7

Splines

7.1 Smoothing by Penalizing Curve Flexibility

Let's go back to the problem of smoothing one-dimensional data. We have data points $(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$, and we want to find a good approximation $\widehat{\mu}$ to the true conditional expectation or regression function μ. Previously, we controlled how smooth we made $\widehat{\mu}$ indirectly, through the bandwidth of our kernels. But why not be more direct, and control smoothness itself?

A natural way to do this is to minimize the **spline objective function**

$$\mathcal{L}(m, \lambda) \equiv \frac{1}{n}\sum_{i=1}^{n}(y_i - m(x_i))^2 + \lambda \int (m''(x))^2 dx \tag{7.1}$$

The first term here is just the mean squared error of using the curve $m(x)$ to predict y. We know and like this; it is an old friend.

The *second* term, however, is something new for us. m'' is the second derivative of m with respect to x — it would be zero if m were linear, so this measures the **curvature** of m at x. The sign of $m''(x)$ says whether the curvature at x is concave or convex, but we don't care about that so we square it. We then integrate this over all x to say how curved m is, on average. Finally, we multiply by λ and add that to the MSE. This is adding a **penalty** to the MSE criterion — given two functions with the same MSE, we prefer the one with less average curvature. We will accept changes in m that increase the MSE by 1 unit if they also reduce the average curvature by at least λ.

The curve or function which solves this minimization problem,

$$\widehat{\mu}_\lambda = \operatorname*{argmin}_m \mathcal{L}(m, \lambda) \tag{7.2}$$

is called a **smoothing spline**, or **spline curve**. The name "spline" comes from a simple tool used by craftsmen to draw smooth curves, which was a thin strip of a flexible material like a soft wood; you pin it in place at particular points, called **knots**, and let it bend between them. (When the gas company dug up my front yard and my neighbor's driveway, the contractors who put everything back used a plywood board to give a smooth, curved edge to the new driveway. That board was a spline, and the knots were pairs of metal stakes on either side of the board. Figure 7.1 shows the spline after concrete was poured on one side of it.) Bending the spline takes energy — the stiffer the material, the more energy has to go into bending it through the same shape, and so the material makes a straighter curve

Figure 7.1 A wooden spline used to create a smooth, curved border for a paved area (Shadyside, Pittsburgh, October 2014).

between given points. For smoothing splines, using a stiffer material corresponds to increasing λ.

It is possible to show (§7.6 below) that all solutions to Eq. 7.1, no matter what the data might be, are piecewise cubic polynomials which are continuous and have continuous first and second derivatives — i.e., not only is $\widehat{\mu}$ continuous, so are $\widehat{\mu}'$ and $\widehat{\mu}''$. The boundaries between the pieces sit at the original data points. By analogy with the craftman's spline, the boundary points are called the **knots** of the smoothing spline. The function is continuous beyond the largest and smallest data points, but it is always linear in those regions.[1]

I will also assert, without proof, that, with enough pieces, such piecewise cubic polynomials can approximate any well-behaved function arbitrarily closely. Finally, smoothing splines are linear smoothers, in the sense of Chapter 1: predicted values are linear combinations of the training-set response values y_i — see Eq. 7.21 below.

7.1.1 The Meaning of the Splines

Look back to the optimization problem. As $\lambda \to \infty$, any curvature at all becomes infinitely costly, and only linear functions are allowed. But we know how to minimize mean squared error with linear functions, that's OLS. So we understand that limit.

On the other hand, as $\lambda \to 0$, we decide that we don't care about curvature. In that case, we can always come up with a function which just interpolates between the data points, an **interpolation spline** passing exactly through each point. More specifically, of the infinitely many functions which interpolate between those points, we pick the one with the minimum average curvature.

At intermediate values of λ, $\widehat{\mu}_\lambda$ becomes a function which compromises between

[1] Can you explain why it is linear outside the data range, in terms of the optimization problem?

having low curvature, and bending to approach all the data points closely (on average). The larger we make λ, the more curvature is penalized. There is a bias-variance trade-off here. As λ grows, the spline becomes less sensitive to the data, with lower variance to its predictions but more bias. As λ shrinks, so does bias, but variance grows. For consistency, we want to let $\lambda \to 0$ as $n \to \infty$, just as, with kernel smoothing, we let the bandwidth $h \to 0$ while $n \to \infty$.

We can also think of the smoothing spline as the function which minimizes the mean squared error, subject to a constraint on the average curvature. This turns on a general corresponds between penalized optimization and optimization under constraints, which is explored in Appendix H.3. The short version is that each level of λ corresponds to imposing a cap on how much curvature the function is allowed to have, on average, and the spline we fit with that λ is the MSE-minimizing curve subject to that constraint.[2] As we get more data, we have more information about the true regression function and can relax the constraint (let λ shrink) without losing reliable estimation.

It will not surprise you to learn that we select λ by cross-validation. Ordinary k-fold CV is entirely possible, but leave-one-out CV works quite well for splines. In fact, the default in most spline software is either leave-one-out CV, or the even faster approximation called "generalized cross-validation" or GCV (see §3.4.3).

