Chapter 4

Introducing GAMs

4.1 Introduction

A generalized additive model (Hastie and Tibshirani, 1986, 1990) is a generalized linear model with a linear predictor involving a sum of smooth functions of covariates. In general the model has a structure something like

$$g(\mu_i) = \mathbf{A}_i\boldsymbol{\theta} + f_1(x_{1i}) + f_2(x_{2i}) + f_3(x_{3i}, x_{4i}) + \ldots \tag{4.1}$$

where $\mu_i \equiv \mathbb{E}(Y_i)$ and $Y_i \sim \text{EF}(\mu_i, \phi)$. Y_i is a response variable, $\text{EF}(\mu_i, \phi)$ denotes an exponential family distribution with mean μ_i and scale parameter, ϕ, $\mathbf{A}_i$ is a row of the model matrix for any strictly parametric model components, $\boldsymbol{\theta}$ is the corresponding parameter vector, and the f_j are smooth functions of the covariates, x_k. The model allows for flexible specification of the dependence of the response on the covariates, but by specifying the model only in terms of 'smooth functions', rather than detailed parametric relationships, it is possible to avoid the sort of cumbersome and unwieldy models seen in section 3.3.5, for example. This flexibility and convenience comes at the cost of two new theoretical problems. It is necessary both to represent the smooth functions in some way and to choose how smooth they should be.

This chapter illustrates how GAMs can be represented using basis expansions for each smooth, each with an associated penalty controlling function smoothness. Estimation can then be carried out by penalized regression methods, and the appropriate degree of smoothness for the f_j can be estimated from data using cross validation or marginal likelihood maximization. To avoid obscuring the basic simplicity of the approach with a mass of technical detail, the most complicated model considered here will be a simple GAM with two univariate smooth components. Furthermore, the methods presented will not be those that are most suitable for general practical use, being rather the methods that enable the basic framework to be explained simply. The ideal way to read this chapter is sitting at a computer, working through the statistics, and its implementation in R, side by side. If adopting this approach recall that the help files for R functions can be accessed by typing `?` followed by the function name, at the command line (e.g., `?lm`, for help on the linear modelling function).

4.2 Univariate smoothing

The representation and estimation of component functions of a model is best introduced by considering a model containing one function of one covariate,

$$y_i = f(x_i) + \epsilon_i, \tag{4.2}$$

where y_i is a response variable, x_i a covariate, f a smooth function and the ϵ_i are independent $N(0, \sigma^2)$ random variables.

4.2.1 Representing a function with basis expansions

To estimate f, using the methods covered in chapters 1 to 3, requires that f be represented in such a way that (4.2) becomes a linear model. This can be done by choosing a *basis*, defining the space of functions of which f (or a close approximation to it) is an element. Choosing a basis amounts to choosing some *basis functions*, which will be treated as completely known: if $b_j(x)$ is the j^{th} such basis function, then f is assumed to have a representation

$$f(x) = \sum_{j=1}^{k} b_j(x)\beta_j, \tag{4.3}$$

for some values of the unknown parameters, β_j. Substituting (4.3) into (4.2) clearly yields a linear model.

A very simple basis: Polynomials

As a simple example, suppose that f is believed to be a 4^{th} order polynomial, so that the space of polynomials of order 4 and below contains f. A basis for this space is $b_1(x) = 1$, $b_2(x) = x$, $b_3(x) = x^2$, $b_4(x) = x^3$ and $b_5(x) = x^4$, so that (4.3) becomes

$$f(x) = \beta_1 + x\beta_2 + x^2\beta_3 + x^3\beta_4 + x^4\beta_5,$$

and (4.2) becomes the simple model

$$y_i = \beta_1 + x_i\beta_2 + x_i^2\beta_3 + x_i^3\beta_4 + x_i^4\beta_5 + \epsilon_i.$$

Figures 4.1 and 4.2 illustrate a basis function representation of a function, f, using a polynomial basis.

The problem with polynomials

Taylor's theorem implies that polynomial bases will be useful for situations in which interest focuses on properties of f in the vicinity of a single specified point. But when the questions of interest relate to f over its whole domain, polynomial bases are problematic (see exercise 1).

The difficulties are most easily illustrated in the context of interpolation. The middle panel of figure 4.3 illustrates an attempt to approximate the function shown in

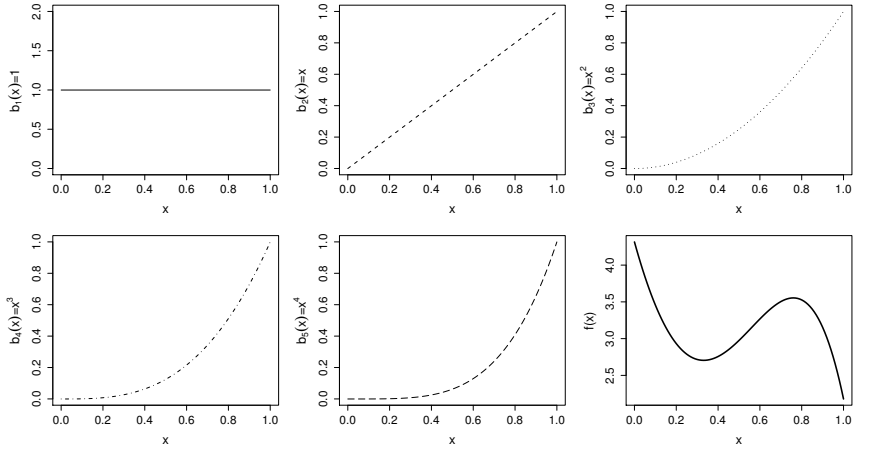

Figure 4.1 *Illustration of the idea of representing a function in terms of basis functions, using a polynomial basis. The first 5 panels (starting from top left) illustrate the 5 basis functions, $b_j(x)$, for a 4th order polynomial basis. The basis functions are each multiplied by a real valued parameter, β_j, and are then summed to give the final curve $f(x)$, an example of which is shown in the bottom right panel. By varying the β_j, we can vary the form of $f(x)$, to produce any polynomial function of order 4 or lower. See also figure 4.2*

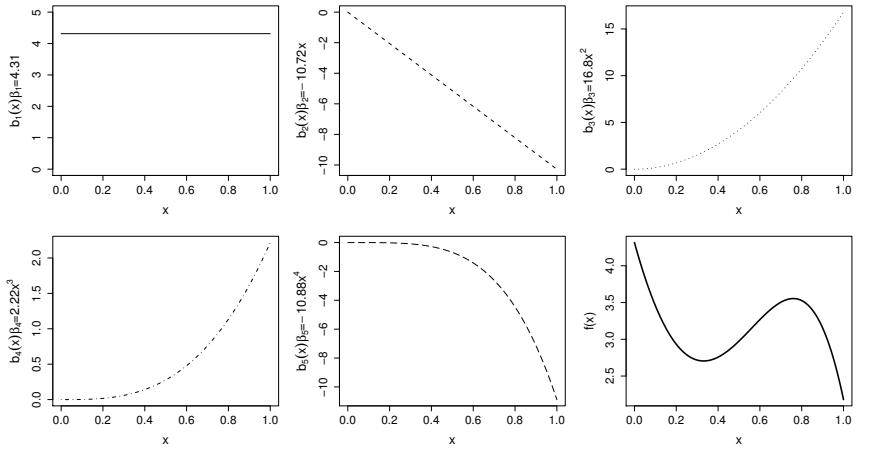

Figure 4.2 *An alternative illustration of how a function is represented in terms of basis functions. As in figure 4.1, a 4th order polynomial basis is illustrated. In this case the 5 basis functions, $b_j(x)$, each multiplied by its coefficient β_j, are shown in the first five figures (starting at top left). Simply summing these 5 curves yields the function, $f(x)$, shown at bottom right.*

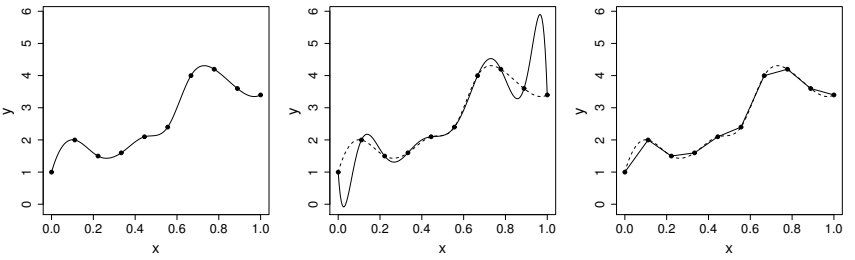

Figure 4.3 *The left panel shows a smooth function sampled at the points shown as black dots. The middle panel shows an attempt to reconstruct the function (dashed curve) by polynomial interpolation (black curve) of the black dots. The right panel shows the equivalent piecewise linear interpolant. The condition that the polynomial should interpolate the data and have all its derivatives continuous leads to quite wild oscillation.*

the left panel of figure 4.3, by polynomial interpolation of the points shown as black dots. The polynomial oscillates wildly in places, in order to accommodate the twin requirements to interpolate the data *and* to have all derivatives w.r.t. x continuous. If we relax the requirement for continuity of derivatives, and simply use a piecewise linear interpolant, then a much better approximation is obtained, as the right hand panel of figure 4.3 illustrates.

