

# 7

## Moving Beyond Linearity

So far in this book, we have mostly focused on linear models. Linear models are relatively simple to describe and implement, and have advantages over other approaches in terms of interpretation and inference. However, standard linear regression can have significant limitations in terms of predictive power. This is because the linearity assumption is almost always an approximation, and sometimes a poor one. In Chapter 6 we see that we can improve upon least squares using ridge regression, the lasso, principal components regression, and other techniques. In that setting, the improvement is obtained by reducing the complexity of the linear model, and hence the variance of the estimates. But we are still using a linear model, which can only be improved so far! In this chapter we relax the linearity assumption while still attempting to maintain as much interpretability as possible. We do this by examining very simple extensions of linear models like polynomial regression and step functions, as well as more sophisticated approaches such as splines, local regression, and generalized additive models.

- *Polynomial regression* extends the linear model by adding extra predictors, obtained by raising each of the original predictors to a power. For example, a *cubic* regression uses three variables,  $X$ ,  $X^2$ , and  $X^3$ , as predictors. This approach provides a simple way to provide a non-linear fit to data.
- *Step functions* cut the range of a variable into  $K$  distinct regions in order to produce a qualitative variable. This has the effect of fitting a piecewise constant function.

- *Regression splines* are more flexible than polynomials and step functions, and in fact are an extension of the two. They involve dividing the range of  $X$  into  $K$  distinct regions. Within each region, a polynomial function is fit to the data. However, these polynomials are constrained so that they join smoothly at the region boundaries, or *knots*. Provided that the interval is divided into enough regions, this can produce an extremely flexible fit.
- *Smoothing splines* are similar to regression splines, but arise in a slightly different situation. Smoothing splines result from minimizing a residual sum of squares criterion subject to a smoothness penalty.
- *Local regression* is similar to splines, but differs in an important way. The regions are allowed to overlap, and indeed they do so in a very smooth way.
- *Generalized additive models* allow us to extend the methods above to deal with multiple predictors.

In Sections 7.1–7.6, we present a number of approaches for modeling the relationship between a response  $Y$  and a single predictor  $X$  in a flexible way. In Section 7.7, we show that these approaches can be seamlessly integrated in order to model a response  $Y$  as a function of several predictors  $X_1, \dots, X_p$ .

## 7.1 Polynomial Regression

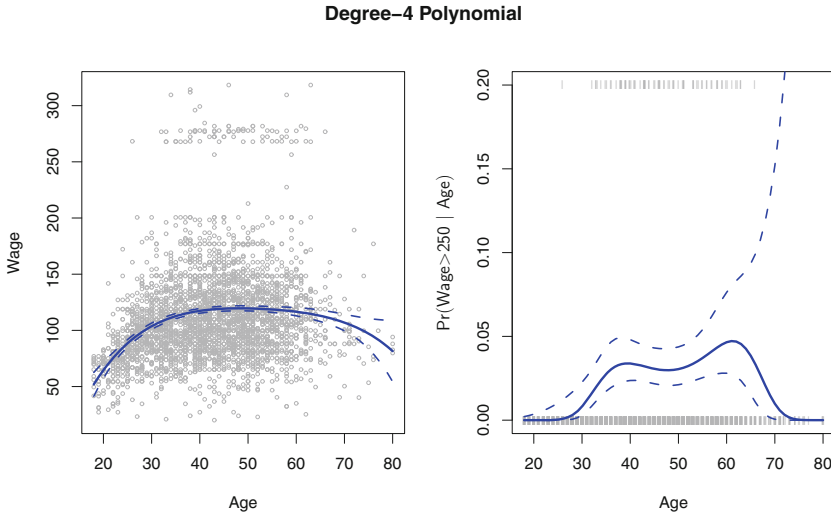
Historically, the standard way to extend linear regression to settings in which the relationship between the predictors and the response is non-linear has been to replace the standard linear model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

with a polynomial function

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \dots + \beta_d x_i^d + \epsilon_i, \quad (7.1)$$

where  $\epsilon_i$  is the error term. This approach is known as *polynomial regression*, and in fact we saw an example of this method in Section 3.3.2. For large enough degree  $d$ , a polynomial regression allows us to produce an extremely non-linear curve. Notice that the coefficients in (7.1) can be easily estimated using least squares linear regression because this is just a standard linear model with predictors  $x_i, x_i^2, x_i^3, \dots, x_i^d$ . Generally speaking, it is unusual to use  $d$  greater than 3 or 4 because for large values of  $d$ , the polynomial curve can become overly flexible and can take on some very strange shapes. This is especially true near the boundary of the  $X$  variable.



**FIGURE 7.1.** The `Wage` data. Left: The solid blue curve is a degree-4 polynomial of `wage` (in thousands of dollars) as a function of `age`, fit by least squares. The dotted curves indicate an estimated 95 % confidence interval. Right: We model the binary event `wage>250` using logistic regression, again with a degree-4 polynomial. The fitted posterior probability of `wage` exceeding \$250,000 is shown in blue, along with an estimated 95 % confidence interval.

The left-hand panel in Figure 7.1 is a plot of `wage` against `age` for the `Wage` data set, which contains income and demographic information for males who reside in the central Atlantic region of the United States. We see the results of fitting a degree-4 polynomial using least squares (solid blue curve). Even though this is a linear regression model like any other, the individual coefficients are not of particular interest. Instead, we look at the entire fitted function across a grid of 62 values for `age` from 18 to 80 in order to understand the relationship between `age` and `wage`.

In Figure 7.1, a pair of dotted curves accompanies the fit; these are  $(2\times)$  standard error curves. Let's see how these arise. Suppose we have computed the fit at a particular value of `age`,  $x_0$ :

$$\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0 + \hat{\beta}_2 x_0^2 + \hat{\beta}_3 x_0^3 + \hat{\beta}_4 x_0^4. \quad (7.2)$$

What is the variance of the fit, i.e.  $\text{Var} \hat{f}(x_0)$ ? Least squares returns variance estimates for each of the fitted coefficients  $\hat{\beta}_j$ , as well as the covariances between pairs of coefficient estimates. We can use these to compute the estimated variance of  $\hat{f}(x_0)$ .<sup>1</sup> The estimated *pointwise* standard error of  $\hat{f}(x_0)$  is the square-root of this variance. This computation is repeated

<sup>1</sup>If  $\hat{\mathbf{C}}$  is the  $5 \times 5$  covariance matrix of the  $\hat{\beta}_j$ , and if  $\ell_0^T = (1, x_0, x_0^2, x_0^3, x_0^4)$ , then  $\text{Var}[\hat{f}(x_0)] = \ell_0^T \hat{\mathbf{C}} \ell_0$ .

at each reference point  $x_0$ , and we plot the fitted curve, as well as twice the standard error on either side of the fitted curve. We plot twice the standard error because, for normally distributed error terms, this quantity corresponds to an approximate 95% confidence interval.

It seems like the wages in Figure 7.1 are from two distinct populations: there appears to be a *high earners* group earning more than \$250,000 per annum, as well as a *low earners* group. We can treat `wage` as a binary variable by splitting it into these two groups. Logistic regression can then be used to predict this binary response, using polynomial functions of `age` as predictors. In other words, we fit the model

$$\Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1x_i + \beta_2x_i^2 + \dots + \beta_dx_i^d)}{1 + \exp(\beta_0 + \beta_1x_i + \beta_2x_i^2 + \dots + \beta_dx_i^d)}. \quad (7.3)$$

The result is shown in the right-hand panel of Figure 7.1. The gray marks on the top and bottom of the panel indicate the ages of the high earners and the low earners. The solid blue curve indicates the fitted probabilities of being a high earner, as a function of `age`. The estimated 95% confidence interval is shown as well. We see that here the confidence intervals are fairly wide, especially on the right-hand side. Although the sample size for this data set is substantial ( $n = 3,000$ ), there are only 79 high earners, which results in a high variance in the estimated coefficients and consequently wide confidence intervals.

## 7.2 Step Functions

Using polynomial functions of the features as predictors in a linear model imposes a *global* structure on the non-linear function of  $X$ . We can instead use *step functions* in order to avoid imposing such a global structure. Here we break the range of  $X$  into *bins*, and fit a different constant in each bin. This amounts to converting a continuous variable into an *ordered categorical variable*.

step function

In greater detail, we create cutpoints  $c_1, c_2, \dots, c_K$  in the range of  $X$ , and then construct  $K + 1$  new variables

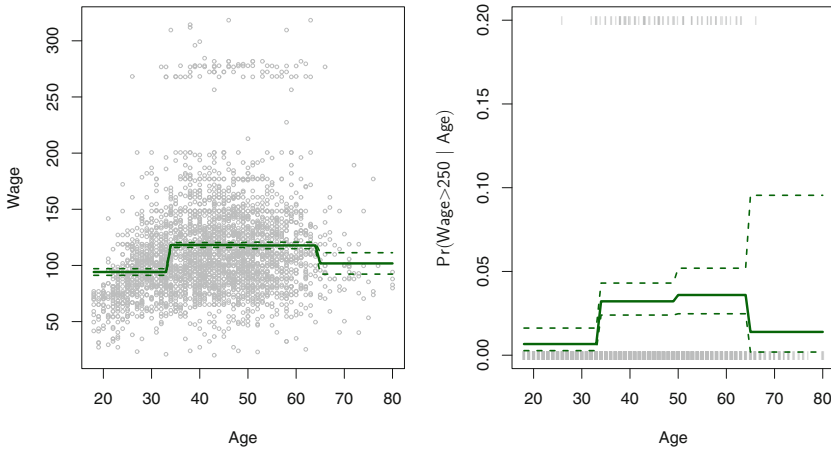
ordered categorical variable

$$\begin{aligned} C_0(X) &= I(X < c_1), \\ C_1(X) &= I(c_1 \leq X < c_2), \\ C_2(X) &= I(c_2 \leq X < c_3), \\ &\vdots \\ C_{K-1}(X) &= I(c_{K-1} \leq X < c_K), \\ C_K(X) &= I(c_K \leq X), \end{aligned} \quad (7.4)$$

where  $I(\cdot)$  is an *indicator function* that returns a 1 if the condition is true, and returns a 0 otherwise. For example,  $I(c_K \leq X)$  equals 1 if  $c_K \leq X$ , and

indicator function

## Piecewise Constant



**FIGURE 7.2.** The `Wage` data. Left: The solid curve displays the fitted value from a least squares regression of `wage` (in thousands of dollars) using step functions of `age`. The dotted curves indicate an estimated 95 % confidence interval. Right: We model the binary event `wage > 250` using logistic regression, again using step functions of `age`. The fitted posterior probability of `wage` exceeding \$250,000 is shown, along with an estimated 95 % confidence interval.

equals 0 otherwise. These are sometimes called *dummy* variables. Notice that for any value of  $X$ ,  $C_0(X) + C_1(X) + \dots + C_K(X) = 1$ , since  $X$  must be in exactly one of the  $K + 1$  intervals. We then use least squares to fit a linear model using  $C_1(X), C_2(X), \dots, C_K(X)$  as predictors<sup>2</sup>:

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \dots + \beta_K C_K(x_i) + \epsilon_i. \quad (7.5)$$

For a given value of  $X$ , at most one of  $C_1, C_2, \dots, C_K$  can be non-zero. Note that when  $X < c_1$ , all of the predictors in (7.5) are zero, so  $\beta_0$  can be interpreted as the mean value of  $Y$  for  $X < c_1$ . By comparison, (7.5) predicts a response of  $\beta_0 + \beta_j$  for  $c_j \leq X < c_{j+1}$ , so  $\beta_j$  represents the average increase in the response for  $X$  in  $c_j \leq X < c_{j+1}$  relative to  $X < c_1$ .