7.2 Computational Example: Splines for Stock Returns

The default R function for fitting a smoothing spline is `smooth.spline`:

```
smooth.spline(x, y, cv = FALSE)
```

where `x` should be a vector of values for input variable, `y` is a vector of values for the response (in the same order), and the switch `cv` controls whether to pick λ by generalized cross-validation (the default) or by leave-one-out cross-validation. The object which `smooth.spline` returns has an `$x` component, *re-arranged in increasing order*, a `$y` component of fitted values, a `$yin` component of original values, etc. See `help(smooth.spline)` for more.

As a concrete illustration, Figure 7.2 looks at the daily logarithmic returns[3] of the S&P 500 stock index, on 5542 consecutive trading days, from 9 February 1993 to 9 February 2015[4].

[2] The slightly longer version: Consider minimizing the MSE (not the penalized MSE), but only over functions m where $\int (m''(x))^2 dx$ is at most some maximum level C. λ would then be the Lagrange multiplier enforcing the constraint. The constrained but unpenalized optimization is equivalent to the penalized but unconstrained one. In economics, λ would be called the "shadow price" of average curvature in units of MSE, the rate at which we'd be willing to pay to have the constraint level C marginally increased.

[3] For a financial asset whose price on day t is p_t and which pays a dividend on that day of d_t, the log-returns on t are $\log (p_t + d_t)/p_{t-1}$. Financiers and other professional gamblers care more about the log returns than about the price change, $p_t - p_{t-1}$, because the log returns give the rate of profit (or loss) on investment. We are using a price series which is adjusted to incorporate dividend (and related) payments.

[4] This uses the handy `pdfetch` library, which downloads data from such public domain sources as the Federal Reserve, Yahoo Finance, etc.

```
require(pdfetch)

## Loading required package: pdfetch

sp <- pdfetch_YAHOO("SPY", fields = "adjclose", from = as.Date("1993-02-09"),
    to = as.Date("2015-02-09"))
sp <- diff(log(sp))
sp <- sp[-1]
```

We want to use the log-returns on one day to predict what they will be on the next. The horizontal axis in the figure shows the log-returns for each of 2527 days t, and the vertical axis shows the corresponding log-return for the succeeding day $t+1$. A linear model fitted to this data displays a slope of -0.0642 (grey line in the figure). Fitting a smoothing spline with cross-validation selects $\lambda = 0.0127$, and the black curve:

```
sp.today <- head(sp, -1)
sp.tomorrow <- tail(sp, -1)
coefficients(lm(sp.tomorrow ~ sp.today))
##   (Intercept)      sp.today
##  0.0003716837 -0.0640901257
sp.spline <- smooth.spline(x = sp.today, y = sp.tomorrow, cv = TRUE)
sp.spline
## Call:
## smooth.spline(x = sp.today, y = sp.tomorrow, cv = TRUE)
##
## Smoothing Parameter  spar= 1.346847  lambda= 0.01299752 (11 iterations)
## Equivalent Degrees of Freedom (Df): 5.855613
## Penalized Criterion (RSS): 0.7825304
## PRESS(l.o.o. CV): 0.0001428132
sp.spline$lambda
## [1] 0.01299752
```

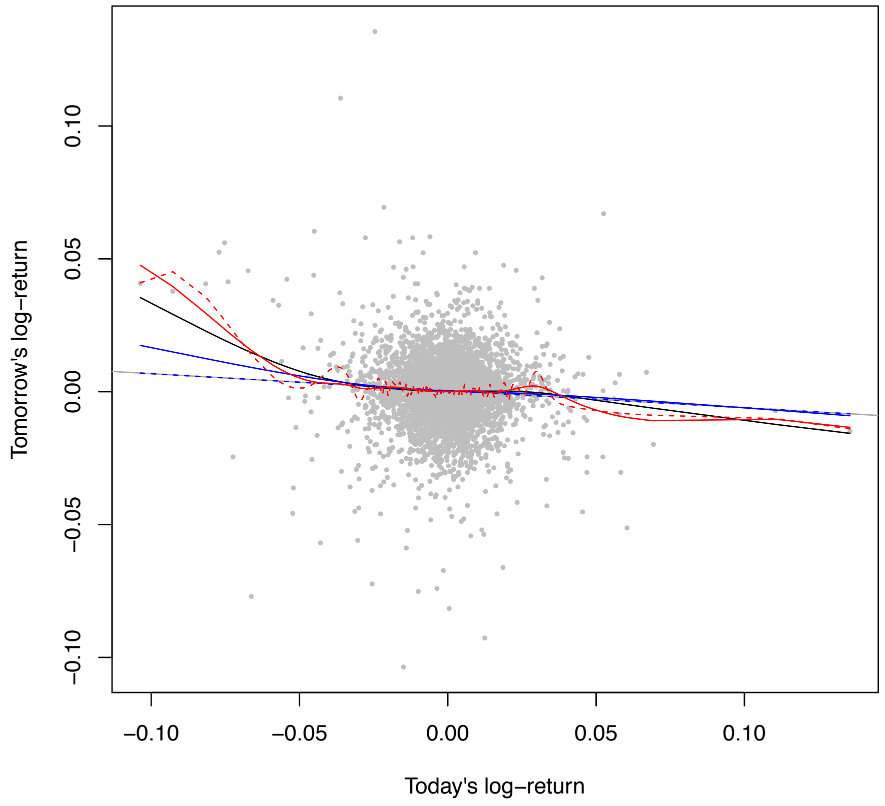
(PRESS is the "prediction sum of squares", i.e., the sum of the squared leave-one-out prediction errors.) This is the curve shown in black in the figure. The blue curves are for large values of λ, and clearly approach the linear regression; the red curves are for smaller values of λ.