It clearly makes sense to use bases that are good at approximating known functions in order to represent unknown functions. Similarly, bases that perform well for interpolating exact observations of a function are also a good starting point for the closely related task of smoothing noisy observations of a function. In subsequent chapters we will see that piecewise linear bases can be improved upon by spline bases having continuity of just a few derivatives, but the piecewise linear case provides such a convenient illustration that we will stick with it for this chapter.

The piecewise linear basis

A basis for piecewise linear functions of a univariate variable x is determined entirely by the locations of the function's derivative discontinuities, that is by the locations at which the linear pieces join up. Let these *knots* be denoted $\{x_j^* : j = 1, \cdots k\}$, and suppose that $x_j^* > x_{j-1}^*$. Then for $j = 2, \ldots, k-1$

$$b_j(x) = \begin{cases} (x - x_{j-1}^*)/(x_j^* - x_{j-1}^*) & x_{j-1}^* < x \leq x_j^* \\ (x_{i+j}^* - x)/(x_{j+1}^* - x_j^*) & x_j^* < x < x_{j+1}^* \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

while

$$b_1(x) = \begin{cases} (x_2^* - x)/(x_2^* - x_1^*) & x < x_2^* \\ 0 & \text{otherwise} \end{cases}$$

and

$$b_k(x) = \begin{cases} (x - x_{k-1}^*)/(x_k^* - x_{k-1}^*) & x > x_{k-1}^* \\ 0 & \text{otherwise} \end{cases}$$

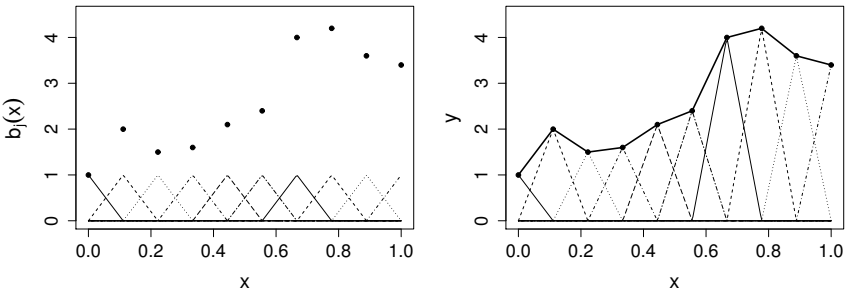

Figure 4.4 *The left panel shows an example tent function basis for interpolating the data shown as black dots. The continuous lines show the tent function basis functions, each of which peaks with value 1 at the x-axis value of one of the data points. The right panel illustrates how the basis functions are each multiplied by a coefficient, before being summed to give the interpolant, shown as the thick black line.*

So $b_j(x)$ is zero everywhere, except over the interval between the knots immediately to either side of x_j^*. $b_j(x)$ increases linearly from 0 at x_{j-1}^* to 1 at x_j^*, and then decreases linearly to 0 at x_{j+1}^*. Basis functions like this, that are non zero only over some finite intervals, are said to have *compact support*. Because of their shape the b_j are often known as *tent functions*. See figure 4.4.

Notice that an exactly equivalent way of defining $b_j(x)$ is as the linear interpolant of the data $\{x_i^*, \delta_i^j : i = 1, \ldots, k\}$ where $\delta_i^j = 1$ if $i = j$ and zero otherwise. This definition makes for very easy coding of the basis in R.

Using this basis to represent $f(x)$, (4.2) now becomes the linear model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$ where $X_{ij} = b_j(x_i)$.

Using the piecewise linear basis

Now consider an illustrative example. It is often claimed, at least by people with little actual knowledge of engines, that a car engine with a larger cylinder capacity will wear out less quickly than a smaller capacity engine. Figure 4.5 shows some data for 19 Volvo engines. The pattern of variation is not entirely clear, so (4.2) might be an appropriate model.

First read the data into R.

```
require(gamair); data(engine); attach(engine)
plot(size,wear,xlab="Engine capacity",ylab="Wear index")
```

Now write an R function defining $b_j(x)$

```
tf <- function(x,xj,j) {
## generate jth tent function from set defined by knots xj
  dj <- xj*0;dj[j] <- 1
  approx(xj,dj,x)$y
}
```

and use it to write an R function that will take a sequence of knots and an array of x values to produce a model matrix for the piecewise linear function.

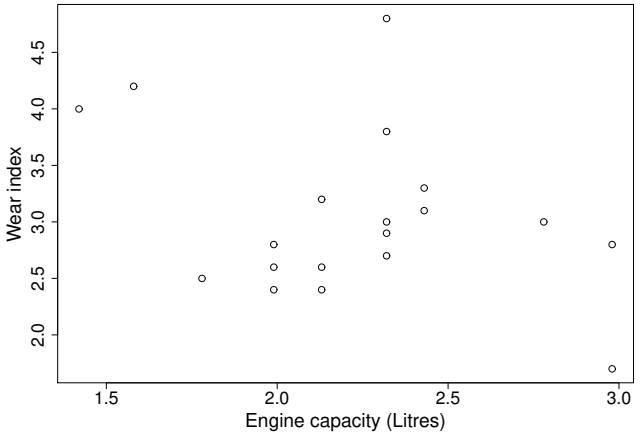

Figure 4.5 *Data on engine wear index versus engine capacity for 19 Volvo car engines, obtained from* `http://www3.bc.sympatico.ca/Volvo_Books/engine3.html`.

```
tf.X <- function(x,xj) {
## tent function basis matrix given data x
## and knot sequence xj
  nk <- length(xj); n <- length(x)
  X <- matrix(NA,n,nk)
  for (j in 1:nk) X[,j] <- tf(x,xj,j)
  X
}
```

All that is required now is to select a set of knots, x_j^*, and the model can be fitted. In the following a rank $k = 6$ basis is used, with the knots spread evenly over the range of the `size` data.

```
sj <- seq(min(size),max(size),length=6) ## generate knots
X <- tf.X(size,sj)                      ## get model matrix
b <- lm(wear ~ X - 1)                   ## fit model
s <- seq(min(size),max(size),length=200)## prediction data
Xp <- tf.X(s,sj)                        ## prediction matrix
plot(size,wear)                         ## plot data
lines(s,Xp %*% coef(b))                 ## overlay estimated f
```

The model fit looks quite plausible (figure 4.6), but the choice of degree of model smoothness, controlled here by the basis dimension, k, was essentially arbitrary. This issue must be addressed if a satisfactory theory for modelling with unknown functions is to be developed.

4.2.2 Controlling smoothness by penalizing wiggliness

One obvious possibility for choosing the degree of smoothing is to try to make use of the hypothesis testing methods from chapter 1, to select k by backwards selection.