An example of fitting step functions to the `Wage` data from Figure 7.1 is shown in the left-hand panel of Figure 7.2. We also fit the logistic regression model

<sup>2</sup>We exclude  $C_0(X)$  as a predictor in (7.5) because it is redundant with the intercept. This is similar to the fact that we need only two dummy variables to code a qualitative variable with three levels, provided that the model will contain an intercept. The decision to exclude  $C_0(X)$  instead of some other  $C_k(X)$  in (7.5) is arbitrary. Alternatively, we could include  $C_0(X), C_1(X), \dots, C_K(X)$ , and exclude the intercept.

$$\Pr(y_i > 250|x_i) = \frac{\exp(\beta_0 + \beta_1 C_1(x_i) + \dots + \beta_K C_K(x_i))}{1 + \exp(\beta_0 + \beta_1 C_1(x_i) + \dots + \beta_K C_K(x_i))} \quad (7.6)$$

in order to predict the probability that an individual is a high earner on the basis of **age**. The right-hand panel of Figure 7.2 displays the fitted posterior probabilities obtained using this approach.

Unfortunately, unless there are natural breakpoints in the predictors, piecewise-constant functions can miss the action. For example, in the left-hand panel of Figure 7.2, the first bin clearly misses the increasing trend of **wage** with **age**. Nevertheless, step function approaches are very popular in biostatistics and epidemiology, among other disciplines. For example, 5-year age groups are often used to define the bins.

### 7.3 Basis Functions

Polynomial and piecewise-constant regression models are in fact special cases of a *basis function* approach. The idea is to have at hand a family of functions or transformations that can be applied to a variable  $X$ :  $b_1(X), b_2(X), \dots, b_K(X)$ . Instead of fitting a linear model in  $X$ , we fit the model

basis  
function

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \beta_3 b_3(x_i) + \dots + \beta_K b_K(x_i) + \epsilon_i. \quad (7.7)$$

Note that the basis functions  $b_1(\cdot), b_2(\cdot), \dots, b_K(\cdot)$  are fixed and known. (In other words, we choose the functions ahead of time.) For polynomial regression, the basis functions are  $b_j(x_i) = x_i^j$ , and for piecewise constant functions they are  $b_j(x_i) = I(c_j \leq x_i < c_{j+1})$ . We can think of (7.7) as a standard linear model with predictors  $b_1(x_i), b_2(x_i), \dots, b_K(x_i)$ . Hence, we can use least squares to estimate the unknown regression coefficients in (7.7). Importantly, this means that all of the inference tools for linear models that are discussed in Chapter 3, such as standard errors for the coefficient estimates and F-statistics for the model's overall significance, are available in this setting.

Thus far we have considered the use of polynomial functions and piecewise constant functions for our basis functions; however, many alternatives are possible. For instance, we can use wavelets or Fourier series to construct basis functions. In the next section, we investigate a very common choice for a basis function: *regression splines*.

regression  
spline

## 7.4 Regression Splines

Now we discuss a flexible class of basis functions that extends upon the polynomial regression and piecewise constant regression approaches that we have just seen.

### 7.4.1 Piecewise Polynomials

Instead of fitting a high-degree polynomial over the entire range of  $X$ , *piecewise polynomial regression* involves fitting separate low-degree polynomials over different regions of  $X$ . For example, a piecewise cubic polynomial works by fitting a cubic regression model of the form

piecewise  
polynomial  
regression

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i, \quad (7.8)$$

where the coefficients  $\beta_0$ ,  $\beta_1$ ,  $\beta_2$ , and  $\beta_3$  differ in different parts of the range of  $X$ . The points where the coefficients change are called *knots*.

knot

For example, a piecewise cubic with no knots is just a standard cubic polynomial, as in (7.1) with  $d = 3$ . A piecewise cubic polynomial with a single knot at a point  $c$  takes the form

$$y_i = \begin{cases} \beta_{01} + \beta_{11}x_i + \beta_{21}x_i^2 + \beta_{31}x_i^3 + \epsilon_i & \text{if } x_i < c; \\ \beta_{02} + \beta_{12}x_i + \beta_{22}x_i^2 + \beta_{32}x_i^3 + \epsilon_i & \text{if } x_i \geq c. \end{cases}$$

In other words, we fit two different polynomial functions to the data, one on the subset of the observations with  $x_i < c$ , and one on the subset of the observations with  $x_i \geq c$ . The first polynomial function has coefficients  $\beta_{01}, \beta_{11}, \beta_{21}, \beta_{31}$ , and the second has coefficients  $\beta_{02}, \beta_{12}, \beta_{22}, \beta_{32}$ . Each of these polynomial functions can be fit using least squares applied to simple functions of the original predictor.

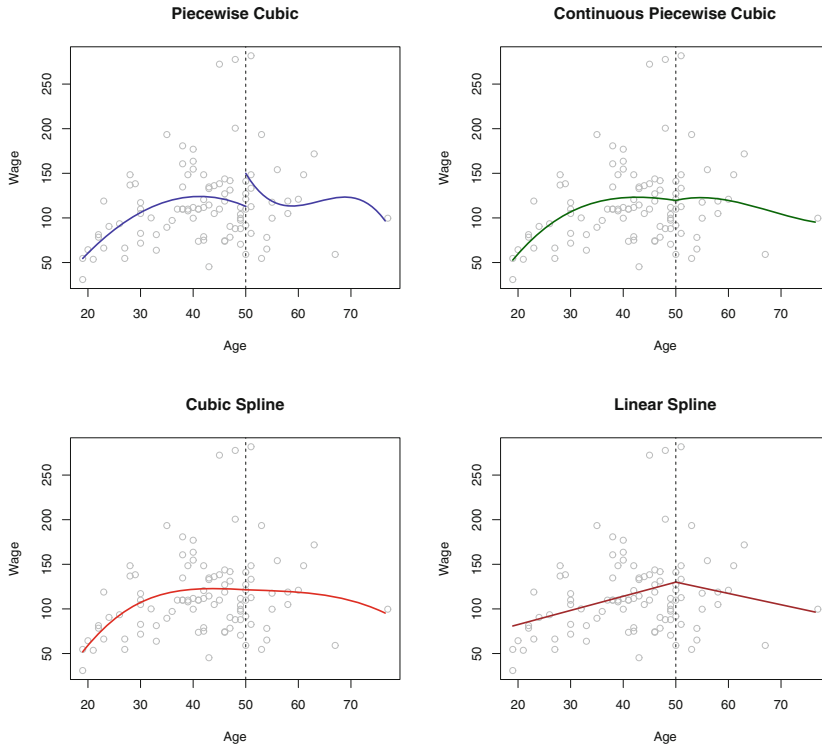
Using more knots leads to a more flexible piecewise polynomial. In general, if we place  $K$  different knots throughout the range of  $X$ , then we will end up fitting  $K + 1$  different cubic polynomials. Note that we do not need to use a cubic polynomial. For example, we can instead fit piecewise linear functions. In fact, our piecewise constant functions of Section 7.2 are piecewise polynomials of degree 0!

The top left panel of Figure 7.3 shows a piecewise cubic polynomial fit to a subset of the `Wage` data, with a single knot at `age=50`. We immediately see a problem: the function is discontinuous and looks ridiculous! Since each polynomial has four parameters, we are using a total of eight *degrees of freedom* in fitting this piecewise polynomial model.

degrees of  
freedom

### 7.4.2 Constraints and Splines

The top left panel of Figure 7.3 looks wrong because the fitted curve is just too flexible. To remedy this problem, we can fit a piecewise polynomial



**FIGURE 7.3.** Various piecewise polynomials are fit to a subset of the `Wage` data, with a knot at `age=50`. Top Left: The cubic polynomials are unconstrained. Top Right: The cubic polynomials are constrained to be continuous at `age=50`. Bottom Left: The cubic polynomials are constrained to be continuous, and to have continuous first and second derivatives. Bottom Right: A linear spline is shown, which is constrained to be continuous.

under the *constraint* that the fitted curve must be continuous. In other words, there cannot be a jump when `age=50`. The top right plot in Figure 7.3 shows the resulting fit. This looks better than the top left plot, but the V-shaped join looks unnatural.

In the lower left plot, we have added two additional constraints: now both the first and second *derivatives* of the piecewise polynomials are continuous at `age=50`. In other words, we are requiring that the piecewise polynomial be not only continuous when `age=50`, but also very *smooth*. Each constraint that we impose on the piecewise cubic polynomials effectively frees up one degree of freedom, by reducing the complexity of the resulting piecewise polynomial fit. So in the top left plot, we are using eight degrees of freedom, but in the bottom left plot we imposed three constraints (continuity, continuity of the first derivative, and continuity of the second derivative) and so are left with five degrees of freedom. The curve in the bottom left

derivative



plot is called a *cubic spline*.<sup>3</sup> In general, a cubic spline with  $K$  knots uses a total of  $4 + K$  degrees of freedom.

cubic spline

In Figure 7.3, the lower right plot is a *linear spline*, which is continuous at `age=50`. The general definition of a degree- $d$  spline is that it is a piecewise degree- $d$  polynomial, with continuity in derivatives up to degree  $d - 1$  at each knot. Therefore, a linear spline is obtained by fitting a line in each region of the predictor space defined by the knots, requiring continuity at each knot.

linear spline

In Figure 7.3, there is a single knot at `age=50`. Of course, we could add more knots, and impose continuity at each.

### 7.4.3 The Spline Basis Representation

The regression splines that we just saw in the previous section may have seemed somewhat complex: how can we fit a piecewise degree- $d$  polynomial under the constraint that it (and possibly its first  $d - 1$  derivatives) be continuous? It turns out that we can use the basis model (7.7) to represent a regression spline. A cubic spline with  $K$  knots can be modeled as

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i, \quad (7.9)$$

for an appropriate choice of basis functions  $b_1, b_2, \dots, b_{K+3}$ . The model (7.9) can then be fit using least squares.

Just as there were several ways to represent polynomials, there are also many equivalent ways to represent cubic splines using different choices of basis functions in (7.9). The most direct way to represent a cubic spline using (7.9) is to start off with a basis for a cubic polynomial—namely,  $x, x^2, x^3$ —and then add one *truncated power basis* function per knot. A truncated power basis function is defined as

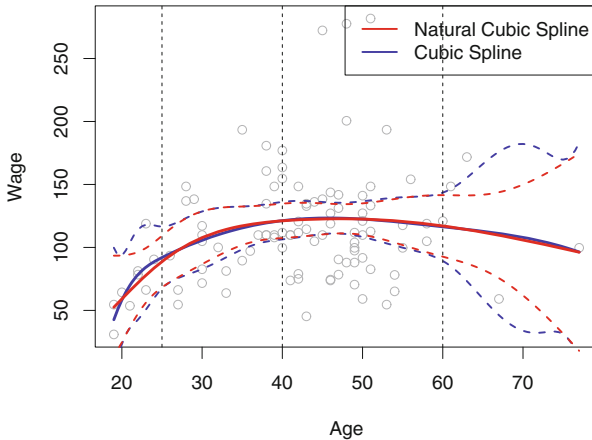
truncated power basis

$$h(x, \xi) = (x - \xi)_+^3 = \begin{cases} (x - \xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise,} \end{cases} \quad (7.10)$$

where  $\xi$  is the knot. One can show that adding a term of the form  $\beta_4 h(x, \xi)$  to the model (7.8) for a cubic polynomial will lead to a discontinuity in only the third derivative at  $\xi$ ; the function will remain continuous, with continuous first and second derivatives, at each of the knots.