The spline can also be used for prediction. For instance, if we want to know what the return to expect following a day when the log return was $+0.01$, we do

```
predict(sp.spline, x = 0.01)
## $x
## [1] 0.01
##
## $y
## [1] 0.0001948564
```

R Syntax Note:

The syntax for `predict` with `smooth.spline` spline differs slightly from the syntax for predict with `lm` or `np`. The latter two want a `newdata` argument, which should be a data-frame with column names matching those in the formula used to fit the model. The `predict` function for `smooth.spline`, though, just wants a vector called `x`. Also, while `predict` for `lm` or `np` returns a vector of predictions, `predict`



```
plot(as.vector(sp.today), as.vector(sp.tomorrow), xlab = "Today's log-return",
    ylab = "Tomorrow's log-return", pch = 16, cex = 0.5, col = "grey")
abline(lm(sp.tomorrow ~ sp.today), col = "darkgrey")
sp.spline <- smooth.spline(x = sp.today, y = sp.tomorrow, cv = TRUE)
lines(sp.spline)
lines(smooth.spline(sp.today, sp.tomorrow, spar = 1.5), col = "blue")
lines(smooth.spline(sp.today, sp.tomorrow, spar = 2), col = "blue", lty = 2)
lines(smooth.spline(sp.today, sp.tomorrow, spar = 1.1), col = "red")
lines(smooth.spline(sp.today, sp.tomorrow, spar = 0.5), col = "red", lty = 2)
```

Figure 7.2 The S& P 500 log-returns data (grey dots), with the OLS linear regression (dark grey line), the spline selected by cross-validation (solid black, $\lambda = 0.0127$), some more smoothed splines (blue, $\lambda = 0.178$ and 727) and some less smooth splines (red, $\lambda = 2.88 \times 10^{-4}$ and 1.06×10^{-8}). Incoveniently, `smooth.spline` does not let us control λ directly, but rather a somewhat complicated but basically exponential transformation of it called `spar`. (See `help(smooth.spline)` for the gory details.) The equivalent λ can be extracted from the return value, e.g., `smooth.spline(sp.today,sp.tomorrow,spar=2)$lambda`.

for `smooth.spline` returns a list with an `x` component (in increasing order) and a `y` component, which is the sort of thing that can be put directly into `points` or `lines` for plotting.

7.2.1 Confidence Bands for Splines

Continuing the example, the smoothing spline selected by cross-validation has a negative slope everywhere, like the regression line, but it's asymmetric — the slope is more negative to the left, and then levels off towards the regression line. (See Figure 7.2 again.) Is this real, or might the asymmetry be a sampling artifact?

We'll investigate by finding confidence bands for the spline, much as we did for kernel regression in Chapter 6 and Problem Set A.27, problem 5. Again, we need to bootstrap, and we can do it either by resampling the residuals or resampling whole data points. Let's take the latter approach, which assumes less about the data. We'll need a simulator:

```
sp.frame <- data.frame(today = sp.today, tomorrow = sp.tomorrow)
sp.resampler <- function() {
    n <- nrow(sp.frame)
    resample.rows <- sample(1:n, size = n, replace = TRUE)
    return(sp.frame[resample.rows, ])
}
```

This treats the points in the scatterplot as a complete population, and then draws a sample from them, with replacement, just as large as the original[5]. We'll also need an estimator. What we want to do is get a whole bunch of spline curves, one on each simulated data set. But since the values of the input variable will change from one simulation to another, to make everything comparable we'll evaluate each spline function on a fixed grid of points, that runs along the range of the data.

```
grid.300 <- seq(from = min(sp.today), to = max(sp.today), length.out = 300)
sp.spline.estimator <- function(data, eval.grid = grid.300) {
    fit <- smooth.spline(x = data[, 1], y = data[, 2], cv = TRUE)
    return(predict(fit, x = eval.grid)$y)
}
```

This sets the number of evaluation points to 300, which is large enough to give visually smooth curves, but not so large as to be computationally unwieldly.

