Chapter 7

Splines

7.1 Smoothing by Directly Penalizing Curve Flexibility

Let's go back to the problem of smoothing one-dimensional data. We imagine, that is to say, that we have data points $(x_1,y_1),(x_2,y_2),\ldots(x_n,y_n)$, and we want to find a function $\hat{r}(x)$ which is a good approximation to the true conditional expectation or regression function $r(x)$. Previously, we rather indirectly controlled how irregular we allowed our estimated regression curve to be, by controlling the bandwidth of our kernels. But why not be more direct, and directly control irregularity?

A natural way to do this, in one dimension, is to minimize the **spline objective function**

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

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'' says whether the curvature 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. In fact, we are willing to accept changes in m that increase the MSE by 1 unit if they also reduce the average curvature by at least λ.

The *solution* to this minimization problem,

$$\hat{r}_\lambda = \underset{m}{\operatorname{argmin}}\, \mathscr{L}(m,\lambda) \tag{7.2}$$

is a function of x, or curve, called a **smoothing spline**, or **smoothing spline func-**

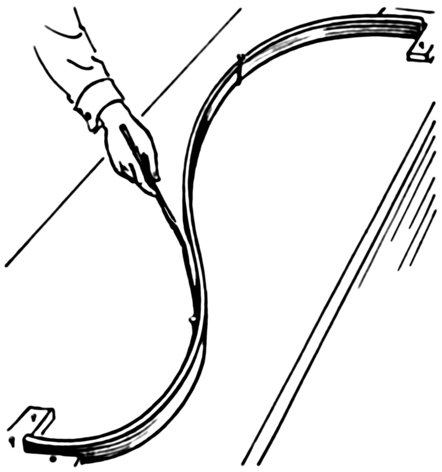


Figure 7.1: A craftsman's spline, from Wikipedia, s.v. "Flat spline".

tion[1].

It is possible to show that all solutions, no matter what the initial data are, are piecewise cubic polynomials which are continuous and have continuous first and second derivatives — i.e., not only is $\hat{r}$ continuous, so are $\hat{r}'$ and $\hat{r}''$. The boundaries between the pieces are located at the original data points. These are called, somewhat obscure, the **knots** of the spline. The function continuous beyond the largest and smallest data points, but it is always linear on those regions.[2] I will not attempt to prove this.

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

[1]The name "spline" actually 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, as in Figure 7.1. (A few years ago, when the gas company dug up my front yard, the contractors they hired to put the driveway back used a plywood board to give a smooth, outward-curve edge to the new driveway. The "knots" were metal stakes which the board was placed between, the curve of the board was a spline, and they poured concrete to one side of the board, which they left standing until the concrete dried.) Bending such a material takes energy — the stiffer the material, the more energy has to go into bending it through the same shape, and so the straighter the curve it will make between given points. For smoothing splines, using a stiffer material corresponds to increasing λ.

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

7.1.1 The Meaning of the Splines

Look back to the optimization problem. As $\lambda \to \infty$, having any curvature at all becomes infinitely penalized, 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, $\hat{r}_\lambda$ becomes a function which compromises between 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 D. 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. 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 an even faster approximation called "generalized cross-validation" or GCV. The details of how to rapidly compute the LOOCV or GCV scores are not especially important for us, but can be found, if you want them, in many books, such as Simonoff (1996, §5.6.3).

7.2 An Example

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

```
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 some data on the stock market, which we will revisit later in the context of time series. The vector `sp` contains the log-returns[3] of the S & P 500 stock index on 2528 consecutive trading days:

```
sp <- read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/12/SPhistory.short.csv")
# We only want closing prices
sp <- sp[,7]
# The data are in reverse chronological order, which is weird for us
sp <- rev(sp)
# And in fact we only want log returns, i.e., difference in logged prices
sp <- diff(log(sp))
```

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 -0.0822 (grey line in the figure). Fitting a smoothing spline with cross-validation selects $\lambda = 0.0513$, and the black curve:

```
> sp.today <- sp[-length(sp)]
> sp.tomorrow <- sp[-1]
> sp.spline <- smooth.spline(x=sp.today,y=sp.tomorrow,cv=TRUE)
Warning message:
In smooth.spline(sp[-length(sp)], sp[-1], cv = TRUE) :
  crossvalidation with non-unique 'x' values seems doubtful
> sp.spline
Call:
smooth.spline(x = sp[-length(sp)], y = sp[-1], cv = TRUE)

Smoothing Parameter  spar= 1.389486  lambda= 0.05129822 (14 iterations)
Equivalent Degrees of Freedom (Df): 4.206137
Penalized Criterion: 0.4885528
```

[3]For a financial asset whose price on day t is p_t, the log-returns on t are $\log p_t/p_{t-1}$. Financiers and other professional gamblers care more about the log returns than about the price change, $p_t - p_{t-1}$.