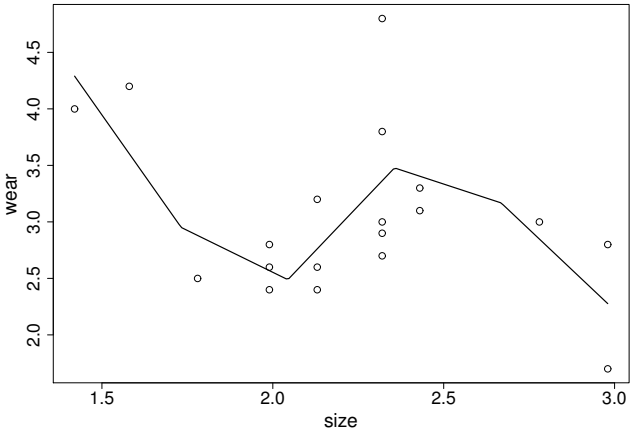

Figure 4.6 *Piecewise linear regression fit (continuous line) to data (○) on engine wear index versus engine capacity (size) for 19 Volvo car engines.*

However, such an approach is problematic, since a model based on $k-1$ evenly spaced knots will not generally be nested within a model based on k evenly spaced knots. It is possible to start with a fine grid of knots and simply drop knots sequentially, as part of backward selection, but the resulting uneven knot spacing can itself lead to poor model performance. Furthermore, the fit of such regression models tends to depend quite strongly on the locations chosen for the knots.

An alternative is to keep the basis dimension fixed at a size a little larger than it is believed could reasonably be necessary, but to control the model's smoothness by adding a 'wiggliness' penalty to the least squares fitting objective. For example, rather than fitting the model by minimizing

$$\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|^2,$$

it could be fitted by minimizing

$$\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|^2+\lambda\sum_{j=2}^{k-1}\{f(x_{j-1}^*)-2f(x_j^*)+f(x_{j+1}^*)\}^2,$$

where the summation term measures wiggliness as a sum of squared second differences of the function at the knots (which crudely approximates the integrated squared second derivative penalty used in cubic spline smoothing: see section 5.1.2, p. 198). When f is very wiggly the penalty will take high values and when f is 'smooth' the penalty will be low.* If f is a straight line then the penalty is actually zero. So the penalty has a *null space* of functions that are un-penalized: the straight lines in this

*Note that even knot spacing has been assumed: uneven knot spacing would usually require some re-weighting of the penalty terms.

case. The dimension of the penalty null space is 2, since the basis for straight lines is 2-dimensional.

The *smoothing parameter*, λ, controls the trade-off between smoothness of the estimated f and fidelity to the data. $\lambda \to \infty$ leads to a straight line estimate for f, while $\lambda = 0$ results in an un-penalized piecewise linear regression estimate.

For the basis of tent functions, it is easy to see that the coefficients of f are simply the function values at the knots, i.e., $\beta_j = f(x_j^*)$. This makes it particularly straightforward to express the penalty as a quadratic form, $\boldsymbol{\beta}^{\mathsf{T}}\mathbf{S}\boldsymbol{\beta}$, in the basis coefficients (although in fact linearity of f in the basis coefficients is all that is required for this). Firstly note that

$$\begin{bmatrix} \beta_1 - 2\beta_2 + \beta_3 \\ \beta_2 - 2\beta_3 + \beta_4 \\ \beta_3 - 2\beta_4 + \beta_5 \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} 1 & -2 & 1 & 0 & . & . & . \\ 0 & 1 & -2 & 1 & 0 & . & . \\ 0 & 0 & 1 & -2 & 1 & 0 & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ . \\ . \end{bmatrix}$$

so that writing the right hand side as $\mathbf{D}\boldsymbol{\beta}$, by definition of $(k-2) \times k$ matrix $\mathbf{D}$, the penalty becomes

$$\sum_{j=2}^{k-1} (\beta_{j-1} - 2\beta_j + \beta_{j+1})^2 = \boldsymbol{\beta}^{\mathsf{T}}\mathbf{D}^{\mathsf{T}}\mathbf{D}\boldsymbol{\beta} = \boldsymbol{\beta}^{\mathsf{T}}\mathbf{S}\boldsymbol{\beta} \tag{4.5}$$

where $\mathbf{S} = \mathbf{D}^{\mathsf{T}}\mathbf{D}$ ($\mathbf{S}$ is obviously rank deficient by the dimension of the penalty null space). Hence the penalized regression fitting problem is to minimize

$$\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\boldsymbol{\beta}^{\mathsf{T}}\mathbf{S}\boldsymbol{\beta} \tag{4.6}$$

w.r.t. $\boldsymbol{\beta}$. The problem of estimating the degree of smoothness for the model is now the problem of estimating the smoothing parameter λ. But before addressing λ estimation, consider $\boldsymbol{\beta}$ estimation given λ.

It is fairly straightforward to show (see exercise 3) that the formal expression for the minimizer of (4.6), the penalized least squares estimator of $\boldsymbol{\beta}$, is

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{S}\right)^{-1}\mathbf{X}^{\mathsf{T}}\mathbf{y}. \tag{4.7}$$

Similarly the influence (hat) matrix, $\mathbf{A}$, for the model can be written

$$\mathbf{A} = \mathbf{X}\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda\mathbf{S}\right)^{-1}\mathbf{X}^{\mathsf{T}}.$$

Recall that $\hat{\boldsymbol{\mu}} = \mathbf{A}\mathbf{y}$. As with the un-penalized linear model, these expressions are not the ones to use for computation, for which the greater numerical stability offered by orthogonal matrix methods is to be preferred. For practical computation, therefore, note that

$$\left\| \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{X} \\ \sqrt{\lambda}\mathbf{D} \end{bmatrix} \boldsymbol{\beta} \right\|^2 = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda\boldsymbol{\beta}^{\mathsf{T}}\mathbf{S}\boldsymbol{\beta}.$$

Obviously any matrix square root such that $\mathbf{D}^{\mathsf{T}}\mathbf{D} = \mathbf{S}$ could be substituted for the original $\mathbf{D}$ here, but at the moment there is no reason to use an alternative. The sum of squares term, on the left hand side, is just a least squares objective for a model in which the model matrix has been augmented by a square root of the penalty matrix, while the response data vector has been augmented with $k - 2$ zeros. Fitting the augmented linear model will therefore yield $\hat{\boldsymbol{\beta}}$.

To see a penalized regression spline in action, first note that $\mathbf{D}$ can be obtained in R using `diff(diag(k),differences=2)`, which applies second order differencing to each column of the rank k identity matrix. Now it is easy to write a simple function for fitting a penalized piecewise linear smoother.

```
prs.fit <- function(y,x,xj,sp) {
  X <- tf.X(x,xj)       ## model matrix
  D <- diff(diag(length(xj)),differences=2) ## sqrt penalty
  X <- rbind(X,sqrt(sp)*D) ## augmented model matrix
  y <- c(y,rep(0,nrow(D))) ## augmented data
  lm(y ~ X - 1)    ## penalized least squares fit
}
```

To use this function, we need to choose the basis dimension, k, the (evenly spaced) knot locations, x_j^*, and a value for the smoothing parameter, λ. Provided that k is large enough that the basis is more flexible than we expect to *need* to represent $f(x)$, then neither the exact choice of k, nor the precise selection of knot locations, has a great deal of influence on the model fit. Rather it is the choice of λ that now plays the crucial role in determining model flexibility, and ultimately the shape of $\hat{f}(x)$. In the following example $k = 20$ and the knots are evenly spread out over the range of observed engine sizes. It is the smoothing parameter, $\lambda = 2$, which really controls the behaviour of the fitted model.

```
sj <- seq(min(size),max(size),length=20) ## knots
b <- prs.fit(wear,size,sj,2) ## penalized fit
plot(size,wear)   ## plot data
Xp <- tf.X(s,sj)  ## prediction matrix
lines(s,Xp %*% coef(b)) ## plot the smooth
```

By changing the value of the smoothing parameter, λ, a variety of models of different smoothness can be obtained. Figure 4.7 illustrates this, but begs the question, which value of λ is 'best'?