In other words, in order to fit a cubic spline to a data set with  $K$  knots, we perform least squares regression with an intercept and  $3 + K$  predictors, of the form  $X, X^2, X^3, h(X, \xi_1), h(X, \xi_2), \dots, h(X, \xi_K)$ , where  $\xi_1, \dots, \xi_K$  are the knots. This amounts to estimating a total of  $K + 4$  regression coefficients; for this reason, fitting a cubic spline with  $K$  knots uses  $K + 4$  degrees of freedom.

<sup>3</sup>Cubic splines are popular because most human eyes cannot detect the discontinuity at the knots.



**FIGURE 7.4.** A cubic spline and a natural cubic spline, with three knots, fit to a subset of the `Wage` data.

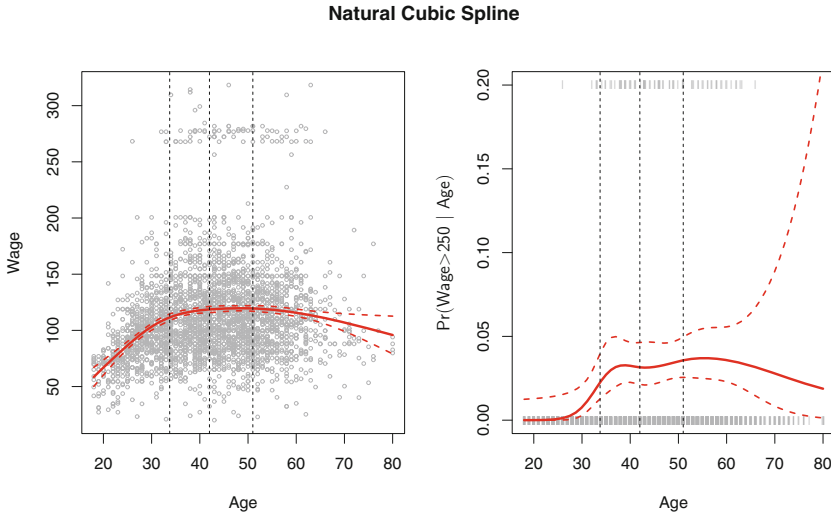
Unfortunately, splines can have high variance at the outer range of the predictors—that is, when  $X$  takes on either a very small or very large value. Figure 7.4 shows a fit to the `Wage` data with three knots. We see that the confidence bands in the boundary region appear fairly wild. A *natural spline* is a regression spline with additional *boundary constraints*: the function is required to be linear at the boundary (in the region where  $X$  is smaller than the smallest knot, or larger than the largest knot). This additional constraint means that natural splines generally produce more stable estimates at the boundaries. In Figure 7.4, a natural cubic spline is also displayed as a red line. Note that the corresponding confidence intervals are narrower.

natural  
spline

#### 7.4.4 Choosing the Number and Locations of the Knots

When we fit a spline, where should we place the knots? The regression spline is most flexible in regions that contain a lot of knots, because in those regions the polynomial coefficients can change rapidly. Hence, one option is to place more knots in places where we feel the function might vary most rapidly, and to place fewer knots where it seems more stable. While this option can work well, in practice it is common to place knots in a uniform fashion. One way to do this is to specify the desired degrees of freedom, and then have the software automatically place the corresponding number of knots at uniform quantiles of the data.

Figure 7.5 shows an example on the `Wage` data. As in Figure 7.4, we have fit a natural cubic spline with three knots, except this time the knot locations were chosen automatically as the 25th, 50th, and 75th percentiles

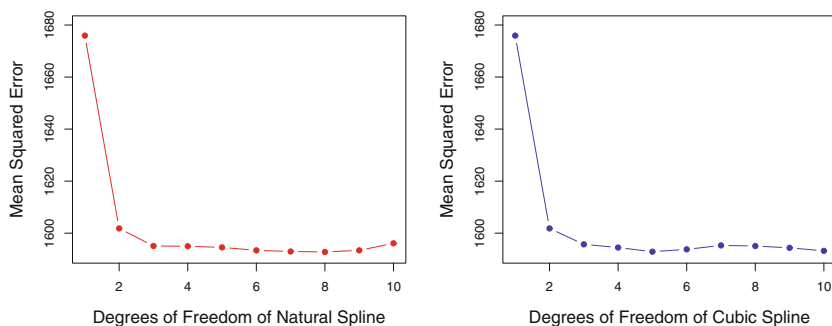


**FIGURE 7.5.** A natural cubic spline function with four degrees of freedom is fit to the `Wage` data. Left: A spline is fit to `wage` (in thousands of dollars) as a function of `age`. Right: Logistic regression is used to model the binary event `wage > 250` as a function of `age`. The fitted posterior probability of `wage` exceeding \$250,000 is shown.

of `age`. This was specified by requesting four degrees of freedom. The argument by which four degrees of freedom leads to three interior knots is somewhat technical.<sup>4</sup>

How many knots should we use, or equivalently how many degrees of freedom should our spline contain? One option is to try out different numbers of knots and see which produces the best looking curve. A somewhat more objective approach is to use cross-validation, as discussed in Chapters 5 and 6. With this method, we remove a portion of the data (say 10%), fit a spline with a certain number of knots to the remaining data, and then use the spline to make predictions for the held-out portion. We repeat this process multiple times until each observation has been left out once, and then compute the overall cross-validated RSS. This procedure can be repeated for different numbers of knots  $K$ . Then the value of  $K$  giving the smallest RSS is chosen.

<sup>4</sup>There are actually five knots, including the two boundary knots. A cubic spline with five knots would have nine degrees of freedom. But natural cubic splines have two additional *natural* constraints at each boundary to enforce linearity, resulting in  $9 - 4 = 5$  degrees of freedom. Since this includes a constant, which is absorbed in the intercept, we count it as four degrees of freedom.



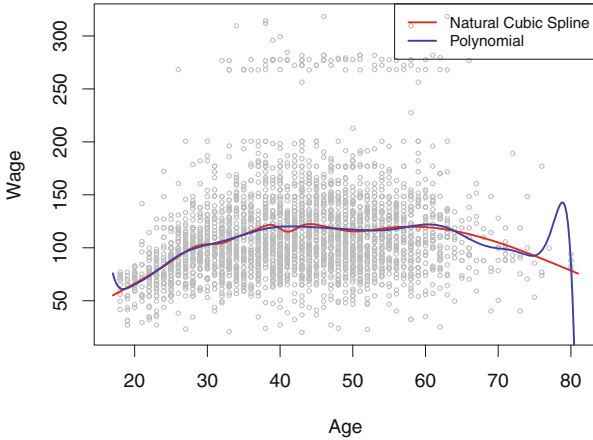
**FIGURE 7.6.** Ten-fold cross-validated mean squared errors for selecting the degrees of freedom when fitting splines to the `Wage` data. The response is `wage` and the predictor `age`. Left: A natural cubic spline. Right: A cubic spline.

Figure 7.6 shows ten-fold cross-validated mean squared errors for splines with various degrees of freedom fit to the `Wage` data. The left-hand panel corresponds to a natural spline and the right-hand panel to a cubic spline. The two methods produce almost identical results, with clear evidence that a one-degree fit (a linear regression) is not adequate. Both curves flatten out quickly, and it seems that three degrees of freedom for the natural spline and four degrees of freedom for the cubic spline are quite adequate.

In Section 7.7 we fit additive spline models simultaneously on several variables at a time. This could potentially require the selection of degrees of freedom for each variable. In cases like this we typically adopt a more pragmatic approach and set the degrees of freedom to a fixed number, say four, for all terms.

#### 7.4.5 Comparison to Polynomial Regression

Regression splines often give superior results to polynomial regression. This is because unlike polynomials, which must use a high degree (exponent in the highest monomial term, e.g.  $X^{15}$ ) to produce flexible fits, splines introduce flexibility by increasing the number of knots but keeping the degree fixed. Generally, this approach produces more stable estimates. Splines also allow us to place more knots, and hence flexibility, over regions where the function  $f$  seems to be changing rapidly, and fewer knots where  $f$  appears more stable. Figure 7.7 compares a natural cubic spline with 15 degrees of freedom to a degree-15 polynomial on the `Wage` data set. The extra flexibility in the polynomial produces undesirable results at the boundaries, while the natural cubic spline still provides a reasonable fit to the data.



**FIGURE 7.7.** On the `Wage` data set, a natural cubic spline with 15 degrees of freedom is compared to a degree-15 polynomial. Polynomials can show wild behavior, especially near the tails.

## 7.5 Smoothing Splines

### 7.5.1 An Overview of Smoothing Splines

In the last section we discussed regression splines, which we create by specifying a set of knots, producing a sequence of basis functions, and then using least squares to estimate the spline coefficients. We now introduce a somewhat different approach that also produces a spline.

In fitting a smooth curve to a set of data, what we really want to do is find some function, say  $g(x)$ , that fits the observed data well: that is, we want  $\text{RSS} = \sum_{i=1}^n (y_i - g(x_i))^2$  to be small. However, there is a problem with this approach. If we don't put any constraints on  $g(x_i)$ , then we can always make RSS zero simply by choosing  $g$  such that it *interpolates* all of the  $y_i$ . Such a function would woefully overfit the data—it would be far too flexible. What we really want is a function  $g$  that makes RSS small, but that is also *smooth*.

How might we ensure that  $g$  is smooth? There are a number of ways to do this. A natural approach is to find the function  $g$  that minimizes

$$\sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt \quad (7.11)$$

where  $\lambda$  is a nonnegative *tuning parameter*. The function  $g$  that minimizes (7.11) is known as a *smoothing spline*.

What does (7.11) mean? Equation 7.11 takes the “Loss+Penalty” formulation that we encounter in the context of ridge regression and the lasso in Chapter 6. The term  $\sum_{i=1}^n (y_i - g(x_i))^2$  is a *loss function* that encourages  $g$  to fit the data well, and the term  $\lambda \int g''(t)^2 dt$  is a *penalty term* smoothing spline  
loss function

that penalizes the variability in  $g$ . The notation  $g''(t)$  indicates the second derivative of the function  $g$ . The first derivative  $g'(t)$  measures the slope of a function at  $t$ , and the second derivative corresponds to the amount by which the slope is changing. Hence, broadly speaking, the second derivative of a function is a measure of its *roughness*: it is large in absolute value if  $g(t)$  is very wiggly near  $t$ , and it is close to zero otherwise. (The second derivative of a straight line is zero; note that a line is perfectly smooth.) The  $\int$  notation is an *integral*, which we can think of as a summation over the range of  $t$ . In other words,  $\int g''(t)^2 dt$  is simply a measure of the total change in the function  $g'(t)$ , over its entire range. If  $g$  is very smooth, then  $g'(t)$  will be close to constant and  $\int g''(t)^2 dt$  will take on a small value. Conversely, if  $g$  is jumpy and variable then  $g'(t)$  will vary significantly and  $\int g''(t)^2 dt$  will take on a large value. Therefore, in (7.11),  $\lambda \int g''(t)^2 dt$  encourages  $g$  to be smooth. The larger the value of  $\lambda$ , the smoother  $g$  will be.