```
PRESS: 0.0001949005
> sp.spline$lambda
[1] 0.05129822
```

(PRESS is the "prediction sum of squares", i.e., the sum of the squared leave-one-out prediction errors. Also, the warning about cross-validation, while well-intentioned, is caused here by there being just two days with log-returns of zero.) This is the curve shown in black in the figure. The curves shown in blue are for large values of λ, and clearly approach the linear regression; the curves shown in red 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 $+0.01$,

```
> predict(sp.spline,x=0.01)
$x
[1] 0.01
$y
[1] -0.0007169499
```

i.e., a very slightly negative log-return. (Giving both an `x` and a `y` value like this means that we can directly feed this output into many graphics routines, like `points` or `lines`.)

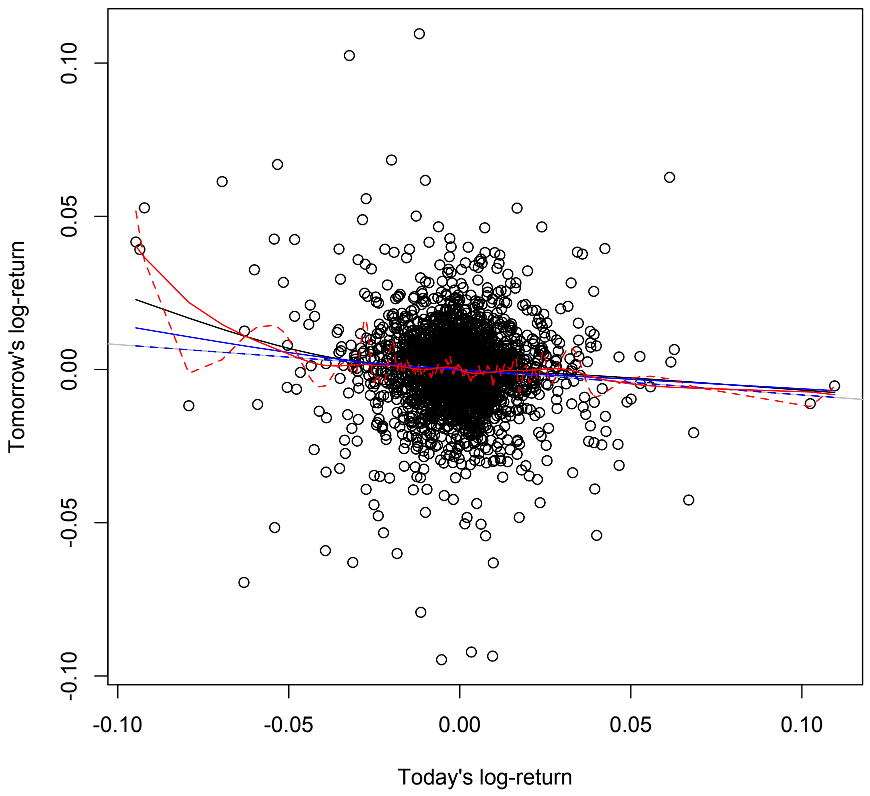
7.2.1 Confidence Bands for Splines

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 in Chapter 5 for kernel regression. 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. 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.