4.2.3 Choosing the smoothing parameter, λ, by cross validation

If λ is too high then the data will be over-smoothed, and if it is too low then the data will be under-smoothed: in both cases this will mean that the estimate $\hat{f}$ will not be close to the true function f. Ideally, it would be good to choose λ so that $\hat{f}$ is as close as possible to f. A suitable criterion might be to choose λ to minimize

$$M = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i - f_i)^2,$$

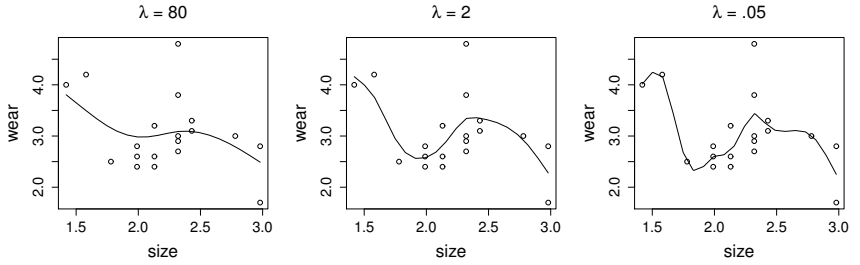

Figure 4.7 *Penalized piecewise linear fits to the engine wear versus capacity data, using three different values for the smoothing parameter, λ. Notice how penalization produces quite smooth estimates, despite the piecewise linear basis.*

where the notation $\hat{f}_i \equiv \hat{f}(x_i)$ and $f_i \equiv f(x_i)$ has been adopted for conciseness. Since f is unknown, M cannot be used directly, but it is possible to derive an estimate of $\mathbb{E}(M) + \sigma^2$, which is the expected squared error in predicting a new variable. Let $\hat{f}^{[-i]}$ be the model fitted to all data except y_i, and define the *ordinary cross validation* score

$$\mathcal{V}_o = \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i^{[-i]} - y_i)^2.$$

This score results from leaving out each datum in turn, fitting the model to the remaining data and calculating the squared difference between the missing datum and its predicted value: these squared differences are then averaged over all the data. Substituting $y_i = f_i + \epsilon_i$,

$$\begin{aligned}\mathcal{V}_o &= \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i^{[-i]} - f_i - \epsilon_i)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(\hat{f}_i^{[-i]} - f_i)^2 - 2(\hat{f}_i^{[-i]} - f_i)\epsilon_i + \epsilon_i^2.\end{aligned}$$

Since $\mathbb{E}(\epsilon_i) = 0$, and ϵ_i and $\hat{f}_i^{[-i]}$ are independent, the second term in the summation vanishes if expectations are taken:

$$\mathbb{E}(\mathcal{V}_o) = \frac{1}{n}\mathbb{E}\left(\sum_{i=1}^{n}(\hat{f}_i^{[-i]} - f_i)^2\right) + \sigma^2.$$

Now, $\hat{f}^{[-i]} \approx \hat{f}$ with equality in the large sample limit, so $\mathbb{E}(\mathcal{V}_o) \approx \mathbb{E}(M) + \sigma^2$ also with equality in the large sample limit. Hence choosing λ in order to minimize $\mathcal{V}_o$ is a reasonable approach if the ideal would be to minimize M. Choosing λ to minimize $\mathcal{V}_o$ is known as ordinary cross validation.

Ordinary cross validation is a reasonable approach, in its own right, even without a mean square (prediction) error justification. If models are judged only by their ability to fit the data from which they were estimated, then complicated models are

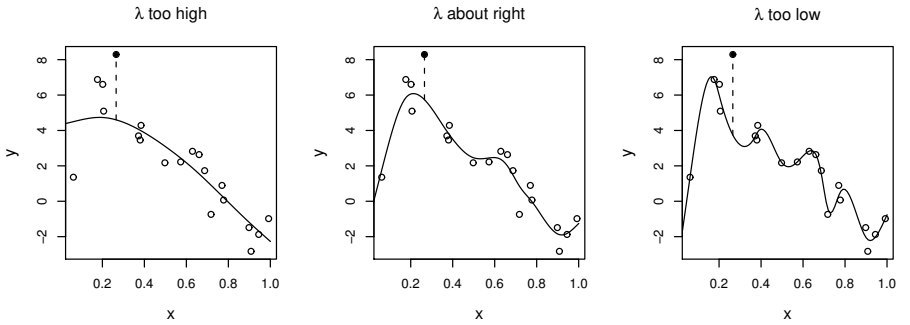

Figure 4.8 *Illustration of the principle behind cross validation. The fifth datum (•) has been omitted from fitting and the continuous curve shows a penalized regression spline fitted to the remaining data (○). When the smoothing parameter is too high the spline fits many of the data poorly and does no better with the missing point. When λ is too low the spline fits the noise as well as the signal and the consequent extra variability causes it to predict the missing datum poorly. For intermediate λ the spline is fitting the underlying signal quite well, but smoothing through the noise: hence, the missing datum is reasonably well predicted. Cross validation leaves out each datum from the data in turn and considers the average ability of models fitted to the remaining data to predict the omitted data.*

always selected over simpler ones. Choosing a model in order to maximize the ability to predict data to which the model was not fitted, does not suffer from this problem, as figure 4.8 illustrates.

It is computationally costly to calculate $\mathcal{V}_o$ by leaving out one datum at a time, refitting the model to each of the n resulting data sets, but it can be shown that

$$\mathcal{V}_o = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}_i)^2/(1 - A_{ii})^2,$$

where $\hat{f}$ is the estimate from fitting to all the data, and $\mathbf{A}$ is the corresponding influence matrix (see section 6.2.2, p. 256). In practice the A_{ii} are often replaced by their mean, $\mathrm{tr}\,(\mathbf{A})/n$, resulting in the *generalized cross validation* score

$$\mathcal{V}_g = \frac{n\sum_{i=1}^{n}(y_i - \hat{f}_i)^2}{[n - \mathrm{tr}\,(\mathbf{A})]^2}.$$

GCV has computational advantages over OCV, and it also has advantages in terms of invariance (see Wahba, 1990, p.53 or sections 6.2.2 and 6.2.3, p. 258). In any case, it can also be shown to minimize $\mathbb{E}(M)$ in the large sample limit.

Returning to the engine wear example, a simple direct search for the GCV optimal smoothing parameter can be made as follows.

```
rho = seq(-9,11,length=90)
n <- length(wear)
V <- rep(NA,90)
```

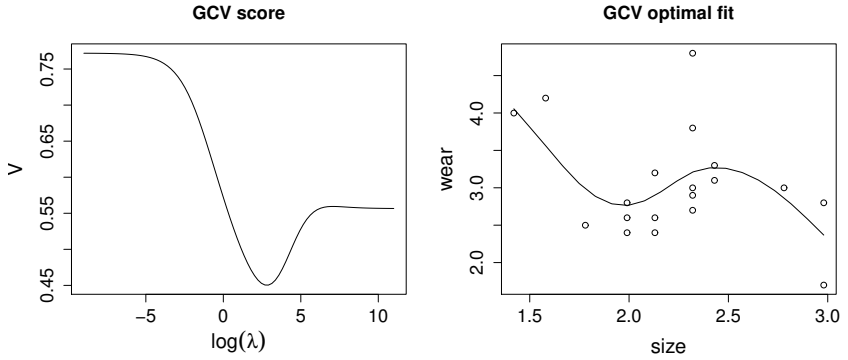

Figure 4.9 *Left panel: the GCV function for the engine wear example against log smoothing parameter. Right panel: the fitted model which minimizes the GCV score.*

```
for (i in 1:90) { ## loop through smoothing params
  b <- prs.fit(wear,size,sj,exp(rho[i])) ## fit model
  trF <- sum(influence(b)$hat[1:n])      ## extract EDF
  rss <- sum((wear-fitted(b)[1:n])^2)    ## residual SS
  V[i] <- n*rss/(n-trF)^2                ## GCV score
}
```

Note that the `influence()` function returns a list of diagnostics including `hat`, an array of the elements on the leading diagonal of the influence/hat matrix of the augmented model. The first n of these are the leading diagonal of the influence matrix of the penalized model (see exercise 4).