When  $\lambda = 0$ , then the penalty term in (7.11) has no effect, and so the function  $g$  will be very jumpy and will exactly interpolate the training observations. When  $\lambda \rightarrow \infty$ ,  $g$  will be perfectly smooth—it will just be a straight line that passes as closely as possible to the training points. In fact, in this case,  $g$  will be the linear least squares line, since the loss function in (7.11) amounts to minimizing the residual sum of squares. For an intermediate value of  $\lambda$ ,  $g$  will approximate the training observations but will be somewhat smooth. We see that  $\lambda$  controls the bias-variance trade-off of the smoothing spline.

The function  $g(x)$  that minimizes (7.11) can be shown to have some special properties: it is a piecewise cubic polynomial with knots at the unique values of  $x_1, \dots, x_n$ , and continuous first and second derivatives at each knot. Furthermore, it is linear in the region outside of the extreme knots. In other words, *the function  $g(x)$  that minimizes (7.11) is a natural cubic spline with knots at  $x_1, \dots, x_n$* ! However, it is not the same natural cubic spline that one would get if one applied the basis function approach described in Section 7.4.3 with knots at  $x_1, \dots, x_n$ —rather, it is a *shrunk* version of such a natural cubic spline, where the value of the tuning parameter  $\lambda$  in (7.11) controls the level of shrinkage.

### 7.5.2 Choosing the Smoothing Parameter $\lambda$

We have seen that a smoothing spline is simply a natural cubic spline with knots at every unique value of  $x_i$ . It might seem that a smoothing spline will have far too many degrees of freedom, since a knot at each data point allows a great deal of flexibility. But the tuning parameter  $\lambda$  controls the roughness of the smoothing spline, and hence the *effective degrees of freedom*. It is possible to show that as  $\lambda$  increases from 0 to  $\infty$ , the effective degrees of freedom, which we write  $df_\lambda$ , decrease from  $n$  to 2.

effective  
degrees of  
freedom

In the context of smoothing splines, why do we discuss *effective* degrees of freedom instead of degrees of freedom? Usually degrees of freedom refer

to the number of free parameters, such as the number of coefficients fit in a polynomial or cubic spline. Although a smoothing spline has  $n$  parameters and hence  $n$  nominal degrees of freedom, these  $n$  parameters are heavily constrained or shrunk down. Hence  $df_\lambda$  is a measure of the flexibility of the smoothing spline—the higher it is, the more flexible (and the lower-bias but higher-variance) the smoothing spline. The definition of effective degrees of freedom is somewhat technical. We can write

$$\hat{\mathbf{g}}_\lambda = \mathbf{S}_\lambda \mathbf{y}, \quad (7.12)$$

where  $\hat{\mathbf{g}}$  is the solution to (7.11) for a particular choice of  $\lambda$ —that is, it is a  $n$ -vector containing the fitted values of the smoothing spline at the training points  $x_1, \dots, x_n$ . Equation 7.12 indicates that the vector of fitted values when applying a smoothing spline to the data can be written as a  $n \times n$  matrix  $\mathbf{S}_\lambda$  (for which there is a formula) times the response vector  $\mathbf{y}$ . Then the effective degrees of freedom is defined to be

$$df_\lambda = \sum_{i=1}^n \{\mathbf{S}_\lambda\}_{ii}, \quad (7.13)$$

the sum of the diagonal elements of the matrix  $\mathbf{S}_\lambda$ .

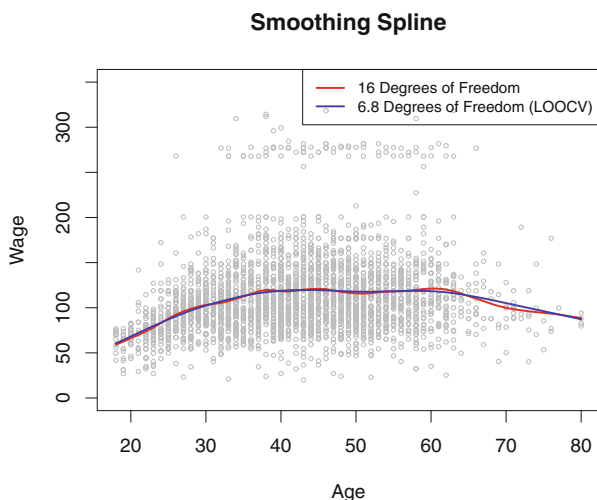
In fitting a smoothing spline, we do not need to select the number or location of the knots—there will be a knot at each training observation,  $x_1, \dots, x_n$ . Instead, we have another problem: we need to choose the value of  $\lambda$ . It should come as no surprise that one possible solution to this problem is cross-validation. In other words, we can find the value of  $\lambda$  that makes the cross-validated RSS as small as possible. It turns out that the *leave-one-out* cross-validation error (LOOCV) can be computed very efficiently for smoothing splines, with essentially the same cost as computing a single fit, using the following formula:

$$\text{RSS}_{cv}(\lambda) = \sum_{i=1}^n (y_i - \hat{g}_\lambda^{(-i)}(x_i))^2 = \sum_{i=1}^n \left[ \frac{y_i - \hat{g}_\lambda(x_i)}{1 - \{\mathbf{S}_\lambda\}_{ii}} \right]^2.$$

The notation  $\hat{g}_\lambda^{(-i)}(x_i)$  indicates the fitted value for this smoothing spline evaluated at  $x_i$ , where the fit uses all of the training observations except for the  $i$ th observation  $(x_i, y_i)$ . In contrast,  $\hat{g}_\lambda(x_i)$  indicates the smoothing spline function fit to all of the training observations and evaluated at  $x_i$ . This remarkable formula says that we can compute each of these *leave-one-out* fits using only  $\hat{g}_\lambda$ , the original fit to *all* of the data!<sup>5</sup> We have a very similar formula (5.2) on page 180 in Chapter 5 for least squares linear regression. Using (5.2), we can very quickly perform LOOCV for the regression splines discussed earlier in this chapter, as well as for least squares regression using arbitrary basis functions.

---

<sup>5</sup>The exact formulas for computing  $\hat{g}_\lambda(x_i)$  and  $\mathbf{S}_\lambda$  are very technical; however, efficient algorithms are available for computing these quantities.



**FIGURE 7.8.** Smoothing spline fits to the *Wage* data. The red curve results from specifying 16 effective degrees of freedom. For the blue curve,  $\lambda$  was found automatically by leave-one-out cross-validation, which resulted in 6.8 effective degrees of freedom.

Figure 7.8 shows the results from fitting a smoothing spline to the *Wage* data. The red curve indicates the fit obtained from pre-specifying that we would like a smoothing spline with 16 effective degrees of freedom. The blue curve is the smoothing spline obtained when  $\lambda$  is chosen using LOOCV; in this case, the value of  $\lambda$  chosen results in 6.8 effective degrees of freedom (computed using (7.13)). For this data, there is little discernible difference between the two smoothing splines, beyond the fact that the one with 16 degrees of freedom seems slightly wigglier. Since there is little difference between the two fits, the smoothing spline fit with 6.8 degrees of freedom is preferable, since in general simpler models are better unless the data provides evidence in support of a more complex model.

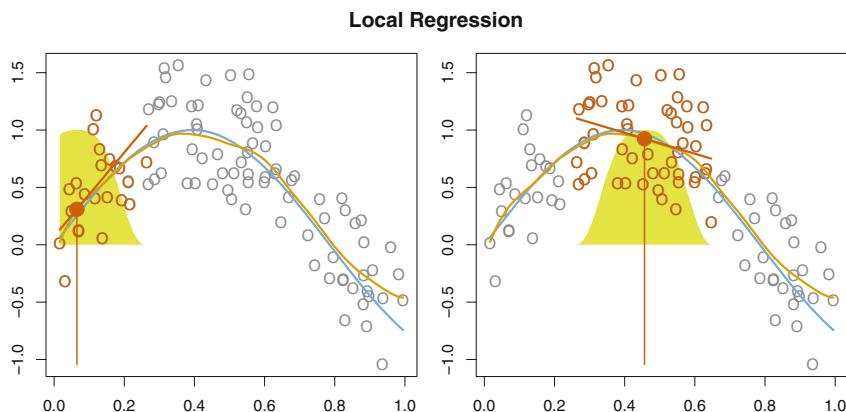
## 7.6 Local Regression

*Local regression* is a different approach for fitting flexible non-linear functions, which involves computing the fit at a target point  $x_0$  using only the nearby training observations. Figure 7.9 illustrates the idea on some simulated data, with one target point near 0.4, and another near the boundary at 0.05. In this figure the blue line represents the function  $f(x)$  from which the data were generated, and the light orange line corresponds to the local regression estimate  $\hat{f}(x)$ . Local regression is described in Algorithm 7.1.

local  
regression

Note that in Step 3 of Algorithm 7.1, the weights  $K_{i0}$  will differ for each value of  $x_0$ . In other words, in order to obtain the local regression fit at a new point, we need to fit a new weighted least squares regression model by





**FIGURE 7.9.** Local regression illustrated on some simulated data, where the blue curve represents  $f(x)$  from which the data were generated, and the light orange curve corresponds to the local regression estimate  $\hat{f}(x)$ . The orange colored points are local to the target point  $x_0$ , represented by the orange vertical line. The yellow bell-shape superimposed on the plot indicates weights assigned to each point, decreasing to zero with distance from the target point. The fit  $\hat{f}(x_0)$  at  $x_0$  is obtained by fitting a weighted linear regression (orange line segment), and using the fitted value at  $x_0$  (orange solid dot) as the estimate  $\hat{f}(x_0)$ .

minimizing (7.14) for a new set of weights. Local regression is sometimes referred to as a *memory-based* procedure, because like nearest-neighbors, we need all the training data each time we wish to compute a prediction. We will avoid getting into the technical details of local regression here—there are books written on the topic.

In order to perform local regression, there are a number of choices to be made, such as how to define the weighting function  $K$ , and whether to fit a linear, constant, or quadratic regression in Step 3 above. (Equation 7.14 corresponds to a linear regression.) While all of these choices make some difference, the most important choice is the *span*  $s$ , defined in Step 1 above. The span plays a role like that of the tuning parameter  $\lambda$  in smoothing splines: it controls the flexibility of the non-linear fit. The smaller the value of  $s$ , the more *local* and wiggly will be our fit; alternatively, a very large value of  $s$  will lead to a global fit to the data using all of the training observations. We can again use cross-validation to choose  $s$ , or we can specify it directly. Figure 7.10 displays local linear regression fits on the *Wage* data, using two values of  $s$ : 0.7 and 0.2. As expected, the fit obtained using  $s = 0.7$  is smoother than that obtained using  $s = 0.2$ .