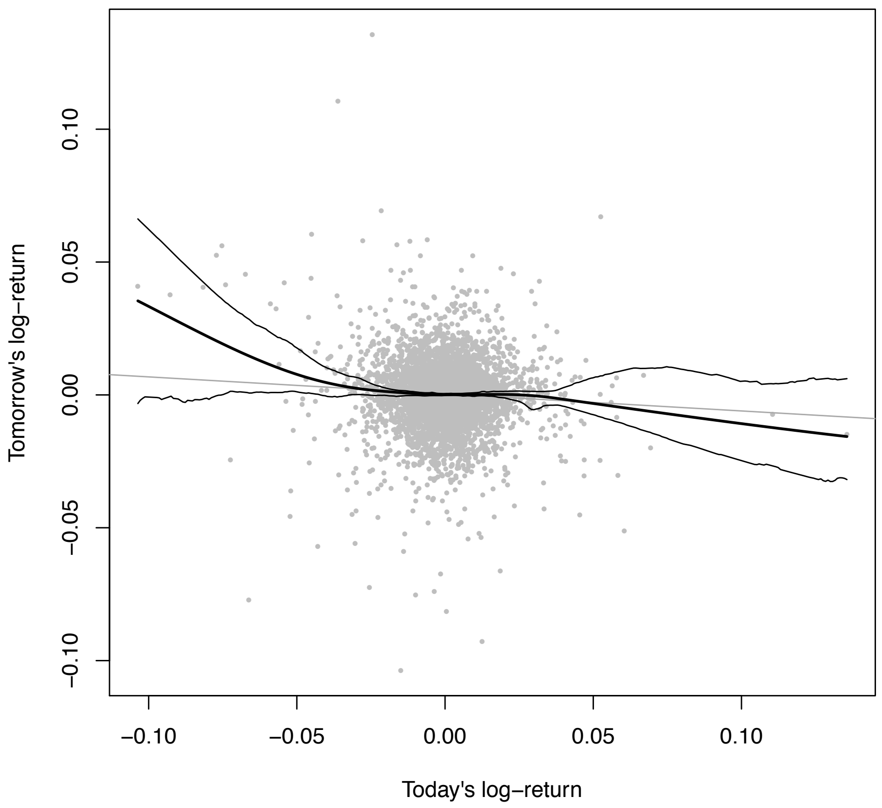
Now put these together to get confidence bands:

```
sp.spline.cis <- function(B, alpha, eval.grid = grid.300) {
    spline.main <- sp.spline.estimator(sp.frame, eval.grid = eval.grid)
    spline.boots <- replicate(B, sp.spline.estimator(sp.resampler(), eval.grid = eval.grid))
    cis.lower <- 2 * spline.main - apply(spline.boots, 1, quantile, probs = 1 -
        alpha/2)
    cis.upper <- 2 * spline.main - apply(spline.boots, 1, quantile, probs = alpha/2)
    return(list(main.curve = spline.main, lower.ci = cis.lower, upper.ci = cis.upper,
        x = eval.grid))
}
```

[5] §25.5 covers more refined ideas about bootstrapping time series.

The return value here is a list which includes the original fitted curve, the lower and upper confidence limits, and the points at which all the functions were evaluated.

Figure 7.3 shows the resulting 95% confidence limits, based on `B=1000` bootstrap replications. (Doing all the bootstrapping took 45 seconds on my laptop.) These are pretty clearly asymmetric in the same way as the curve fit to the whole data, but notice how wide they are, and how they get wider the further we go from the center of the distribution in either direction.



```
sp.cis <- sp.spline.cis(B = 1000, alpha = 0.05)
plot(as.vector(sp.today), as.vector(sp.tomorrow), xlab = "Today's log-return",
    ylab = "Tomorrow's log-return", pch = 16, cex = 0.5, col = "grey")
abline(lm(sp.tomorrow ~ sp.today), col = "darkgrey")
lines(x = sp.cis$x, y = sp.cis$main.curve, lwd = 2)
lines(x = sp.cis$x, y = sp.cis$lower.ci)
lines(x = sp.cis$x, y = sp.cis$upper.ci)
```

Figure 7.3 Bootstrapped pointwise confidence band for the smoothing spline of the S & P 500 data, as in Figure 7.2. The 95% confidence limits around the main spline estimate are based on 1000 bootstrap re-samplings of the data points in the scatterplot.

7.3 Basis Functions and Degrees of Freedom

7.3.1 Basis Functions

Splines, I said, are piecewise cubic polynomials. To see how to fit them, let's think about how to fit a global cubic polynomial. We would define four **basis functions**[6],

$$B_1(x) = 1 \tag{7.3}$$

$$B_2(x) = x \tag{7.4}$$

$$B_3(x) = x^2 \tag{7.5}$$

$$B_4(x) = x^3 \tag{7.6}$$

and chose to only consider regression functions that are linear combinations of the basis functions,

$$\mu(x) = \sum_{j=1}^{4} \beta_j B_j(x) \tag{7.7}$$

Such regression functions would be linear in the *transformed* variables $B_1(x), \ldots B_4(x)$, even though it is nonlinear in x.