For the example, `V[54]` is the lowest GCV score, so that the optimal smoothing parameter, from those tried, is $\hat{\lambda} \approx 18$. Plots of the GCV score and the optimal model are easily produced

```
plot(rho,V,type="l",xlab=expression(log(lambda)),
                    main="GCV score")
sp <- exp(rho[V==min(V)])      ## extract optimal sp
b <- prs.fit(wear,size,sj,sp) ## re-fit
plot(size,wear,main="GCV optimal fit")
lines(s,Xp %*% coef(b))
```

The results are shown in figure 4.9.

4.2.4 The Bayesian/mixed model alternative

At some level we introduce smoothing penalties because we believe that the truth is more likely to be smooth than wiggly. We might as well formalise this belief in a Bayesian way, and specify a prior distribution on function wiggliness. Perhaps the simplest choice is an exponential prior

$$\propto \exp(-\lambda \boldsymbol{\beta}^{\mathsf{T}} \mathbf{S} \boldsymbol{\beta} / \sigma^2)$$

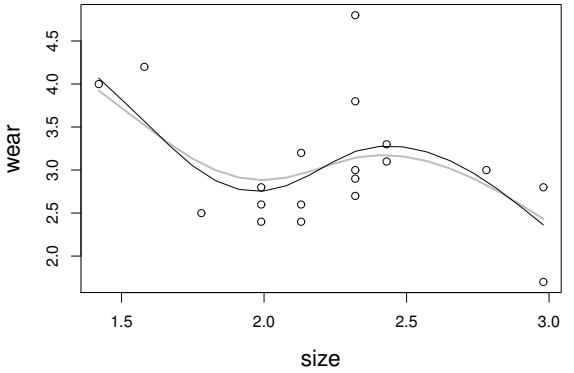

Figure 4.10 *Smooth model fits to the engine wear data with smoothing parameters estimated using marginal likelihood maximization (grey) or REML (black).*

(where scaling by σ^2 is introduced merely for later convenience), but this is immediately recognisable as being equivalent to an improper multivariate normal prior $\boldsymbol{\beta} \sim N(\mathbf{0}, \sigma^2\mathbf{S}^-/\lambda)$. That is, the prior precision matrix is proportional to $\mathbf{S}$: because $\mathbf{S}$ is rank deficient by the dimension of the penalty null space, the prior covariance matrix is proportional to the pseudo-inverse[†] $\mathbf{S}^-$.

This Bayesian interpretation of the smoothing penalty gives the model the structure of a linear mixed model as discussed in chapter 2, and in consequence the MAP estimate of $\boldsymbol{\beta}$ is the solution (4.7) to (4.6), while

$$\boldsymbol{\beta}|\mathbf{y} \sim N(\hat{\boldsymbol{\beta}}, (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{S})^{-1}\sigma^2)$$

— the Bayesian posterior distribution of $\boldsymbol{\beta}$ (this is equivalent to (2.17), p. 80). Also, having given the model this extra structure opens up the possibility of estimating σ^2 and λ using marginal likelihood maximization or REML.

In this section we will re-parameterize slightly to get the smooth model into a form such that its marginal likelihood can be evaluated using the simple routine `llm` from section 2.4.4 (p. 81). R routine `optim` can be used to fit the model. The same re-parameterization allows the model to be easily estimated using `lme` (see section 2.5, p. 86). As we will see in chapter 6, this re-parameterization is not necessary: it just simplifies matters for the moment, and perhaps makes the relationship between fixed effects and the penalty null space clearer than might otherwise be the case.

The re-parameterization is to re-write the model in terms of parameters, $\boldsymbol{\beta}' = \mathbf{D}_+\boldsymbol{\beta}$ where

$$\mathbf{D}_+ = \begin{bmatrix} \mathbf{I}_2 & \mathbf{0} \\ & \mathbf{D} \end{bmatrix}.$$

So we now have $\mathbf{X}\boldsymbol{\beta} = \mathbf{X}\mathbf{D}_+^{-1}\boldsymbol{\beta}'$ and $\boldsymbol{\beta}^\mathsf{T}\mathbf{S}\boldsymbol{\beta} = \sum_{i=3}^{k}\beta_i'^2$. If we write the first 2

[†] Consider eigen-decomposition $\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^\mathsf{T}$. Let $\boldsymbol{\Lambda}^-$ denote the diagonal matrix of the inverse of the non-zero eigenvalues, with zeroes in place of the inverse for any zero eigenvalues. Then $\mathbf{S}^- = \mathbf{U}\boldsymbol{\Lambda}^-\mathbf{U}^\mathsf{T}$.

elements of $\boldsymbol{\beta}'$ as $\boldsymbol{\beta}^*$ and the remainder as $\mathbf{b}$, the Bayesian smoothing prior becomes $\mathbf{b} \sim N(\mathbf{0}, \mathbf{I}\sigma^2/\lambda)$ (which is proper). $\boldsymbol{\beta}^*$ is completely unpenalized, so we treat this as a vector of fixed effects. To make the connection to a standard mixed model completely clear, let $\mathbf{X}^*$ now denote the first 2 columns of $\mathbf{X}\mathbf{D}_+^{-1}$, while $\mathbf{Z}$ is the matrix of the remaining columns. Then the smooth model has become

$$\mathbf{y} = \mathbf{X}^*\boldsymbol{\beta}^* + \mathbf{Z}\mathbf{b} + \boldsymbol{\epsilon}, \quad \mathbf{b} \sim N(\mathbf{0}, \mathbf{I}\sigma^2/\lambda), \quad \boldsymbol{\epsilon} \sim N(\mathbf{0}, \mathbf{I}\sigma^2)$$

which is self-evidently in the standard form of a linear mixed model given in section 2.3, (p.77). Here is the code to re-parameterize the model and estimate it using `optim` and `llm` from section 2.4.4 (p. 81):

```
X0 <- tf.X(size,sj)          ## X in original parameterization
D <- rbind(0,0,diff(diag(20),difference=2))
diag(D) <- 1                    ## augmented D
X <- t(backsolve(t(D),t(X0)))  ## re-parameterized X
Z <- X[,-c(1,2)]; X <- X[,1:2] ## mixed model matrices
## estimate smoothing and variance parameters...
m <- optim(c(0,0),llm,method="BFGS",X=X,Z=Z,y=wear)
b <- attr(llm(m$par,X,Z,wear),"b") ## extract coefficients
## plot results...
plot(size,wear)
Xp1 <- t(backsolve(t(D),t(Xp))) ## re-parameterized pred. mat.
lines(s,Xp1 %*% as.numeric(b),col="grey",lwd=2)
```

The resulting plot is shown in figure 4.10.

Estimation using REML via `lme` is also easy. In `lme` terms all the data belong to a single group, so to use `lme` we must create a dummy grouping variable enforcing this. A covariance matrix proportional to the identity matrix is then specified via the `pdIdent` function.

```
library(nlme)
g <- factor(rep(1,nrow(X)))         ## dummy factor
m <- lme(wear ~ X - 1, random=list(g = pdIdent(~ Z-1)))
lines(s,Xp1 %*% as.numeric(coef(m))) ## and to plot
```

The curve of the estimated smooth is shown in black in figure 4.10. Notice how the REML based estimate (black) is more variable than the ML based estimate (grey), as expected from section 2.4.5 (p. 83).

4.3 Additive models

Now suppose that two explanatory variables, x and v, are available for a response variable, y, and that a simple additive model structure,

$$y_i = \alpha + f_1(x_i) + f_2(v_i) + \epsilon_i, \tag{4.8}$$

is appropriate. α is an intercept parameter, the f_j are smooth functions, and the ϵ_i are independent $N(0, \sigma^2)$ random variables.

There are two points to note about this model. Firstly, the assumption of additive effects is a fairly strong one: $f_1(x) + f_2(v)$ is a quite restrictive special case of

the general smooth function of two variables $f(x, v)$. Secondly, the fact that the model now contains more than one function introduces an identifiability problem: f_1 and f_2 are each only estimable to within an additive constant. To see this, note that any constant could be simultaneously added to f_1 and subtracted from f_2, without changing the model predictions. Hence identifiability constraints have to be imposed on the model before fitting.