The idea of local regression can be generalized in many different ways. In a setting with multiple features  $X_1, X_2, \dots, X_p$ , one very useful generalization involves fitting a multiple linear regression model that is global in some variables, but local in another, such as time. Such *varying coefficient*

---

**Algorithm 7.1** *Local Regression At  $X = x_0$* 

---

1. Gather the fraction  $s = k/n$  of training points whose  $x_i$  are closest to  $x_0$ .
2. Assign a weight  $K_{i0} = K(x_i, x_0)$  to each point in this neighborhood, so that the point furthest from  $x_0$  has weight zero, and the closest has the highest weight. All but these  $k$  nearest neighbors get weight zero.
3. Fit a *weighted least squares regression* of the  $y_i$  on the  $x_i$  using the aforementioned weights, by finding  $\hat{\beta}_0$  and  $\hat{\beta}_1$  that minimize

$$\sum_{i=1}^n K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2. \quad (7.14)$$

4. The fitted value at  $x_0$  is given by  $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$ .
- 

*models* are a useful way of adapting a model to the most recently gathered data. Local regression also generalizes very naturally when we want to fit models that are local in a pair of variables  $X_1$  and  $X_2$ , rather than one. We can simply use two-dimensional neighborhoods, and fit bivariate linear regression models using the observations that are near each target point in two-dimensional space. Theoretically the same approach can be implemented in higher dimensions, using linear regressions fit to  $p$ -dimensional neighborhoods. However, local regression can perform poorly if  $p$  is much larger than about 3 or 4 because there will generally be very few training observations close to  $x_0$ . Nearest-neighbors regression, discussed in Chapter 3, suffers from a similar problem in high dimensions.

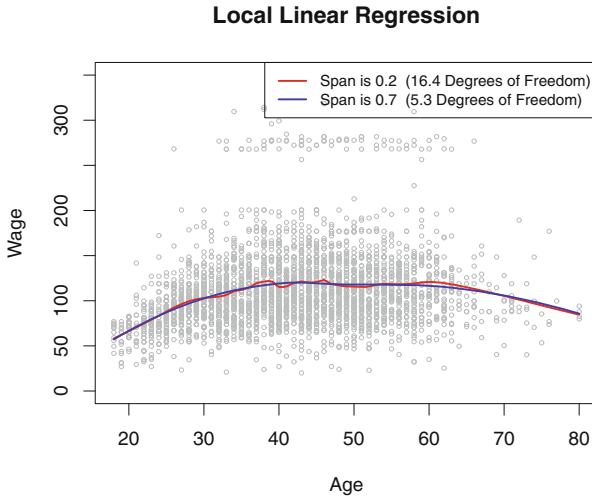
varying  
coefficient  
model

## 7.7 Generalized Additive Models

In Sections 7.1–7.6, we present a number of approaches for flexibly predicting a response  $Y$  on the basis of a single predictor  $X$ . These approaches can be seen as extensions of simple linear regression. Here we explore the problem of flexibly predicting  $Y$  on the basis of several predictors,  $X_1, \dots, X_p$ . This amounts to an extension of multiple linear regression.

*Generalized additive models* (GAMs) provide a general framework for extending a standard linear model by allowing non-linear functions of each of the variables, while maintaining *additivity*. Just like linear models, GAMs can be applied with both quantitative and qualitative responses. We first examine GAMs for a quantitative response in Section 7.7.1, and then for a qualitative response in Section 7.7.2.

generalized  
additive  
model  
additivity



**FIGURE 7.10.** Local linear fits to the `Wage` data. The span specifies the fraction of the data used to compute the fit at each target point.

### 7.7.1 GAMs for Regression Problems

A natural way to extend the multiple linear regression model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

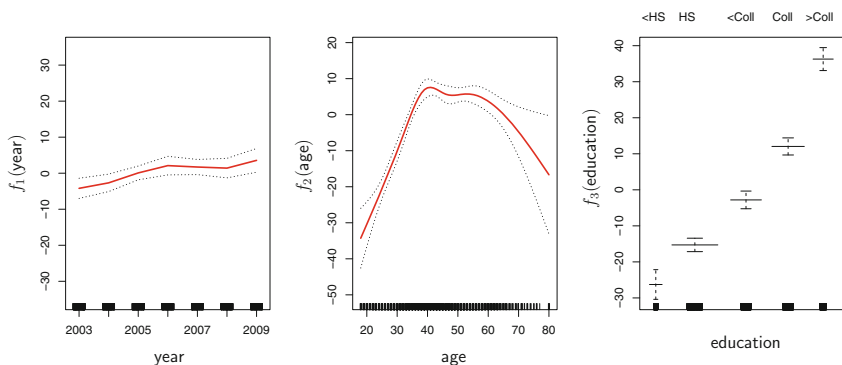
in order to allow for non-linear relationships between each feature and the response is to replace each linear component  $\beta_j x_{ij}$  with a (smooth) non-linear function  $f_j(x_{ij})$ . We would then write the model as

$$\begin{aligned} y_i &= \beta_0 + \sum_{j=1}^p f_j(x_{ij}) + \epsilon_i \\ &= \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i. \end{aligned} \quad (7.15)$$

This is an example of a GAM. It is called an *additive* model because we calculate a separate  $f_j$  for each  $X_j$ , and then add together all of their contributions.

In Sections 7.1–7.6, we discuss many methods for fitting functions to a single variable. The beauty of GAMs is that we can use these methods as building blocks for fitting an additive model. In fact, for most of the methods that we have seen so far in this chapter, this can be done fairly trivially. Take, for example, natural splines, and consider the task of fitting the model

$$\text{wage} = \beta_0 + f_1(\text{year}) + f_2(\text{age}) + f_3(\text{education}) + \epsilon \quad (7.16)$$



**FIGURE 7.11.** For the `Wage` data, plots of the relationship between each feature and the response, `wage`, in the fitted model (7.16). Each plot displays the fitted function and pointwise standard errors. The first two functions are natural splines in `year` and `age`, with four and five degrees of freedom, respectively. The third function is a step function, fit to the qualitative variable `education`.

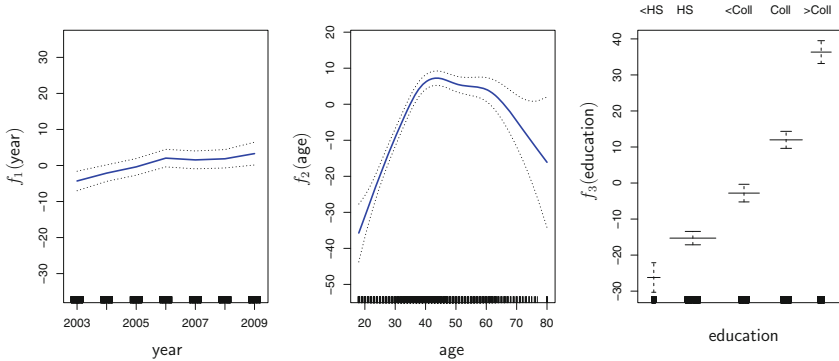
on the `Wage` data. Here `year` and `age` are quantitative variables, and `education` is a qualitative variable with five levels: `<HS`, `HS`, `<Coll`, `Coll`, `>Coll`, referring to the amount of high school or college education that an individual has completed. We fit the first two functions using natural splines. We fit the third function using a separate constant for each level, via the usual dummy variable approach of Section 3.3.1.

Figure 7.11 shows the results of fitting the model (7.16) using least squares. This is easy to do, since as discussed in Section 7.4, natural splines can be constructed using an appropriately chosen set of basis functions. Hence the entire model is just a big regression onto spline basis variables and dummy variables, all packed into one big regression matrix.

Figure 7.11 can be easily interpreted. The left-hand panel indicates that holding `age` and `education` fixed, `wage` tends to increase slightly with `year`; this may be due to inflation. The center panel indicates that holding `education` and `year` fixed, `wage` tends to be highest for intermediate values of `age`, and lowest for the very young and very old. The right-hand panel indicates that holding `year` and `age` fixed, `wage` tends to increase with `education`: the more educated a person is, the higher their salary, on average. All of these findings are intuitive.

Figure 7.12 shows a similar triple of plots, but this time  $f_1$  and  $f_2$  are smoothing splines with four and five degrees of freedom, respectively. Fitting a GAM with a smoothing spline is not quite as simple as fitting a GAM with a natural spline, since in the case of smoothing splines, least squares cannot be used. However, standard software such as the `gam()` function in `R` can be used to fit GAMs using smoothing splines, via an approach known as *backfitting*. This method fits a model involving multiple predictors by

backfitting



**FIGURE 7.12.** Details are as in Figure 7.11, but now  $f_1$  and  $f_2$  are smoothing splines with four and five degrees of freedom, respectively.

repeatedly updating the fit for each predictor in turn, holding the others fixed. The beauty of this approach is that each time we update a function, we simply apply the fitting method for that variable to a *partial residual*.<sup>6</sup>

The fitted functions in Figures 7.11 and 7.12 look rather similar. In most situations, the differences in the GAMs obtained using smoothing splines versus natural splines are small.

We do not have to use splines as the building blocks for GAMs: we can just as well use local regression, polynomial regression, or any combination of the approaches seen earlier in this chapter in order to create a GAM. GAMs are investigated in further detail in the lab at the end of this chapter.

### Pros and Cons of GAMs

Before we move on, let us summarize the advantages and limitations of a GAM.

- ▲ GAMs allow us to fit a non-linear  $f_j$  to each  $X_j$ , so that we can automatically model non-linear relationships that standard linear regression will miss. This means that we do not need to manually try out many different transformations on each variable individually.
- ▲ The non-linear fits can potentially make more accurate predictions for the response  $Y$ .
- ▲ Because the model is additive, we can still examine the effect of each  $X_j$  on  $Y$  individually while holding all of the other variables fixed. Hence if we are interested in inference, GAMs provide a useful representation.

<sup>6</sup>A partial residual for  $X_3$ , for example, has the form  $r_i = y_i - f_1(x_{i1}) - f_2(x_{i2})$ . If we know  $f_1$  and  $f_2$ , then we can fit  $f_3$  by treating this residual as a response in a non-linear regression on  $X_3$ .

- ▲ The smoothness of the function  $f_j$  for the variable  $X_j$  can be summarized via degrees of freedom.
- ◆ The main limitation of GAMs is that the model is restricted to be additive. With many variables, important interactions can be missed. However, as with linear regression, we can manually add interaction terms to the GAM model by including additional predictors of the form  $X_j \times X_k$ . In addition we can add low-dimensional interaction functions of the form  $f_{jk}(X_j, X_k)$  into the model; such terms can be fit using two-dimensional smoothers such as local regression, or two-dimensional splines (not covered here).

For fully general models, we have to look for even more flexible approaches such as random forests and boosting, described in Chapter 8. GAMs provide a useful compromise between linear and fully nonparametric models.

### 7.7.2 GAMs for Classification Problems

GAMs can also be used in situations where  $Y$  is qualitative. For simplicity, here we will assume  $Y$  takes on values zero or one, and let  $p(X) = \Pr(Y = 1|X)$  be the conditional probability (given the predictors) that the response equals one. Recall the logistic regression model (4.6):

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p. \quad (7.17)$$

This *logit* is the log of the odds of  $P(Y = 1|X)$  versus  $P(Y = 0|X)$ , which (7.17) represents as a linear function of the predictors. A natural way to extend (7.17) to allow for non-linear relationships is to use the model

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p). \quad (7.18)$$

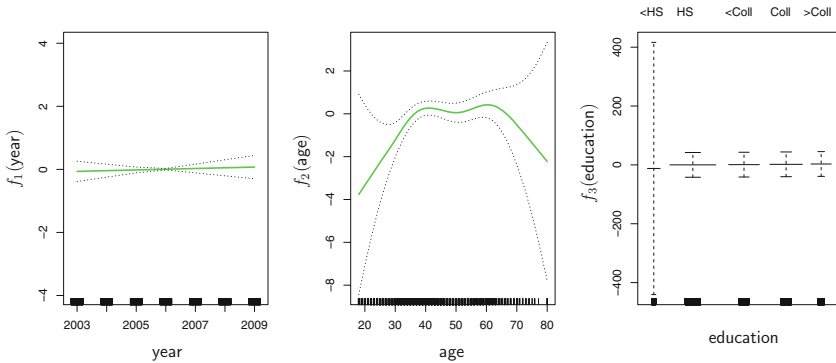
Equation 7.18 is a logistic regression GAM. It has all the same pros and cons as discussed in the previous section for quantitative responses.