```
plot(sp.today,sp.tomorrow,xlab="Today's log-return",
     ylab="Tomorrow's log-return")
abline(lm(sp.tomorrow ~ sp.today),col="grey")
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 (circles), together with the OLS linear regression (grey line), the spline selected by cross-validation (solid black curve, $\lambda = 0.0513$), some more smoothed splines (blue, $\lambda = 0.322$ and 1320) and some less smooth splines (red, $\lambda = 4.15 \times 10^{-4}$ and 1.92×10^{-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`.

```
# Set up a grid of evenly-spaced points on which to evaluate the spline
grid.300 <- seq(from=min(sp.today),to=max(sp.today),length.out=300)


sp.spline.estimator <- function(data,eval.grid=grid.300) {
  # Fit spline to data, with cross-validation to pick lambda
  fit <- smooth.spline(x=data[,1],y=data[,2],cv=TRUE)
  # Do the prediction on the grid and return the predicted values
  return(predict(fit,x=eval.grid)$y)  # We only want the predicted values
}
```

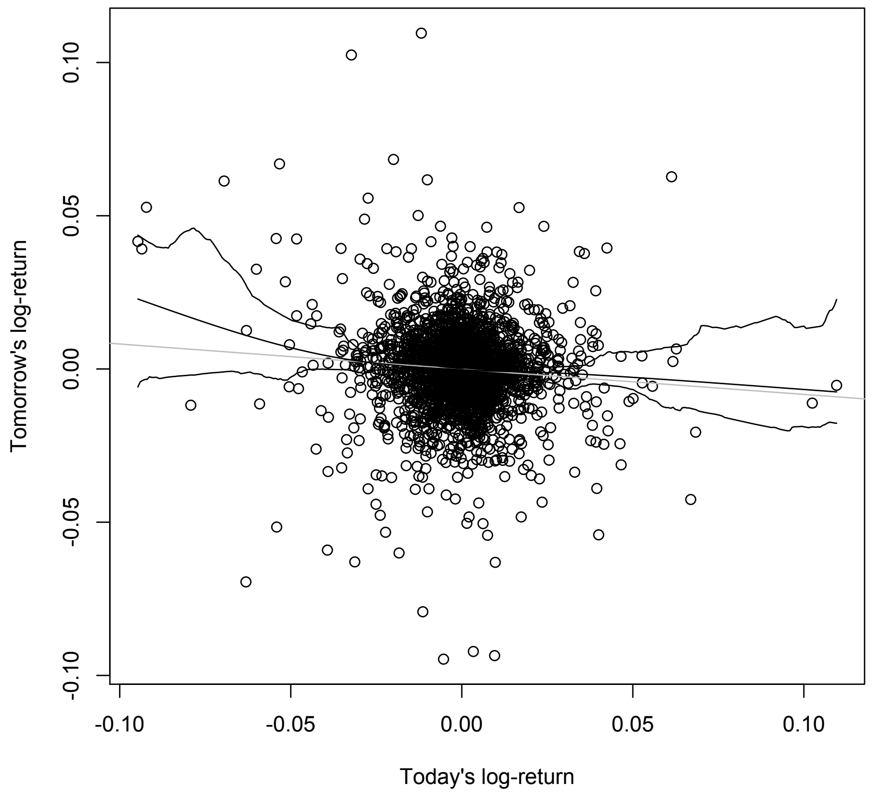
This sets the number of evaluation points to 300, which is large enough to give visually smooth curves, but not so large as to be computationally unwieldly.