Provided that the identifiability issue is addressed, the additive model can be represented using penalized regression splines, estimated by penalized least squares and the degree of smoothing selected by cross validation or (RE)ML, in the same way as for the simple univariate model.

4.3.1 Penalized piecewise regression representation of an additive model

Each smooth function in (4.8) can be represented using a penalized piecewise linear basis. Specifically, let

$$f_1(x) = \sum_{j=1}^{k_1} b_j(x)\delta_j$$

where the δ_j are unknown coefficients, while the $b_j(x)$ are basis functions of the form (4.4), defined using a sequence of k_1 knots, x_j^*, evenly spaced over the range of x. Similarly

$$f_2(v) = \sum_{j=1}^{k_2} \mathcal{B}_j(v)\gamma_j$$

where the γ_j are the unknown coefficients and the $\mathcal{B}_j(v)$ are basis functions of the form (4.4), defined using a sequence of k_2 knots, v_j^*, evenly spaced over the range of v. Defining n-vector $\mathbf{f}_1 = [f_1(x_1), \ldots, f_1(x_n)]^\mathsf{T}$, we have $\mathbf{f}_1 = \mathbf{X}_1\boldsymbol{\delta}$ where $b_j(x_i)$ is element i, j of $\mathbf{X}_1$. Similarly, $\mathbf{f}_2 = \mathbf{X}_2\boldsymbol{\gamma}$, where $\mathcal{B}_j(v_i)$ is element i, j of $\mathbf{X}_2$.

A penalty of the form (4.5) is also associated with each function: $\boldsymbol{\delta}^\mathsf{T}\mathbf{D}_1^\mathsf{T}\mathbf{D}_1\boldsymbol{\delta} = \boldsymbol{\delta}^\mathsf{T}\bar{\mathbf{S}}_1\boldsymbol{\delta}$ for f_1 and $\boldsymbol{\gamma}^\mathsf{T}\mathbf{D}_2^\mathsf{T}\mathbf{D}_2\boldsymbol{\gamma} = \boldsymbol{\gamma}^\mathsf{T}\bar{\mathbf{S}}_2\boldsymbol{\gamma}$ for f_2.

Now it is necessary to deal with the identifiability problem. For estimation purposes, almost any linear constraint that removed the problem could be used, but most choices lead to uselessly wide confidence intervals for the constrained functions. The best constraints from this viewpoint are sum-to-zero constraints, such as

$$\sum_{i=1}^{n} f_1(x_i) = 0, \quad \text{or equivalently } \mathbf{1}^\mathsf{T}\mathbf{f}_1 = 0,$$

where $\mathbf{1}$ is an n vector of 1's. Notice how this constraint still allows f_1 to have exactly the same shape as before constraint, with exactly the same penalty value. The constraint's only effect is to shift f_1, vertically, so that its mean value is zero.

To apply the constraint, note that we require $\mathbf{1}^\mathsf{T}\mathbf{X}_1\boldsymbol{\delta} = 0$ for all $\boldsymbol{\delta}$, which implies that $\mathbf{1}^\mathsf{T}\mathbf{X}_1 = \mathbf{0}$. To achieve this latter condition the column mean can be subtracted from each column of $\mathbf{X}_1$. That is, we define a column centred matrix

$$\tilde{\mathbf{X}}_1 = \mathbf{X}_1 - \mathbf{1}\mathbf{1}^\mathsf{T}\mathbf{X}_1/n$$

and set $\tilde{\mathbf{f}}_1 = \tilde{\mathbf{X}}_1\boldsymbol{\delta}$. It's easy to check that this constraint imposes no more than a shift in the level of $\mathbf{f}_1$:

$$\tilde{\mathbf{f}}_1 = \tilde{\mathbf{X}}_1\boldsymbol{\delta} = \mathbf{X}_1\boldsymbol{\delta} - \mathbf{1}\mathbf{1}^\mathsf{T}\mathbf{X}_1\boldsymbol{\delta}/n = \mathbf{X}_1\boldsymbol{\delta} - \mathbf{1}c = \mathbf{f}_1 - c$$

by definition of the scalar $c = \mathbf{1}^\mathsf{T}\mathbf{X}_1\boldsymbol{\delta}/n$. Finally note that the column centring reduces the rank of $\tilde{\mathbf{X}}_1$ to $k_1 - 1$, so that only $k_1 - 1$ elements of the k_1 vector $\boldsymbol{\delta}$ can be uniquely estimated. A simple identifiability constraint deals with this problem: a single element of $\boldsymbol{\delta}$ is set to zero, and the corresponding column of $\tilde{\mathbf{X}}_1$ and $\mathbf{D}$ is deleted.[‡] The column centred rank reduced basis will automatically satisfy the identifiability constraint. In what follows the tildes will be dropped, and it is assumed that the $\mathbf{X}_j$, $\mathbf{D}_j$, etc. are the constrained versions.

Here is an R function which produces constrained versions of $\mathbf{X}_j$ and $\mathbf{D}_j$.

```
tf.XD <- function(x,xk,cmx=NULL,m=2) {
## get X and D subject to constraint
  nk <- length(xk)
  X <- tf.X(x,xk)[,-nk]                    ## basis matrix
  D <- diff(diag(nk),differences=m)[,-nk] ## root penalty
  if (is.null(cmx)) cmx <- colMeans(X)
  X <- sweep(X,2,cmx)         ## subtract cmx from columns
  list(X=X,D=D,cmx=cmx)
}
```

`tf.XD` calls the functions producing the unconstrained basis and square root penalty matrices, given knot sequence `xk` and covariate values `x`. It drops a column of each resulting matrix and centres the remaining columns of the basis matrix. `cmx` is the vector of values to subtract from the columns of the `X`. For setting up a basis `cmx` should be `NULL`, in which case it is set to the column means of the basis matrix `X`. However, when using `tf.XD` to produce a basis matrix for *predicting* at new covariate values, it is essential that the basis matrix columns are centred using the same constants used for the *original* basis setup, so these must be supplied. Later code will clarify this.

Having set up constrained bases for the f_j it is now straightforward to re-express (4.8) as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where $\mathbf{X} = (\mathbf{1}, \mathbf{X}_1, \mathbf{X}_2)$ and $\boldsymbol{\beta}^\mathsf{T} = (\alpha, \boldsymbol{\delta}^\mathsf{T}, \boldsymbol{\gamma}^\mathsf{T})$. Largely for later notational convenience it is useful to express the penalties as quadratic forms in the full coefficient vector $\boldsymbol{\beta}$, which is easily done by simply padding out $\bar{\mathbf{S}}_j$ with zeroes, as appropriate. For example,

$$\boldsymbol{\beta}^\mathsf{T}\mathbf{S}_1\boldsymbol{\beta} = (\alpha, \boldsymbol{\delta}^\mathsf{T}, \boldsymbol{\gamma}^\mathsf{T}) \begin{bmatrix} 0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{S}}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \alpha \\ \boldsymbol{\delta} \\ \boldsymbol{\gamma} \end{bmatrix} = \boldsymbol{\delta}^\mathsf{T}\bar{\mathbf{S}}_1\boldsymbol{\delta}.$$

[‡]The recipe given here is applicable to any basis which includes the constant function in its span, and has a penalty that is zero for constant functions. However, for bases that explicitly include a constant function, it is not sufficient to set any coefficient to zero: the coefficient for the constant is the one to constrain.