We fit a GAM to the **Wage** data in order to predict the probability that an individual's income exceeds \$250,000 per year. The GAM that we fit takes the form

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 \times \text{year} + f_2(\text{age}) + f_3(\text{education}), \quad (7.19)$$

where

$$p(X) = \Pr(\text{wage} > 250 | \text{year}, \text{age}, \text{education}).$$



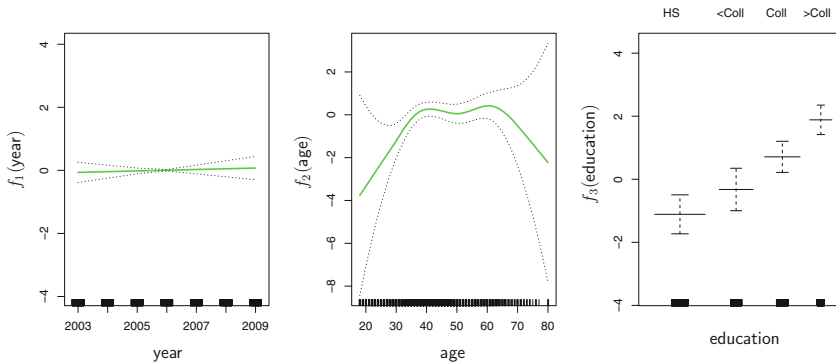
**FIGURE 7.13.** For the `Wage` data, the logistic regression GAM given in (7.19) is fit to the binary response  $\mathbf{I}(\text{wage} > 250)$ . Each plot displays the fitted function and pointwise standard errors. The first function is linear in `year`, the second function a smoothing spline with five degrees of freedom in `age`, and the third a step function for `education`. There are very wide standard errors for the first level `<HS` of `education`.

Once again  $f_2$  is fit using a smoothing spline with five degrees of freedom, and  $f_3$  is fit as a step function, by creating dummy variables for each of the levels of education. The resulting fit is shown in Figure 7.13. The last panel looks suspicious, with very wide confidence intervals for level `<HS`. In fact, there are no ones for that category: no individuals with less than a high school education make more than \$250,000 per year. Hence we refit the GAM, excluding the individuals with less than a high school education. The resulting model is shown in Figure 7.14. As in Figures 7.11 and 7.12, all three panels have the same vertical scale. This allows us to visually assess the relative contributions of each of the variables. We observe that `age` and `education` have a much larger effect than `year` on the probability of being a high earner.

## 7.8 Lab: Non-linear Modeling

In this lab, we re-analyze the `Wage` data considered in the examples throughout this chapter, in order to illustrate the fact that many of the complex non-linear fitting procedures discussed can be easily implemented in `R`. We begin by loading the `ISLR` library, which contains the data.

```
> library(ISLR)
> attach(Wage)
```



**FIGURE 7.14.** The same model is fit as in Figure 7.13, this time excluding the observations for which **education** is **<HS**. Now we see that increased education tends to be associated with higher salaries.

### 7.8.1 Polynomial Regression and Step Functions

We now examine how Figure 7.1 was produced. We first fit the model using the following command:

```
> fit=lm(wage~poly(age,4),data=Wage)
> coef(summary(fit))
```

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	111.704	0.729	153.28	<2e-16
poly(age, 4)1	447.068	39.915	11.20	<2e-16
poly(age, 4)2	-478.316	39.915	-11.98	<2e-16
poly(age, 4)3	125.522	39.915	3.14	0.0017
poly(age, 4)4	-77.911	39.915	-1.95	0.0510

This syntax fits a linear model, using the `lm()` function, in order to predict **wage** using a fourth-degree polynomial in **age**: `poly(age,4)`. The `poly()` command allows us to avoid having to write out a long formula with powers of **age**. The function returns a matrix whose columns are a basis of *orthogonal polynomials*, which essentially means that each column is a linear combination of the variables **age**, **age**<sup>2</sup>, **age**<sup>3</sup> and **age**<sup>4</sup>.

orthogonal  
polynomial

However, we can also use `poly()` to obtain **age**, **age**<sup>2</sup>, **age**<sup>3</sup> and **age**<sup>4</sup> directly, if we prefer. We can do this by using the `raw=TRUE` argument to the `poly()` function. Later we see that this does not affect the model in a meaningful way—though the choice of basis clearly affects the coefficient estimates, it does not affect the fitted values obtained.

```
> fit2=lm(wage~poly(age,4,raw=T),data=Wage)
> coef(summary(fit2))
```

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	-1.84e+02	6.00e+01	-3.07	0.002180
poly(age, 4, raw = T)1	2.12e+01	5.89e+00	3.61	0.000312
poly(age, 4, raw = T)2	-5.64e-01	2.06e-01	-2.74	0.006261



```
poly(age, 4, raw = T)3 6.81e-03 3.07e-03 2.22 0.026398
poly(age, 4, raw = T)4 -3.20e-05 1.64e-05 -1.95 0.051039
```

There are several other equivalent ways of fitting this model, which showcase the flexibility of the formula language in R. For example

```
> fit2a=lm(wage~age+I(age^2)+I(age^3)+I(age^4),data=Wage)
> coef(fit2a)
(Intercept)      age      I(age^2)      I(age^3)      I(age^4)
-1.84e+02    2.12e+01   -5.64e-01    6.81e-03   -3.20e-05
```

This simply creates the polynomial basis functions on the fly, taking care to protect terms like `age^2` via the *wrapper* function `I()` (the `^` symbol has a special meaning in formulas). wrapper

```
> fit2b=lm(wage~cbind(age, age^2, age^3, age^4),data=Wage)
```

This does the same more compactly, using the `cbind()` function for building a matrix from a collection of vectors; any function call such as `cbind()` inside a formula also serves as a wrapper.

We now create a grid of values for `age` at which we want predictions, and then call the generic `predict()` function, specifying that we want standard errors as well.

```
> agelims=range(age)
> age.grid=seq(from=agelims[1],to=agelims[2])
> preds=predict(fit,newdata=list(age=age.grid),se=TRUE)
> se.bands=cbind(preds$fit+2*preds$se.fit,preds$fit-2*preds$se.
  fit)
```

Finally, we plot the data and add the fit from the degree-4 polynomial.

```
> par(mfrow=c(1,2),mar=c(4.5,4.5,1,1),oma=c(0,0,4,0))
> plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
> title("Degree-4 Polynomial",outer=T)
> lines(age.grid,preds$fit,lwd=2,col="blue")
> matlines(age.grid,se.bands,lwd=1,col="blue",lty=3)
```

Here the `mar` and `oma` arguments to `par()` allow us to control the margins of the plot, and the `title()` function creates a figure title that spans both subplots. title()

We mentioned earlier that whether or not an orthogonal set of basis functions is produced in the `poly()` function will not affect the model obtained in a meaningful way. What do we mean by this? The fitted values obtained in either case are identical:

```
> preds2=predict(fit2,newdata=list(age=age.grid),se=TRUE)
> max(abs(preds$fit-preds2$fit))
[1] 7.39e-13
```

In performing a polynomial regression we must decide on the degree of the polynomial to use. One way to do this is by using hypothesis tests. We now fit models ranging from linear to a degree-5 polynomial and seek to determine the simplest model which is sufficient to explain the relationship

between `wage` and `age`. We use the `anova()` function, which performs an *analysis of variance* (ANOVA, using an F-test) in order to test the null hypothesis that a model  $\mathcal{M}_1$  is sufficient to explain the data against the alternative hypothesis that a more complex model  $\mathcal{M}_2$  is required. In order to use the `anova()` function,  $\mathcal{M}_1$  and  $\mathcal{M}_2$  must be *nested* models: the predictors in  $\mathcal{M}_1$  must be a subset of the predictors in  $\mathcal{M}_2$ . In this case, we fit five different models and sequentially compare the simpler model to the more complex model.

`anova()`  
analysis of  
variance

```
> fit.1=lm(wage~age,data=Wage)
> fit.2=lm(wage~poly(age,2),data=Wage)
> fit.3=lm(wage~poly(age,3),data=Wage)
> fit.4=lm(wage~poly(age,4),data=Wage)
> fit.5=lm(wage~poly(age,5),data=Wage)
> anova(fit.1,fit.2,fit.3,fit.4,fit.5)
Analysis of Variance Table

Model 1: wage ~ age
Model 2: wage ~ poly(age, 2)
Model 3: wage ~ poly(age, 3)
Model 4: wage ~ poly(age, 4)
Model 5: wage ~ poly(age, 5)
  Res.Df    RSS Df Sum of Sq    F Pr(>F)
1     2998 5022216
2     2997 4793430   1    228786 143.59 <2e-16 ***
3     2996 4777674   1     15756   9.89 0.0017 **
4     2995 4771604   1      6070   3.81 0.0510 .
5     2994 4770322   1     1283   0.80 0.3697
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-value comparing the linear **Model 1** to the quadratic **Model 2** is essentially zero ( $<10^{-15}$ ), indicating that a linear fit is not sufficient. Similarly the p-value comparing the quadratic **Model 2** to the cubic **Model 3** is very low (0.0017), so the quadratic fit is also insufficient. The p-value comparing the cubic and degree-4 polynomials, **Model 3** and **Model 4**, is approximately 5% while the degree-5 polynomial **Model 5** seems unnecessary because its p-value is 0.37. Hence, either a cubic or a quartic polynomial appear to provide a reasonable fit to the data, but lower- or higher-order models are not justified.

In this case, instead of using the `anova()` function, we could have obtained these p-values more succinctly by exploiting the fact that `poly()` creates orthogonal polynomials.

```
> coef(summary(fit.5))
              Estimate Std. Error  t value Pr(>|t|)
(Intercept)    111.70     0.7288  153.2780 0.000e+00
poly(age, 5)1    447.07    39.9161  11.2002 1.491e-28
poly(age, 5)2   -478.32    39.9161 -11.9830 2.368e-32
poly(age, 5)3    125.52    39.9161   3.1446 1.679e-03
```

```
poly(age, 5) 4    -77.91    39.9161   -1.9519   5.105e-02
poly(age, 5) 5    -35.81    39.9161   -0.8972   3.697e-01
```

Notice that the p-values are the same, and in fact the square of the t-statistics are equal to the F-statistics from the `anova()` function; for example:

```
> (-11.983)^2
[1] 143.6
```

However, the ANOVA method works whether or not we used orthogonal polynomials; it also works when we have other terms in the model as well. For example, we can use `anova()` to compare these three models:

```
> fit.1=lm(wage~education+age,data=Wage)
> fit.2=lm(wage~education+poly(age,2),data=Wage)
> fit.3=lm(wage~education+poly(age,3),data=Wage)
> anova(fit.1,fit.2,fit.3)
```

As an alternative to using hypothesis tests and ANOVA, we could choose the polynomial degree using cross-validation, as discussed in Chapter 5.

Next we consider the task of predicting whether an individual earns more than \$250,000 per year. We proceed much as before, except that first we create the appropriate response vector, and then apply the `glm()` function using `family="binomial"` in order to fit a polynomial logistic regression model.