To estimate the coefficients of the cubic polynomial, we would apply each basis function to each data point x_i and gather the results in an $n \times 4$ matrix $\mathbf{B}$,

$$B_{ij} = B_j(x_i) \tag{7.8}$$

Then we would do OLS using the $\mathbf{B}$ matrix in place of the usual data matrix $\mathbf{x}$:

$$\hat{\beta} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{y} \tag{7.9}$$

Since splines are piecewise cubics, things proceed similarly, but we need to be a little more careful in defining the basis functions. Recall that we have n values of the input variable x, $x_1, x_2, \ldots x_n$. For the rest of this section, I will assume that these are in increasing order, because it simplifies the notation. These n "knots" define $n+1$ pieces or segments: $n-1$ of them between the knots, one from $-\infty$ to x_1, and one from x_n to $+\infty$. A third-order polynomial on each segment would seem to need a constant, linear, quadratic and cubic term per segment. So the segment running from x_i to x_{i+1} would need the basis functions

$$\mathbf{1}_{(x_i,x_{i+1})}(x),\ (x-x_i)\mathbf{1}_{(x_i,x_{i+1})}(x),\ (x-x_i)^2\mathbf{1}_{(x_i,x_{i+1})}(x),\ (x-x_i)^3\mathbf{1}_{(x_i,x_{i+1})}(x) \tag{7.10}$$

where as usual the indicator function $\mathbf{1}_{(x_i,x_{i+1})}(x)$ is 1 if $x \in (x_i, x_{i+1})$ and 0 otherwise. This makes it seem like we need $4(n+1) = 4n+4$ basis functions.

However, we know from linear algebra that the number of basis vectors we need is equal to the number of dimensions of the vector space. The number of adjustable coefficients for an arbitrary piecewise cubic with $n+1$ segments is indeed $4n+4$, but splines are constrained to be smooth. The spline must be continuous, which means that at each x_i, the value of the cubic from the left,

[6] See App. B.11 for brief reminders about basis functions.

defined on (x_{i-1}, x_i), must match the value of the cubic from the right, defined on (x_i, x_{i+1}). This gives us one constraint per data point, reducing the number of adjustable coefficients to at most $3n+4$. Since the first and second derivatives are also continuous, we are down to just $n+4$ coefficients. Finally, we know that the spline function is linear outside the range of the data, i.e., on $(-\infty, x_1)$ and on (x_n, ∞), lowering the number of coefficients to n. There are no more constraints, so we end up needing only n basis functions. And in fact, from linear algebra, any set of n piecewise cubic functions which are linearly independent[7] can be used as a basis. One common choice is

$$B_1(x) = 1 \tag{7.11}$$

$$B_2(x) = x \tag{7.12}$$

$$B_{i+2}(x) = \frac{(x-x_i)^3_+ - (x-x_n)^3_+}{x_n - x_i} - \frac{(x-x_{n-1})^3_+ - (x-x_n)^3_+}{x_n - x_{n-1}} \tag{7.13}$$

where $(a)_+ = a$ if $a > 0$, and $= 0$ otherwise. This rather unintuitive-looking basis has the nice property that the second and third derivatives of each B_j are zero outside the interval (x_1, x_n).

Now that we have our basis functions, we can once again write the spline as a weighted sum of them,

$$m(x) = \sum_{j=1}^{m} \beta_j B_j(x) \tag{7.14}$$

and put together the matrix $\mathbf{B}$ where $B_{ij} = B_j(x_i)$. We can write the spline objective function in terms of the basis functions,

$$n\mathcal{L} = (\mathbf{y} - \mathbf{B}\beta)^T(\mathbf{y} - \mathbf{B}\beta) + n\lambda\beta^T\Omega\beta \tag{7.15}$$

where the matrix Ω encodes information about the curvature of the basis functions:

$$\Omega_{jk} = \int B_j''(x)B_k''(x)dx \tag{7.16}$$

Notice that only the quadratic and cubic basis functions will make non-zero contributions to Ω. With the choice of basis above, the second derivatives are non-zero on, at most, the interval (x_1, x_n), so each of the integrals in Ω is going to be finite. This is something we (or, realistically, R) can calculate *once*, no matter what λ is. Now we can find the smoothing spline by differentiating with respect to β:

$$0 = -2\mathbf{B}^T\mathbf{y} + 2\mathbf{B}^T\mathbf{B}\hat{\beta} + 2n\lambda\Omega\hat{\beta} \tag{7.17}$$

$$\mathbf{B}^T\mathbf{y} = \left(\mathbf{B}^T\mathbf{B} + n\lambda\Omega\right)\hat{\beta} \tag{7.18}$$

$$\hat{\beta} = \left(\mathbf{B}^T\mathbf{B} + n\lambda\Omega\right)^{-1}\mathbf{B}^T\mathbf{y} \tag{7.19}$$

[7] Recall that vectors $\vec{v}_1, \vec{v}_2, \ldots \vec{v}_d$ are linearly independent when there is no way to write any one of the vectors as a weighted sum of the others. The same definition applies to functions.