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)
  # Draw B boottrap samples, fit the spline to each
    # Result has length(eval.grid) rows and B columns
  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/
  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=seq(from=min(sp.today),to=max(sp.today),length.out=m)))
}
```

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(sp.today,sp.tomorrow,xlab="Today's log-return",
     ylab="Tomorrow's log-return")
abline(lm(sp.tomorrow ~ sp.today),col="grey")
lines(x=sp.cis$x,y=sp.cis$main.curve)
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**,

$$B_1(x) = 1 \quad (7.3)$$
$$B_2(x) = x \quad (7.4)$$
$$B_3(x) = x^2 \quad (7.5)$$
$$B_4(x) = x^3 \quad (7.6)$$

with the hypothesis being that the regression function is a weight sum of these,

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

That is, the regression would be linear in the *transformed* variable $B_1(x), \ldots B_4(x)$, 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 $\mathbf{B}$,

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

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

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

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

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

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, 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[4] can be used as a basis. One common choice is

$$\begin{aligned} B_1(x) &= 1 && (7.11)\\ B_2(x) &= x && (7.12)\\ B_{i+2}(x) &= \frac{(x-x_i)_+^3 - (x-x_n)_+^3}{x_n - x_i} - \frac{(x-x_{n-1})_+^3 - (x-x_n)_+^3}{x_n - x_{n-1}} && (7.13) \end{aligned}$$

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,

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

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

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

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:

$$\begin{aligned} 0 &= -2\mathbf{B}^T\mathbf{y} + 2\mathbf{B}^T\mathbf{B}\hat{\beta} + 2n\lambda\Omega\hat{\beta} && (7.17)\\ \mathbf{B}^T\mathbf{y} &= \left(\mathbf{B}^T\mathbf{B} + n\lambda\Omega\right)\hat{\beta} && (7.18)\\ \hat{\beta} &= \left(\mathbf{B}^T\mathbf{B} + n\lambda\Omega\right)^{-1}\mathbf{B}^T\mathbf{y} && (7.19) \end{aligned}$$

[4]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:

$$\hat{y} = \hat{m}(x) = \mathbf{B}\hat{\beta} \tag{7.20}$$

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

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 plain polynomial regression. Second, the "denominator" is not $\mathbf{x}^T\mathbf{x}$, 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[5], 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 2 explores this idea.

7.3.2 Degrees of Freedom

You may have noticed that we haven't, so far, talked about the degrees of freedom of our regression models. This is one of those concepts which is much more important for linear regression than elsewhere, but it does still have its uses, and this is a good place to explain how it's calculated for more general models.

First, though, we need to recall how it works for linear regression. We'll start with an $n \times p$ data matrix of predictor variables $\mathbf{x}$, and an $n \times 1$ column matrix of response values $\mathbf{y}$. The ordinary least squares estimate of the p-dimensional coefficient vector β is

$$\hat{\beta} = \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T\mathbf{y} \tag{7.22}$$

[5]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. If you see the phrase "B splines", it refers to a particular choice of spline basis functions.

This implies, in turn, that we can write the fitted values in terms of $\mathbf{x}$ and $\mathbf{y}$:

$$\hat{\mathbf{y}} = \mathbf{x}\hat{\beta} \tag{7.23}$$

$$= \left(\mathbf{x}\left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T\right)\mathbf{y} \tag{7.24}$$

$$= \mathbf{h}\mathbf{y} \tag{7.25}$$

where $\mathbf{h}$ is the $n \times n$ matrix, where h_{ij} says how much of each observed y_j contributes to each fitted $\hat{y}_i$. This is called the **influence matrix**, or less formally the **hat matrix**.

Notice that $\mathbf{h}$ depends *only* on the predictor variables in $\mathbf{x}$; the observed response values in $\mathbf{y}$ don't matter. If we change around $\mathbf{y}$, the fitted values $\hat{\mathbf{y}}$ will also change, but only within the limits allowed by $\mathbf{h}$. There are n independent coordinates along which $\mathbf{y}$ can change, so we say the data have n degrees of freedom. Once $\mathbf{x}$ and so $\mathbf{h}$ are fixed, however, $\hat{\mathbf{y}}$ has to lie in an $(n-p)$-dimensional hyper-plane in this n-dimensional space. There are only $n-p$ *independent* coordinates along which the fitted values can move. Hence we say that the residual degrees of freedom are $n-p$, and p degrees of freedom are captured by the linear regression.

The algebraic expression of this fact is that, for a linear regression, the trace of $\mathbf{h}$ is always p:

$$\operatorname{tr}\mathbf{h} = \operatorname{tr}\left(\mathbf{x}\left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T\right) \tag{7.26}$$