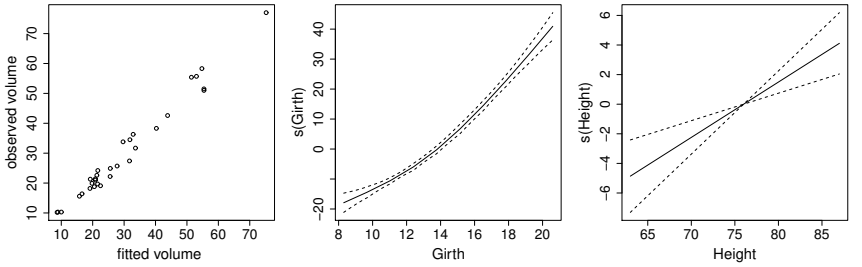

Figure 4.11 *The best fit two term additive model for the* `tree` *data. The left panel shows actual versus predicted tree volumes. The middle panel is the estimate of the smooth function of girth. The right panel is the estimate of the smooth function of height.*

4.3.2 Fitting additive models by penalized least squares

The coefficient estimates $\hat{\boldsymbol{\beta}}$ of the model (4.8) are obtained by minimization of the penalized least squares objective

$$\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda_1 \boldsymbol{\beta}^{\mathsf{T}} \mathbf{S}_1 \boldsymbol{\beta} + \lambda_2 \boldsymbol{\beta}^{\mathsf{T}} \mathbf{S}_2 \boldsymbol{\beta},$$

where the smoothing parameters λ_1 and λ_2 control the weight to be given to the objective of making f_1 and f_2 smooth, relative to the objective of closely fitting the response data. For the moment, assume that these smoothing parameters are given.

Similarly to the single smooth case we have

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda_1 \mathbf{S}_1 + \lambda_2 \mathbf{S}_2\right)^{-1} \mathbf{X}^{\mathsf{T}} \mathbf{y} \text{ and } \mathbf{A} = \mathbf{X}\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \lambda_1 \mathbf{S}_1 + \lambda_2 \mathbf{S}_2\right)^{-1} \mathbf{X}^{\mathsf{T}},$$

but again these expressions are sub-optimal with regard to computational stability and it is better to re-write the objective as

$$\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda_1 \boldsymbol{\beta}^{\mathsf{T}} \mathbf{S}_1 \boldsymbol{\beta} + \lambda_2 \boldsymbol{\beta}^{\mathsf{T}} \mathbf{S}_2 \boldsymbol{\beta} = \left\| \begin{bmatrix} \mathbf{y} \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{X} \\ \mathbf{B} \end{bmatrix} \boldsymbol{\beta} \right\|^2, \tag{4.9}$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{0} & \sqrt{\lambda_1}\mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \sqrt{\lambda_2}\mathbf{D}_2 \end{bmatrix}$$

(or any other matrix such that $\mathbf{B}^{\mathsf{T}}\mathbf{B} = \lambda_1 \mathbf{S}_1 + \lambda_2 \mathbf{S}_2$).

As in the single smooth case, the right hand side of (4.9) is simply the unpenalized least squares objective for an augmented version of the model and corresponding response data. Hence, the model can be fitted by standard linear regression using stable orthogonal matrix based methods.

Here is a function to set up and fit a simple two term additive model, assuming the same number of knots for each smooth.

```
am.fit <- function(y,x,v,sp,k=10) {
  ## setup bases and penalties...
  xk <- seq(min(x),max(x),length=k)
  xdx <- tf.XD(x,xk)
  vk <- seq(min(v),max(v),length=k)
  xdv <- tf.XD(v,vk)
  ## create augmented model matrix and response...
  nD <- nrow(xdx$D)*2
  sp <- sqrt(sp)
  X <- cbind(c(rep(1,nrow(xdx$X)),rep(0,nD)),
             rbind(xdx$X,sp[1]*xdx$D,xdv$D*0),
             rbind(xdv$X,xdx$D*0,sp[2]*xdv$D))
  y1 <- c(y,rep(0,nD))
  ## fit model..
  b <- lm(y1 ~ X - 1)
  ## compute some useful quantities...
  n <- length(y)
  trA <- sum(influence(b)$hat[1:n]) ## EDF
  rsd <- y - fitted(b)[1:n] ## residuals
  rss <- sum(rsd^2)         ## residual SS
  sig.hat <- rss/(n-trA)       ## residual variance
  gcv <- sig.hat*n/(n-trA)     ## GCV score
  Vb <- vcov(b)*sig.hat/summary(b)$sigma^2 ## coeff cov matrix
  ## return fitted model...
  list(b=coef(b),Vb=Vb,edf=trA,gcv=gcv,fitted=fitted(b)[1:n],
       rsd=rsd,xk=list(xk,vk),cmx=list(xdx$cmx,xdv$cmx))
}
```

In addition to the quantities that we met in the single smooth case, `am.fit` also returns an estimate of the Bayesian covariance matrix for the model coefficients:

$$\hat{\mathbf{V}}_\beta = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda_1\mathbf{S}_1 + \lambda_2\mathbf{S}_2)^{-1}\hat{\sigma}^2$$

where $\hat{\sigma}^2$ is taken as the residual sum of squares for the fitted model, divided by the effective residual degrees of freedom. Following section 4.2.4 the posterior distribution for $\boldsymbol{\beta}$ is

$$\boldsymbol{\beta}|\mathbf{y} \sim N(\hat{\boldsymbol{\beta}}, \mathbf{V}_\beta), \tag{4.10}$$

and this result can be used for further inference about $\boldsymbol{\beta}$ (see section 6.10, p. 293).

Let us use the routine to estimate an additive model for the data in R data frame `trees`. The data are `Volume`, `Girth` and `Height` for 31 felled cherry trees. Interest lies in predicting `Volume`, and we can try estimating the model

$$\texttt{Volume}_i = \alpha + f_1(\texttt{Girth}_i) + f_2(\texttt{Height}_i) + \epsilon_i.$$

Now that we have two smoothing parameters, grid searching for the GCV optimal values starts to become inefficient. Instead R function `optim` can be used to minimize the GCV score. The function to be optimised has to be in a particular form for use with `optim`: the optimization parameter vector must be the first argument, and the function must be real valued. A simple wrapper for `am.fit` suffices:

```
am.gcv <- function(lsp,y,x,v,k) {
## function suitable for GCV optimization by optim
  am.fit(y,x,v,exp(lsp),k)$gcv
}
```

Using log smoothing parameters for optimization ensures that the estimated smoothing parameters are non-negative. Fitting the model is now straightforward

```
## find GCV optimal smoothing parameters...
fit <- optim(c(0,0), am.gcv, y=trees$Volume, x=trees$Girth,
             v=trees$Height,k=10)
sp <- exp(fit$par) ## best fit smoothing parameters
## Get fit at GCV optimal smoothing parameters...
fit <- am.fit(trees$Volume,trees$Girth,trees$Height,sp,k=10)
```

Now let's plot the smooth effects. The following function will do this.

```
am.plot <- function(fit,xlab,ylab) {
## produces effect plots for simple 2 term
## additive model
  start <- 2 ## where smooth coeffs start in beta
  for (i in 1:2) {
    ## sequence of values at which to predict...
    x <- seq(min(fit$xk[[i]]), max(fit$xk[[i]]), length=200)
    ## get prediction matrix for this smooth...
    Xp <- tf.XD(x, fit$xk[[i]], fit$cmx[[i]])$X
    ## extract coefficients and cov matrix for this smooth
    stop <- start + ncol(Xp)-1; ind <- start:stop
    b <- fit$b[ind];Vb <- fit$Vb[ind,ind]
    ## values for smooth at x...
    fv <- Xp %*% b
    ## standard errors of smooth at x....
    se <- rowSums((Xp %*% Vb) * Xp)^.5
    ## 2 s.e. limits for smooth...
    ul <- fv + 2 * se; ll <- fv - 2 * se
    ## plot smooth and limits...
    plot(x, fv, type="l", ylim=range(c(ul,ll)), xlab=xlab[i],
         ylab=ylab[i])
    lines(x, ul, lty=2); lines(x, ll, lty=2)
    start <- stop + 1
  }
}
```

Calling it with the fitted tree model

```
par(mfrow=c(1,3))
plot(fit$fitted,trees$Vol,xlab="fitted volume ",
     ylab="observed volume")
am.plot(fit,xlab=c("Girth","Height"),
        ylab=c("s(Girth)","s(Height)"))
```

gives the result in figure 4.11. Notice that the smooth of `Height` is estimated to be a straight line, and as a result its confidence interval has zero width at some point.

The zero width point in the interval occurs because the sum to zero constraint exactly determines where the straight line must pass through zero.

As with the one dimensional smooth, the additive model could also be estimated as a linear mixed model, but let us move on.