Notice, incidentally, that we can now show splines are linear smoothers:

$$\widehat{\mu}(x) = \mathbf{B}\hat{\beta} \tag{7.20}$$

$$= \mathbf{B}\left(\mathbf{B}^T\mathbf{B} + n\lambda\Omega\right)^{-1}\mathbf{B}^T\mathbf{y} \tag{7.21}$$

Once again, if this were ordinary linear regression, the OLS estimate of the coefficients would be $(\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$. In comparison to that, we've made two changes. First, we've substituted the basis function matrix $\mathbf{B}$ for the original matrix of independent variables, $\mathbf{x}$ — a change we'd have made already for a polynomial regression. Second, the "denominator" is not $\mathbf{x}^T\mathbf{x}$, or even $\mathbf{B}^T\mathbf{B}$, but $\mathbf{B}^T\mathbf{B}+n\lambda\Omega$. Since $\mathbf{x}^T\mathbf{x}$ is n times the covariance matrix of the independent variables, we are taking the covariance matrix of the spline basis functions and adding some extra covariance — how much depends on the shapes of the functions (through Ω) and how much smoothing we want to do (through λ). The larger we make λ, the less the actual data matters to the fit.

In addition to explaining how splines can be fit quickly (do some matrix arithmetic), this illustrates two important tricks. One, which we won't explore further here, is to turn a nonlinear regression problem into one which is linear in another set of basis functions. This is like using not just one transformation of the input variables, but a whole library of them, and letting the data decide which transformations are important. There remains the issue of selecting the basis functions, which can be quite tricky. In addition to the spline basis[8], most choices are various sorts of waves — sine and cosine waves of different frequencies, various wave-forms of limited spatial extent ("wavelets"), etc. The ideal is to chose a function basis where only a few non-zero coefficients would need to be estimated, but this requires some understanding of the data...

The other trick is that of stabilizing an unstable estimation problem by adding a penalty term. This reduces variance at the cost of introducing some bias. Exercise 7.2 explores this idea.

Effective degrees of freedom

In §1.5.3.2, we defined the number of effective degrees of freedom for a linear smoother with smoothing matrix $\mathbf{w}$ as just $\operatorname{tr}\mathbf{w}$. Thus, Eq. 7.21 lets us calculate the effective degrees of freedom of a spline, as $\operatorname{tr}\left(\mathbf{B}(\mathbf{B}^T\mathbf{B} + n\lambda\Omega)^{-1}\mathbf{B}^T\right)$. You should be able to convince yourself from this that increasing λ will, all else being equal, reduce the effective degrees of freedom of the fit.

7.4 Splines in Multiple Dimensions

Suppose we have *two* input variables, x and z, and a single response y. How could we do a spline fit?

[8] Or, really, bases; there are multiple sets of basis functions for the splines, just like there are multiple sets of basis vectors for the plane. Phrases like "B splines" and "P splines" refer to particular choices of spline basis functions.

One approach is to generalize the spline optimization problem so that we penalize the curvature of the spline surface (no longer a curve). The appropriate penalized least-squares objective function to minimize is

$$\mathcal{L}(m,\lambda) = \sum_{i=1}^{n}(y_i - m(x_i,z_i))^2 + \lambda\int\left[\left(\frac{\partial^2 m}{dx^2}\right)^2 + 2\left(\frac{\partial^2 m}{dxdz}\right)^2 + \left(\frac{\partial^2 m}{dz^2}\right)^2\right]dxdz \tag{7.22}$$

The solution is called a **thin-plate spline**. This is appropriate when the two input variables x and z should be treated more or less symmetrically[9].

An alternative is use the spline basis functions from section 7.3. We write

$$m(x) = \sum_{j=1}^{M_1}\sum_{k=1}^{M_2}\beta_{jk}B_j(x)B_k(z) \tag{7.23}$$

Doing all possible multiplications of one set of numbers or functions with another is said to give their **outer product** or **tensor product**, so this is known as a **tensor product spline** or **tensor spline**. We have to chose the number of terms to include for each variable (M_1 and M_2), since using n for each would give n^2 basis functions, and fitting n^2 coefficients to n data points is asking for trouble.

7.5 Smoothing Splines versus Kernel Regression

For one input variable and one output variable, smoothing splines can basically do everything which kernel regression can do[10]. The advantages of splines are their computational speed and (once we've calculated the basis functions) simplicity, as well as the clarity of controlling curvature directly. Kernels however are easier to program (if slower to run), easier to analyze mathematically[11], and extend more straightforwardly to multiple variables, and to combinations of discrete and continuous variables.

7.6 Some of the Math Behind Splines

Above, I claimed that a solution to the optimization problem Eq. 7.1 exists, and is a continuous, piecewise-cubic polynomial, with continuous first and second derivatives, with pieces at the x_i, and linear outside the range of the x_i. I do not know of any truly elementary way of showing this, but I will sketch here how it's established, if you're interested.

Eq. 7.1 asks us to find the function which minimize the sum of the MSE and

[9] Generalizations to more than two input variables are conceptually straightforward — just keep adding up more partial derivatives — but the book-keeping gets annoying.