```
> fit=glm(I(wage>250)~poly(age,4),data=Wage,family=binomial)
```

Note that we again use the wrapper `I()` to create this binary response variable on the fly. The expression `wage>250` evaluates to a logical variable containing `TRUE`s and `FALSE`s, which `glm()` coerces to binary by setting the `TRUE`s to 1 and the `FALSE`s to 0.

Once again, we make predictions using the `predict()` function.

```
> preds=predict(fit,newdata=list(age=age.grid),se=T)
```

However, calculating the confidence intervals is slightly more involved than in the linear regression case. The default prediction type for a `glm()` model is `type="link"`, which is what we use here. This means we get predictions for the *logit*: that is, we have fit a model of the form

$$\log\left(\frac{\Pr(Y = 1|X)}{1 - \Pr(Y = 1|X)}\right) = X\beta,$$

and the predictions given are of the form  $X\hat{\beta}$ . The standard errors given are also of this form. In order to obtain confidence intervals for  $\Pr(Y = 1|X)$ , we use the transformation

$$\Pr(Y = 1|X) = \frac{\exp(X\beta)}{1 + \exp(X\beta)}.$$

```
> pfit=exp(preds$fit)/(1+exp(preds$fit))
> se.bands.logit = cbind(preds$fit+2*preds$se.fit, preds$fit-2*
  preds$se.fit)
> se.bands = exp(se.bands.logit)/(1+exp(se.bands.logit))
```

Note that we could have directly computed the probabilities by selecting the `type="response"` option in the `predict()` function.

```
> preds=predict(fit,newdata=list(age=age.grid),type="response",
  se=T)
```

However, the corresponding confidence intervals would not have been sensible because we would end up with negative probabilities!

Finally, the right-hand plot from Figure 7.1 was made as follows:

```
> plot(age,I(wage>250),xlim=agelims,type="n",ylim=c(0,.2))
> points(jitter(age), I((wage>250)/5),cex=.5,pch="|",
  col="darkgrey")
> lines(age.grid,pfit,lwd=2, col="blue")
> matlines(age.grid,se.bands,lwd=1,col="blue",lty=3)
```

We have drawn the `age` values corresponding to the observations with `wage` values above 250 as gray marks on the top of the plot, and those with `wage` values below 250 are shown as gray marks on the bottom of the plot. We used the `jitter()` function to jitter the `age` values a bit so that observations with the same `age` value do not cover each other up. This is often called a *rug plot*.

`jitter()`

rug plot

In order to fit a step function, as discussed in Section 7.2, we use the `cut()` function.

`cut()`

```
> table(cut(age,4))
(17.9,33.5] (33.5,49] (49,64.5] (64.5,80.1]
      750      1399      779      72
> fit=lm(wage~cut(age,4),data=Wage)
> coef(summary(fit))
```

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	94.16	1.48	63.79	0.00e+00
cut(age, 4) (33.5,49]	24.05	1.83	13.15	1.98e-38
cut(age, 4) (49,64.5]	23.66	2.07	11.44	1.04e-29
cut(age, 4) (64.5,80.1]	7.64	4.99	1.53	1.26e-01

Here `cut()` automatically picked the cutpoints at 33.5, 49, and 64.5 years of age. We could also have specified our own cutpoints directly using the `breaks` option. The function `cut()` returns an ordered categorical variable; the `lm()` function then creates a set of dummy variables for use in the regression. The `age<33.5` category is left out, so the intercept coefficient of \$94,160 can be interpreted as the average salary for those under 33.5 years of age, and the other coefficients can be interpreted as the average additional salary for those in the other age groups. We can produce predictions and plots just as we did in the case of the polynomial fit.

## 7.8.2 Splines

In order to fit regression splines in R, we use the `splines` library. In Section 7.4, we saw that regression splines can be fit by constructing an appropriate matrix of basis functions. The `bs()` function generates the entire matrix of basis functions for splines with the specified set of knots. By default, cubic splines are produced. Fitting `wage` to `age` using a regression spline is simple:

```
> library(splines)
> fit=lm(wage~bs(age,knots=c(25,40,60)),data=Wage)
> pred=predict(fit,newdata=list(age=age.grid),se=T)
> plot(age,wage,col="gray")
> lines(age.grid,pred$fit,lwd=2)
> lines(age.grid,pred$fit+2*pred$se,lty="dashed")
> lines(age.grid,pred$fit-2*pred$se,lty="dashed")
```

Here we have prespecified knots at ages 25, 40, and 60. This produces a spline with six basis functions. (Recall that a cubic spline with three knots has seven degrees of freedom; these degrees of freedom are used up by an intercept, plus six basis functions.) We could also use the `df` option to produce a spline with knots at uniform quantiles of the data.

```
> dim(bs(age,knots=c(25,40,60)))
[1] 3000 6
> dim(bs(age,df=6))
[1] 3000 6
> attr(bs(age,df=6),"knots")
 25%  50%  75%
33.8 42.0 51.0
```

In this case R chooses knots at ages 33.8, 42.0, and 51.0, which correspond to the 25th, 50th, and 75th percentiles of `age`. The function `bs()` also has a `degree` argument, so we can fit splines of any degree, rather than the default degree of 3 (which yields a cubic spline).

In order to instead fit a natural spline, we use the `ns()` function. Here we fit a natural spline with four degrees of freedom.

```
> fit2=lm(wage~ns(age,df=4),data=Wage)
> pred2=predict(fit2,newdata=list(age=age.grid),se=T)
> lines(age.grid,pred2$fit,col="red",lwd=2)
```

As with the `bs()` function, we could instead specify the knots directly using the `knots` option.

In order to fit a smoothing spline, we use the `smooth.spline()` function. Figure 7.8 was produced with the following code:

```
> plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
> title("Smoothing Spline")
> fit=smooth.spline(age,wage,df=16)
> fit2=smooth.spline(age,wage,cv=TRUE)
> fit2$df
[1] 6.8
> lines(fit,col="red",lwd=2)
```

`smooth.spline()`

```
> lines(fit2, col="blue", lwd=2)
> legend("topright", legend=c("16 DF", "6.8 DF"),
        col=c("red", "blue"), lty=1, lwd=2, cex=.8)
```

Notice that in the first call to `smooth.spline()`, we specified `df=16`. The function then determines which value of  $\lambda$  leads to 16 degrees of freedom. In the second call to `smooth.spline()`, we select the smoothness level by cross-validation; this results in a value of  $\lambda$  that yields 6.8 degrees of freedom.

In order to perform local regression, we use the `loess()` function.

`loess()`

```
> plot(age, wage, xlim=agelims, cex=.5, col="darkgrey")
> title("Local Regression")
> fit=loess(wage~age, span=.2, data=Wage)
> fit2=loess(wage~age, span=.5, data=Wage)
> lines(age.grid, predict(fit, data.frame(age=age.grid)),
        col="red", lwd=2)
> lines(age.grid, predict(fit2, data.frame(age=age.grid)),
        col="blue", lwd=2)
> legend("topright", legend=c("Span=0.2", "Span=0.5"),
        col=c("red", "blue"), lty=1, lwd=2, cex=.8)
```

Here we have performed local linear regression using spans of 0.2 and 0.5: that is, each neighborhood consists of 20% or 50% of the observations. The larger the span, the smoother the fit. The `locfit` library can also be used for fitting local regression models in R.

### 7.8.3 GAMs

We now fit a GAM to predict `wage` using natural spline functions of `year` and `age`, treating `education` as a qualitative predictor, as in (7.16). Since this is just a big linear regression model using an appropriate choice of basis functions, we can simply do this using the `lm()` function.

```
> gam1=lm(wage~ns(year,4)+ns(age,5)+education, data=Wage)
```

We now fit the model (7.16) using smoothing splines rather than natural splines. In order to fit more general sorts of GAMs, using smoothing splines or other components that cannot be expressed in terms of basis functions and then fit using least squares regression, we will need to use the `gam` library in R.

The `s()` function, which is part of the `gam` library, is used to indicate that we would like to use a smoothing spline. We specify that the function of `year` should have 4 degrees of freedom, and that the function of `age` will have 5 degrees of freedom. Since `education` is qualitative, we leave it as is, and it is converted into four dummy variables. We use the `gam()` function in order to fit a GAM using these components. All of the terms in (7.16) are fit simultaneously, taking each other into account to explain the response.

`s()`

`gam()`

```
> library(gam)
> gam.m3=gam(wage~s(year,4)+s(age,5)+education, data=Wage)
```

In order to produce Figure 7.12, we simply call the `plot()` function:

```
> par(mfrow=c(1,3))
> plot(gam.m3, se=TRUE, col="blue")
```

The generic `plot()` function recognizes that `gam.m3` is an object of class `gam`, and invokes the appropriate `plot.gam()` method. Conveniently, even though `gam1` is not of class `gam` but rather of class `lm`, we can *still* use `plot.gam()` on it. Figure 7.11 was produced using the following expression:

```
> plot.gam(gam1, se=TRUE, col="red")
```

Notice here we had to use `plot.gam()` rather than the *generic* `plot()` function.

In these plots, the function of `year` looks rather linear. We can perform a series of ANOVA tests in order to determine which of these three models is best: a GAM that excludes `year` ( $\mathcal{M}_1$ ), a GAM that uses a linear function of `year` ( $\mathcal{M}_2$ ), or a GAM that uses a spline function of `year` ( $\mathcal{M}_3$ ).

```
> gam.m1=gam(wage~s(age,5)+education, data=Wage)
> gam.m2=gam(wage~year+s(age,5)+education, data=Wage)
> anova(gam.m1, gam.m2, gam.m3, test="F")
Analysis of Deviance Table
```

```
Model 1: wage ~ s(age, 5) + education
Model 2: wage ~ year + s(age, 5) + education
Model 3: wage ~ s(year, 4) + s(age, 5) + education
  Resid. Df Resid. Dev Df Deviance    F    Pr(>F)
1      2990      3711730
2      2989      3693841    1    17889 14.5 0.00014 ***
3      2986      3689770    3     4071  1.1 0.34857
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We find that there is compelling evidence that a GAM with a linear function of `year` is better than a GAM that does not include `year` at all ( $p$ -value = 0.00014). However, there is no evidence that a non-linear function of `year` is needed ( $p$ -value = 0.349). In other words, based on the results of this ANOVA,  $\mathcal{M}_2$  is preferred.

The `summary()` function produces a summary of the gam fit.

```
> summary(gam.m3)

Call: gam(formula = wage ~ s(year, 4) + s(age, 5) + education,
  data = Wage)
Deviance Residuals:
   Min       1Q   Median       3Q      Max
-119.43  -19.70   -3.33   14.17  213.48

(Dispersion Parameter for gaussian family taken to be 1236)

Null Deviance: 5222086 on 2999 degrees of freedom
Residual Deviance: 3689770 on 2986 degrees of freedom
```

```
AIC: 29888
```

```
Number of Local Scoring Iterations: 2
```

```
DF for Terms and F-values for Nonparametric Effects
```

	Df	Npar	Df	Npar	F	Pr(F)
(Intercept)	1					
s(year, 4)	1		3		1.1	0.35
s(age, 5)	1		4		32.4	<2e-16 ***
education	4					

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The p-values for `year` and `age` correspond to a null hypothesis of a linear relationship versus the alternative of a non-linear relationship. The large p-value for `year` reinforces our conclusion from the ANOVA test that a linear function is adequate for this term. However, there is very clear evidence that a non-linear term is required for `age`.

