(X))\,|\,\lambda_f(x)) + \mathbf{E}_{h_\lambda} \left\| f(\lambda_f(X)) - \hat{f}_s(\lambda_f(X)) \right\|^2 \end{aligned} \tag{5.4}$$

which is once again a trade-off between bias and variance. Notice that were we to look at the closest distance between these curves, then the interpolating curve would be favored. As in the regression case, the quantity $EPE(s) = \mathbf{E}_h \left\| X - \hat{f}_s(\lambda_f(X)) \right\|^2$ estimates $EMSE(s) + D(f)$, where $D(f) = \mathbf{E} \left\| X - f(\lambda_f(X)) \right\|^2$. It is thus equivalent to choose s to minimize $EMSE(s)$ or $EPE(s)$. As in the regression case, the cross-validated estimate

$$CVRSS(s) = \sum_{j=1}^{p} \left[\sum_{i=1}^{n} (x_{ji} - \,\mathbf{Smooth}_s^{(i)}(x_j\,|\,\lambda_i))^2 \right], \tag{5.5}$$

where $\lambda_i = \lambda_f(x_i)$, attempts to do this. Since we do not know λ_i, we pick $\lambda_i = \lambda_{\hat{f}^{(k)}}(x_i)$ where $\hat{f}^{(k)}$ is the (non cross-validated) estimate of f. In practice, we evaluate $CVRSS(s)$ for a few values of s and pick the one that gives the minimum.

From the computing angle, if the smoother is linear one can easily find the cross-validated fits. In this case $\hat{y} = Cy$ for some smoother matrix C, and the cross-validated fit $\hat{y}_{(i)}$ is given by $\hat{y}_{(i)} = \sum_{j \neq i} \frac{c_{ij} y_j}{1 - c_{ii}}$ (Wahba 1975).

There are a number of issues connected with the algorithms that have not yet been mentioned, such as a robustness and outlier detection, what to display and how to do it, and bootstrap techniques. The next chapter consists of many examples, and we will deal with these issues as they arise.