[10] In fact, there is a technical sense in which, for large n, splines act like a kernel regression with a specific non-Gaussian kernel, and a bandwidth which varies over the data, being smaller in high-density regions. See Simonoff (1996, §5.6.2), or, for more details, Silverman (1984).

[11] Most of the bias-variance analysis for kernel regression can be done with basic calculus, as we did in Chapter **??**. The corresponding analysis for splines requires working in infinite-dimensional function spaces called "Hilbert spaces". It's a pretty theory, if you like that sort of thing.

a certain integral. Even the MSE can be brought inside the integral, using Dirac delta functions:

$$\mathcal{L} = \int \left[\lambda (m''(x))^2 + \frac{1}{n} \sum_{i=1}^{n} (y_i - m(x_i))^2 \delta(x - x_i) \right] dx \tag{7.24}$$

In what follows, without loss of generality, assume that the x_i are ordered, so $x_1 \leq x_2 \leq \ldots x_i \leq x_{i+1} \leq \ldots x_n$. With some loss of generality but a great gain in simplicity, assume none of the x_i are equal, so we can make those inequalities strict.

The subject which deals with maximizing or minimizing integrals of functions is the calculus of variations[12], and one of its basic tricks is to write the integrand as a function of x, the function, and its derivatives:

$$\mathcal{L} = \int L(x, m, m', m'') dx \tag{7.25}$$

where, in our case,

$$L = \lambda (m''(x))^2 + \frac{1}{n} \sum_{i=1}^{n} (y_i - m(x_i))^2 \delta(x - x_i) \tag{7.26}$$

This sets us up to use a general theorem of the calculus of variations, to the effect that any function $\hat{m}$ which minimizes L must also solve L's **Euler-Lagrange equation**:

$$\left. \frac{\partial L}{\partial m} - \frac{d}{dx}\frac{\partial L}{\partial m'} + \frac{d^2}{dx^2}\frac{\partial L}{\partial m''} \right|_{m=\hat{m}} = 0 \tag{7.27}$$

In our case, the Euler-Lagrange equation reads

$$-\frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{m}(x_i)) \delta(x - x_i) + 2\lambda \frac{d^2}{dx^2} \hat{m}''(x) = 0 \tag{7.28}$$

Remembering that $\hat{m}''(x) = d^2\hat{m}/dx^2$,

$$\frac{d^4}{dx^4} \hat{m}(x) = \frac{1}{n\lambda} \sum_{i=1}^{n} (y_i - \hat{m}(x_i)) \delta(x - x_i) \tag{7.29}$$

The right-hand side is zero at any point x other than one of the x_i, so the fourth derivative has to be zero in between the x_i. This in turn means that the function must be piecewise cubic. Now fix an x_i, and pick any two points which bracket it, but are both greater than x_{i-1} and less than x_{i+1}; call them l and u. Integrate

[12] In addition to its uses in statistics, the calculus of variations also shows up in physics ("what is the path of least action?"), control theory ("what is the cheapest route to the objective?") and stochastic processes ("what is the most probable trajectory?"). Gershenfeld (1999, ch. 4) is a good starting point.

our Euler-Lagrange equation from l to u:

$$\int_l^u \frac{d^4}{dx^4}\hat{m}(x)dx = \int_l^u \frac{1}{n\lambda}\sum_{i=1}^{n}(y_i - \hat{m}(x_i))\delta(x - x_i) \tag{7.30}$$

$$\hat{m}'''(u) - \hat{m}'''(l) = \frac{y_i - \hat{m}(x_i)}{n\lambda} \tag{7.31}$$

That is, the third derivative makes a jump when we move across x_i, though (since the fourth derivative is zero), it doesn't matter which pair of points above and below x_i we compare third derivatives at. Integrating the equation again,

$$\hat{m}''(u) - \hat{m}''(l) = (u - l)\frac{y_i - \hat{m}(x_i)}{n\lambda} \tag{7.32}$$

Letting u and l approach x_i from either side, so $u - l \to 0$, we see that $\hat{m}''$ makes no jump at x_i. Repeating this trick twice more, we conclude the same about $\hat{m}'$ and $\hat{m}$ itself. In other words, $\hat{m}$ must be continuous, with continuous first and second derivatives, and a third derivative that is constant on each (x_i, x_{i+1}) interval. Since the fourth derivative is zero on those intervals (and undefined at the x_i), the function must be a piecewise cubic, with the piece boundaries at the x_i, and continuity (up to the second derivative) across pieces.

To see that the optimal function must be linear below x_1 and above x_n, suppose that it wasn't. Clearly, though, we could reduce the curvature as much as we want in those regions, without altering the value of the function at the boundary, or even its first derivative there. This would yield a better function, i.e., one with a lower value of $\mathcal{L}$, since the MSE would be unchanged and the average curvature would be smaller. Taking this to the limit, then, the function must be linear outside the observed data range.

We have now shown[13] that the optimal function $\hat{m}$, *if it exists*, must have all the properties I claimed for it. We have not shown either that there is a solution, or that a solution is unique if it does exist. However, we can use the fact that solutions, *if* there are any, are piecewise cubics obeying continuity conditions to set up a system of equations to find their coefficients. In fact, we did so already in §7.3.1, where we saw it's a system of n independent linear equations in n unknowns. Such a thing does indeed have a unique solution, here Eq. 7.19.