We can make predictions from `gam` objects, just like from `lm` objects, using the `predict()` method for the class `gam`. Here we make predictions on the training set.

```
> preds=predict(gam.m2,newdata=Wage)
```

We can also use local regression fits as building blocks in a GAM, using the `lo()` function.

```
> gam.lo=gam(wage~s(year,df=4)+lo(age,span=0.7)+education,
             data=Wage)
> plot.gam(gam.lo, se=TRUE, col="green")
```

`lo()`

Here we have used local regression for the `age` term, with a span of 0.7. We can also use the `lo()` function to create interactions before calling the `gam()` function. For example,

```
> gam.lo.i=gam(wage~lo(year,age,span=0.5)+education,
              data=Wage)
```

fits a two-term model, in which the first term is an interaction between `year` and `age`, fit by a local regression surface. We can plot the resulting two-dimensional surface if we first install the `akima` package.

```
> library(akima)
> plot(gam.lo.i)
```

In order to fit a logistic regression GAM, we once again use the `I()` function in constructing the binary response variable, and set `family=binomial`.

```
> gam.lr=gam(I(wage>250)~year+s(age,df=5)+education,
            family=binomial,data=Wage)
> par(mfrow=c(1,3))
> plot(gam.lr,se=T,col="green")
```



It is easy to see that there are no high earners in the <HS category:

```
> table(education, I(wage>250))
```


education	FALSE	TRUE
1. < HS Grad	268	0
2. HS Grad	966	5
3. Some College	643	7
4. College Grad	663	22
5. Advanced Degree	381	45

Hence, we fit a logistic regression GAM using all but this category. This provides more sensible results.

```
> gam.lr.s=gam(I(wage>250)~year+s(age,df=5)+education,family=
  binomial,data=Wage,subset=(education!="1. < HS Grad"))
> plot(gam.lr.s,se=T,col="green")
```

## 7.9 Exercises

### Conceptual

1. It was mentioned in the chapter that a cubic regression spline with one knot at  $\xi$  can be obtained using a basis of the form  $x, x^2, x^3, (x - \xi)_+^3$ , where  $(x - \xi)_+^3 = (x - \xi)^3$  if  $x > \xi$  and equals 0 otherwise. We will now show that a function of the form 

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)_+^3$$

is indeed a cubic regression spline, regardless of the values of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

- (a) Find a cubic polynomial

$$f_1(x) = a_1 + b_1 x + c_1 x^2 + d_1 x^3$$

such that  $f(x) = f_1(x)$  for all  $x \leq \xi$ . Express  $a_1, b_1, c_1, d_1$  in terms of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ .

- (b) Find a cubic polynomial

$$f_2(x) = a_2 + b_2 x + c_2 x^2 + d_2 x^3$$

such that  $f(x) = f_2(x)$  for all  $x > \xi$ . Express  $a_2, b_2, c_2, d_2$  in terms of  $\beta_0, \beta_1, \beta_2, \beta_3, \beta_4$ . We have now established that  $f(x)$  is a piecewise polynomial.

- (c) Show that  $f_1(\xi) = f_2(\xi)$ . That is,  $f(x)$  is continuous at  $\xi$ .
- (d) Show that  $f_1'(\xi) = f_2'(\xi)$ . That is,  $f'(x)$  is continuous at  $\xi$ .

(e) Show that  $f_1''(\xi) = f_2''(\xi)$ . That is,  $f''(x)$  is continuous at  $\xi$ .

Therefore,  $f(x)$  is indeed a cubic spline.

*Hint: Parts (d) and (e) of this problem require knowledge of single-variable calculus. As a reminder, given a cubic polynomial*

$$f_1(x) = a_1 + b_1x + c_1x^2 + d_1x^3,$$

the first derivative takes the form

$$f_1'(x) = b_1 + 2c_1x + 3d_1x^2$$

and the second derivative takes the form

$$f_1''(x) = 2c_1 + 6d_1x.$$

2. Suppose that a curve  $\hat{g}$  is computed to smoothly fit a set of  $n$  points using the following formula:

$$\hat{g} = \arg \min_g \left( \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int [g^{(m)}(x)]^2 dx \right),$$

where  $g^{(m)}$  represents the  $m$ th derivative of  $g$  (and  $g^{(0)} = g$ ). Provide example sketches of  $\hat{g}$  in each of the following scenarios.

- (a)  $\lambda = \infty, m = 0$ .
  - (b)  $\lambda = \infty, m = 1$ .
  - (c)  $\lambda = \infty, m = 2$ .
  - (d)  $\lambda = \infty, m = 3$ .
  - (e)  $\lambda = 0, m = 3$ .
3. Suppose we fit a curve with basis functions  $b_1(X) = X$ ,  $b_2(X) = (X - 1)^2 I(X \geq 1)$ . (Note that  $I(X \geq 1)$  equals 1 for  $X \geq 1$  and 0 otherwise.) We fit the linear regression model

$$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon,$$

and obtain coefficient estimates  $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1, \hat{\beta}_2 = -2$ . Sketch the estimated curve between  $X = -2$  and  $X = 2$ . Note the intercepts, slopes, and other relevant information.

4. Suppose we fit a curve with basis functions  $b_1(X) = I(0 \leq X \leq 2) - (X - 1)I(1 \leq X \leq 2)$ ,  $b_2(X) = (X - 3)I(3 \leq X \leq 4) + I(4 < X \leq 5)$ . We fit the linear regression model

$$Y = \beta_0 + \beta_1 b_1(X) + \beta_2 b_2(X) + \epsilon,$$

and obtain coefficient estimates  $\hat{\beta}_0 = 1, \hat{\beta}_1 = 1, \hat{\beta}_2 = 3$ . Sketch the estimated curve between  $X = -2$  and  $X = 2$ . Note the intercepts, slopes, and other relevant information.

5. Consider two curves,  $\hat{g}_1$  and  $\hat{g}_2$ , defined by

$$\hat{g}_1 = \arg \min_g \left( \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int [g^{(3)}(x)]^2 dx \right),$$

$$\hat{g}_2 = \arg \min_g \left( \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int [g^{(4)}(x)]^2 dx \right),$$

where  $g^{(m)}$  represents the  $m$ th derivative of  $g$ .

- As  $\lambda \rightarrow \infty$ , will  $\hat{g}_1$  or  $\hat{g}_2$  have the smaller training RSS?
- As  $\lambda \rightarrow \infty$ , will  $\hat{g}_1$  or  $\hat{g}_2$  have the smaller test RSS?
- For  $\lambda = 0$ , will  $\hat{g}_1$  or  $\hat{g}_2$  have the smaller training and test RSS?

### Applied

- In this exercise, you will further analyze the `Wage` data set considered throughout this chapter.
  - Perform polynomial regression to predict `wage` using `age`. Use cross-validation to select the optimal degree  $d$  for the polynomial. What degree was chosen, and how does this compare to the results of hypothesis testing using ANOVA? Make a plot of the resulting polynomial fit to the data.
  - Fit a step function to predict `wage` using `age`, and perform cross-validation to choose the optimal number of cuts. Make a plot of the fit obtained.
- The `Wage` data set contains a number of other features not explored in this chapter, such as marital status (`maritl`), job class (`jobclass`), and others. Explore the relationships between some of these other predictors and `wage`, and use non-linear fitting techniques in order to fit flexible models to the data. Create plots of the results obtained, and write a summary of your findings.
- Fit some of the non-linear models investigated in this chapter to the `Auto` data set. Is there evidence for non-linear relationships in this data set? Create some informative plots to justify your answer.
- This question uses the variables `dis` (the weighted mean of distances to five Boston employment centers) and `nox` (nitrogen oxides concentration in parts per 10 million) from the `Boston` data. We will treat `dis` as the predictor and `nox` as the response.
  - Use the `poly()` function to fit a cubic polynomial regression to predict `nox` using `dis`. Report the regression output, and plot the resulting data and polynomial fits.

- (b) Plot the polynomial fits for a range of different polynomial degrees (say, from 1 to 10), and report the associated residual sum of squares.
  - (c) Perform cross-validation or another approach to select the optimal degree for the polynomial, and explain your results.
  - (d) Use the `bs()` function to fit a regression spline to predict `nox` using `dis`. Report the output for the fit using four degrees of freedom. How did you choose the knots? Plot the resulting fit.
  - (e) Now fit a regression spline for a range of degrees of freedom, and plot the resulting fits and report the resulting RSS. Describe the results obtained.
  - (f) Perform cross-validation or another approach in order to select the best degrees of freedom for a regression spline on this data. Describe your results.
10. This question relates to the `College` data set.
- (a) Split the data into a training set and a test set. Using out-of-state tuition as the response and the other variables as the predictors, perform forward stepwise selection on the training set in order to identify a satisfactory model that uses just a subset of the predictors.
  - (b) Fit a GAM on the training data, using out-of-state tuition as the response and the features selected in the previous step as the predictors. Plot the results, and explain your findings.
  - (c) Evaluate the model obtained on the test set, and explain the results obtained.
  - (d) For which variables, if any, is there evidence of a non-linear relationship with the response?
11. In Section 7.7, it was mentioned that GAMs are generally fit using a *backfitting* approach. The idea behind backfitting is actually quite simple. We will now explore backfitting in the context of multiple linear regression.

Suppose that we would like to perform multiple linear regression, but we do not have software to do so. Instead, we only have software to perform simple linear regression. Therefore, we take the following iterative approach: we repeatedly hold all but one coefficient estimate fixed at its current value, and update only that coefficient estimate using a simple linear regression. The process is continued until *convergence*—that is, until the coefficient estimates stop changing.

We now try this out on a toy example.

- (a) Generate a response  $Y$  and two predictors  $X_1$  and  $X_2$ , with  $n = 100$ .
- (b) Initialize  $\hat{\beta}_1$  to take on a value of your choice. It does not matter what value you choose.
- (c) Keeping  $\hat{\beta}_1$  fixed, fit the model

$$Y - \hat{\beta}_1 X_1 = \beta_0 + \beta_2 X_2 + \epsilon.$$

You can do this as follows:

```
> a=y-beta1*x1
> beta2=lm(a~x2)$coef[2]
```

- (d) Keeping  $\hat{\beta}_2$  fixed, fit the model

$$Y - \hat{\beta}_2 X_2 = \beta_0 + \beta_1 X_1 + \epsilon.$$

You can do this as follows:

```
> a=y-beta2*x2
> beta1=lm(a~x1)$coef[2]
```

- (e) Write a for loop to repeat (c) and (d) 1,000 times. Report the estimates of  $\hat{\beta}_0$ ,  $\hat{\beta}_1$ , and  $\hat{\beta}_2$  at each iteration of the for loop. Create a plot in which each of these values is displayed, with  $\hat{\beta}_0$ ,  $\hat{\beta}_1$ , and  $\hat{\beta}_2$  each shown in a different color.
- (f) Compare your answer in (e) to the results of simply performing multiple linear regression to predict  $Y$  using  $X_1$  and  $X_2$ . Use the `abline()` function to overlay those multiple linear regression coefficient estimates on the plot obtained in (e).
- (g) On this data set, how many backfitting iterations were required in order to obtain a “good” approximation to the multiple regression coefficient estimates?
12. This problem is a continuation of the previous exercise. In a toy example with  $p = 100$ , show that one can approximate the multiple linear regression coefficient estimates by repeatedly performing simple linear regression in a backfitting procedure. How many backfitting iterations are required in order to obtain a “good” approximation to the multiple regression coefficient estimates? Create a plot to justify your answer.