$$= \operatorname{tr}\left(\mathbf{x}^T\mathbf{x}\left(\mathbf{x}^T\mathbf{x}\right)^{-1}\right) \tag{7.27}$$

$$= \operatorname{tr}\mathbf{I}_p = p \tag{7.28}$$

since for any matrices $\mathbf{a}, \mathbf{b}$, $\operatorname{tr}(\mathbf{ab}) = \operatorname{tr}(\mathbf{ba})$, and $\mathbf{x}^T\mathbf{x}$ is a $p \times p$ matrix[6].

For the general class of linear smoothers (Chapter 1), at an arbitrary point x the predicted value of y is a weighted (linear) combination of the observed values,

$$\hat{r}(x) = \sum_{j=1}^{n} \hat{w}(x, x_j) y_j \tag{7.29}$$

In particular,

$$\hat{y}_i = \hat{r}(x_i) = \sum_{j=1}^{n} \hat{w}(x_i, x_j) y_j \tag{7.30}$$

and so we can write

$$\hat{\mathbf{y}} = \mathbf{h}\mathbf{y} \tag{7.31}$$

where now, in the general case, $h_{ij} = \hat{w}(x_i, x_j)$. We still call $\mathbf{h}$ the hat or influence matrix. For a kernel smoother, this can be directly calculated from the kernels, but for a spline we need to use Eq. 7.21.

By analogy with Eq. 7.28, we *define* the **effective degrees of freedom** of a linear smoother to be $\operatorname{tr}\mathbf{h}$. Many of the formulas you learned for linear regression, e.g., dividing the residual sum of squares by $n-p$ to get an unbiased estimate of the noise variance, continue to hold approximately for linear smoothers with the effective degrees of freedom in place of p.

[6]This assumes that $\mathbf{x}^T\mathbf{x}$ has an inverse. Can you work out what happens when it does not?

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?

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

$$\mathscr{L}(m,\lambda) = \sum_{i=1}^{n}(y_i - m(x_i,z_i))^2 + \lambda \int dx dz \left[\left(\frac{\partial^2 m}{dx^2}\right)^2 + 2\left(\frac{\partial^2 m}{dxdz}\right)^2 + \left(\frac{\partial^2 m}{dz^2}\right)^2 \right] \tag{7.32}$$

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[7].

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

$$m(x) = \sum_{j=1}^{M_1}\sum_{k=1}^{M_2} \beta_{jk} B_j(x) B_k(z) \tag{7.33}$$

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[8]. 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[9], and extend more straightforwardly to multiple variables, and to combinations of discrete and continuous variables.

7.6 Further Reading

There are good discussions of splines in Simonoff (1996, §5), Hastie *et al.* (2009, ch. 5) and Wasserman (2006, §5.5). Wood (2006, ch. 4) includes a thorough practical

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

[8]In fact, as $n \to \infty$, smoothing splines approach the kernel regression curve estimated with a specific, rather non-Gaussian kernel. See Simonoff (1996, §5.6.2).

[9]Most of the bias-variance analysis for kernel regression can be done with basic calculus, as we did in Chapter 4. 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.

treatment of splines as a preparation for additive models (see Chapter 8 in these notes) and generalized additive models (see Chapters 12–13).

The classic reference, by one of the people who really developed splines as a useful statistical tool, is Wahba (1990), which is great if you already know what a Hilbert space is and how to manipulate it.

The first introduction of spline smoothing in the statistical literature seems to be Whittaker (1922). ("Graduation" was the term often used then for what we call "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. He does not, however, use the word "spline", and I am not sure when that analogy was made.

In economics and econometrics, the use spline smoothing is known as 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, which they said was 1600, regardless of the data[10]. See Paige and Trindade (2010) for a (polite) discussion, and demonstration of the advantages of cross-validation.

7.7 Exercises

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.

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.

   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

$$\widehat{\beta}_{OLS} = \operatorname*{argmin}_{\beta} \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i\beta)^2 , \tag{7.34}$$

[10] As it were: Hodrick and Prescott re-invented the wheel, and decided that it should be an octagon.

the regularized or penalized estimate is

$$\widehat{\beta}_{RR} = \underset{\beta}{\operatorname{argmin}} \left[\frac{1}{n} \sum_{i=1}^{n} (y_i - x_i \beta)^2 \right] + \sum_{j=1}^{p} \beta_j^2 \qquad (7.35)$$

(a) Show that the matrix form of the ridge-regression objective function is

$$n^{-1}(\mathbf{y} - \mathbf{x}\beta)^T (\mathbf{y} - \mathbf{x}\beta) + \lambda \beta^T \beta \qquad (7.36)$$

(b) Show that the optimum is

$$\widehat{\beta}_{RR} = (\mathbf{x}^T \mathbf{x} + n\lambda \mathbf{I})^{-1} \mathbf{x}^T \mathbf{y} \qquad (7.37)$$

(c) What happens as $\lambda \rightarrow 0$? As $\lambda \rightarrow \infty$? (For the latter, it may help to think about the case of a one-dimensional X first.)

(d) 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**.

(e) 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.