7.7 Further Reading

There are good discussions of splines in Simonoff (1996, ch. 5), Hastie *et al.* (2009, ch. 5) and Wasserman (2006, §5.5). Wood (2006, ch. 4) includes a thorough practical treatment of splines as a preparation for additive models (see Chapter 8 below) and generalized additive models (see Chapters **??**–12). The classic reference, by one of the inventors of splines as a useful statistical tool, is Wahba (1990); it's great if you already know what a Hilbert space is and how to navigate one.

[13] For a very weak value of "shown", admittedly.

Historical notes

The first introduction of spline smoothing in the statistical literature seems to be Whittaker (1922). (His "graduation" is more or less our "smoothing".) He begins with an "inverse probability" (we would now say "Bayesian") argument for minimizing Eq. 7.1 to find the most probable curve, based on the *a priori* hypothesis of smooth Gaussian curves observed through Gaussian error, and gives tricks for fitting splines more easily with the mathematical technology available in 1922.

The general optimization problem, and the use of the word "spline", seems to have its roots in numerical analysis in the early 1960s; those spline functions were intended as ways of smoothly interpolating between given points. The connection to statistical smoothing was made by Schoenberg (1964) (who knew about Whittaker's earlier work) and by Reinsch (1967) (who gave code). Splines were then developed as a practical tool in statistics and in applied mathematics in the 1960s and 1970s. Silverman (1985) is a still-readable and insightful summary of this work.

In econometrics, spline smoothing a time series is called the "Hodrick-Prescott filter", after two economists who re-discovered the technique in 1981, along with a fallacious argument that λ should always take a particular value (1600, as it happens), regardless of the data. See Paige and Trindade (2010) for a (polite) discussion, and demonstration of the advantages of cross-validation.

Exercises

7.1 The `smooth.spline` function lets you set the effective degrees of freedom explicitly. Write a function which chooses the number of degrees of freedom by five-fold cross-validation.

7.2 When we can't measure our predictor variables perfectly, it seems like a good idea to try to include multiple measurements for each one of them. For instance, if we were trying to predict grades in college from grades in high school, we might include the student's grade from each year separately, rather than simply averaging them. Multiple measurements of the same variable will however tend to be strongly correlated, so this means that a linear regression will be *nearly* multi-collinear. This in turn means that it will tend to have multiple, mutually-canceling large coefficients. This makes it hard to interpret the regression and hard to treat the predictions seriously. (See §2.1.1.)

One strategy for coping with this situation is to carefully select the variables one uses in the regression. Another, however, is to add a penalty for large coefficient values. For historical reasons, this second strategy is called **ridge regression**, or **Tikhonov regularization**. Specifically, while the OLS estimate is

$$\widehat{\beta}_{OLS} = \operatorname*{argmin}_{\beta} \frac{1}{n}\sum_{i=1}^{n}{\left(y_i - x_i \cdot \beta\right)^2} \ , \tag{7.33}$$

the regularized or penalized estimate is

$$\widehat{\beta}_{RR} = \operatorname*{argmin}_{\beta} \left[\frac{1}{n}\sum_{i=1}^{n}{\left(y_i - x_i \cdot \beta\right)^2}\right] + \lambda\sum_{j=1}^{p}{\beta_j^2} \tag{7.34}$$

1. Show that the matrix form of the ridge-regression objective function is
$$n^{-1}(\mathbf{y} - \mathbf{x}\beta)^T(\mathbf{y} - \mathbf{x}\beta) + \lambda\beta^T\beta \tag{7.35}$$
2. Show that the optimum is
$$\widehat{\beta}_{RR} = (\mathbf{x}^T\mathbf{x} + n\lambda\mathbf{I})^{-1}\mathbf{x}^T\mathbf{y} \tag{7.36}$$
(This is where the name "ridge regression" comes from: we take $\mathbf{x}^T\mathbf{x}$ and add a "ridge" along the diagonal of the matrix.)
3. What happens as $\lambda \to 0$? As $\lambda \to \infty$? (For the latter, it may help to think about the case of a one-dimensional X first.)
4. Let $Y = Z+\epsilon$, with $Z \sim \mathcal{U}(-1, 1)$ and $\epsilon \sim \mathcal{N}(0, 0.05)$. Generate 2000 draws from Z and Y. Now let $X_i = 0.9Z + \eta$, with $\eta \sim \mathcal{N}(0, 0.05)$, for $i \in 1 : 50$. Generate corresponding X_i values. Using the first 1000 rows of the data only, do ridge regression of Y on the X_i (not on Z), plotting the 50 coefficients as functions of λ. Explain why ridge regression is called a **shrinkage estimator**.
5. Use cross-validation with the first 1000 rows to pick the optimal value of λ. Compare the out-of-sample performance you get with this penalty to the out-of-sample performance of OLS.

For more on ridge regression, see Appendix H.3.5.