4.4 Generalized additive models

Generalized additive models (GAMs) follow from additive models, as generalized linear models follow from linear models. That is, the linear predictor now predicts some known smooth monotonic function of the expected value of the response, and the response may follow any exponential family distribution, or simply have a known mean variance relationship, permitting the use of a quasi-likelihood approach. The resulting model has a general form something like (4.1) in section 4.1.

As an illustration, suppose that we would like to model the `trees` data using a GAM of the form:

$$\log\{\mathbb{E}(\texttt{Volume}_i)\} = f_1(\texttt{Girth}_i) + f_2(\texttt{Height}_i), \quad \texttt{Volume}_i \sim \text{ gamma.}$$

This model is perhaps more natural than the additive model, as we might expect volume to be the product of some function of girth and some function of height, and it is reasonable to expect the variance in volume to increase with mean volume.

Whereas the additive model was estimated by penalized least squares, the GAM will be fitted by penalized likelihood maximization, and in practice this will be achieved by penalized iterative least squares (PIRLS).[§] For given smoothing parameters, the following steps are iterated to convergence.

1. Given the current linear predictor estimate, $\hat{\boldsymbol{\eta}}$, and corresponding estimated mean response vector, $\hat{\boldsymbol{\mu}}$, calculate:

$$w_i = \frac{1}{V(\hat{\mu}_i)g'(\hat{\mu}_i)^2} \quad \text{and} \quad z_i = g'(\hat{\mu}_i)(y_i - \hat{\mu}_i) + \hat{\eta}_i$$

   where $\text{var}(Y_i) = V(\mu_i)\phi$, as in section 3.1.2, and g is the link function.

2. Defining $\mathbf{W}$ as the diagonal matrix such that $W_{ii} = w_i$, minimize

$$\|\sqrt{\mathbf{W}}\mathbf{z} - \sqrt{\mathbf{W}}\mathbf{X}\boldsymbol{\beta})\|^2 + \lambda_1\boldsymbol{\beta}^{\mathsf{T}}\mathbf{S}_1\boldsymbol{\beta} + \lambda_2\boldsymbol{\beta}^{\mathsf{T}}\mathbf{S}_2\boldsymbol{\beta}$$

   w.r.t. $\boldsymbol{\beta}$ to obtain new estimate $\hat{\boldsymbol{\beta}}$, and hence updated estimates $\hat{\boldsymbol{\eta}} = \mathbf{X}\hat{\boldsymbol{\beta}}$ and $\hat{\mu}_i = g^{-1}(\hat{\eta}_i)$.

The penalized least squares problem at step 2 is exactly the problem already solved for the simple additive model. Note the link to section 3.4.1 (p. 148).

For the trees GAM, the link function, g, is the log function, so $g'(\mu_i) = \mu_i^{-1}$, while for the gamma distribution, $V(\mu_i) = \mu_i^2$ (see table 3.1, p. 104). Hence, for the log-link gamma model, we have:

$$w_i = 1 \quad \text{and} \quad z_i = (y_i - \hat{\mu}_i)/\hat{\mu}_i + \hat{\eta}_i.$$

[§] There is no simple trick to produce an unpenalized GLM whose likelihood is equivalent to the penalized likelihood of the GAM that we wish to fit.

So, given λ_1 and λ_2 it will be straightforward to obtain $\hat{\boldsymbol{\beta}}$, but what should be used as the GCV score for this model? A natural choice is to use the GCV score for the final linear model in the PIRLS iteration (although this choice is poor for binary data: see section 6.2, p. 255 for better performing alternatives). It is easy to show that this GCV score is equivalent to the usual GCV score, but with the Pearson statistic replacing the residual sum of squares. Obviously we could also estimate the smoothing parameters by exploiting the Bayesian/mixed model connection of section 4.2.4, and estimating the model as a generalized linear mixed model using the methods of section 3.4 (p. 147).

The following function implements the PIRLS loop for the log-gamma model, and returns the required GCV score in its return list.

```
gam.fit <- function(y,x,v,sp,k=10) {
## gamma error log link 2 term gam fit...
  eta <- log(y) ## get initial eta
  not.converged <- TRUE
  old.gcv <- -100 ## don't converge immediately
  while (not.converged) {
    mu <- exp(eta)  ## current mu estimate
    z <- (y - mu)/mu + eta ## pseudodata
    fit <- am.fit(z,x,v,sp,k) ## penalized least squares
    if (abs(fit$gcv-old.gcv)<1e-5*fit$gcv) {
      not.converged <- FALSE
    }
    old.gcv <- fit$gcv
    eta <- fit$fitted ## updated eta
  }
  fit$fitted <- exp(fit$fitted) ## mu
  fit
}
```

Again a simple wrapper is needed in order to optimize the GCV score using `optim`

```
gam.gcv <- function(lsp,y,x,v,k=10) {
  gam.fit(y,x,v,exp(lsp),k=k)$gcv
}
```

Now fitting and plotting proceeds exactly as in the simple additive case.

```
fit <- optim(c(0,0),gam.gcv,y=trees$Volume,x=trees$Girth,
             v=trees$Height,k=10)
sp <- exp(fit$par)
fit <- gam.fit(trees$Volume,trees$Girth,trees$Height,sp)
par(mfrow=c(1,3))
plot(fit$fitted,trees$Vol,xlab="fitted volume ",
     ylab="observed volume")
am.plot(fit,xlab=c("Girth","Height"),
        ylab=c("s(Girth)","s(Height)"))
```

The resulting plots are shown in figure 4.12.

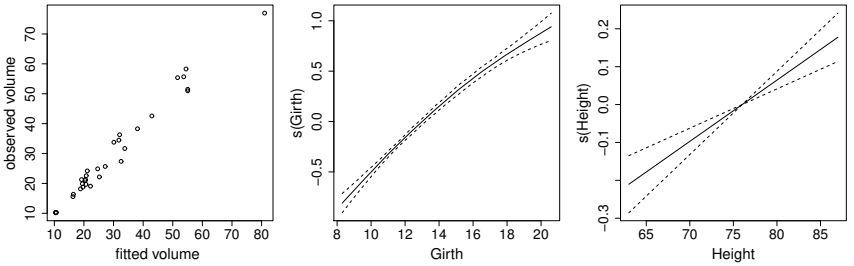

Figure 4.12 *The best fit two term generalized additive model for the* `tree` *data. The left panel shows actual versus predicted tree volumes. The middle panel is the estimate of the smooth function of girth. The right panel is the estimate of the smooth function of height.*

4.5 Summary

The preceding sections have illustrated how models based on smooth functions of predictor variables can be represented, and estimated, once a basis and wiggliness penalty have been chosen for each smooth in the model. Estimation is by penalized versions of the least squares and maximum-likelihood methods used for linear models and GLMs. Indeed technically GAMs are simply GLMs estimated subject to smoothing penalties. The most substantial difficulty introduced by this penalization is the need to select the degree of penalization, that is, to estimate the smoothing parameters. As we have seen, GCV provides one quite reasonable solution, and marginal likelihood provides an alternative.

The rest of this book will stick to the basic framework presented here, simply adding refinements to it. We will consider a variety of basis-penalty smoothers somewhat preferable to the piecewise linear basis given here, and some alternatives to GCV for smoothness estimation. More efficient and reliable computational methods will be developed, and the theoretical basis for inference will be more fully expounded. The link between smooths and random effects will also be developed, as will models based on linear functionals of smooths. However, throughout, functions are represented using penalized basis expansions, estimation of coefficients is by penalized likelihood maximisation and estimation of smoothing parameters uses a separate criterion, such as GCV or REML.

4.6 Introducing package `mgcv`

Before considering smoothers and GAM theory in more detail, it is worth briefly introducing the `mgcv` package. The `gam` function from `mgcv` is very much like the `glm` function covered in chapter 3. The main difference is that the `gam` model formula can include smooth terms, `s()` and `te()` (as well as the `te` variants `ti` and `t2`). Also there are a number of options available for controlling automatic smoothing parameter estimation, or for directly controlling model smoothness (summarized in table 4.1).

The cherry tree data provide a simple example with which to introduce the modelling functions available in R package mgcv.

```
library(mgcv)    ## load the package
data(trees)
ct1 <- gam(Volume ~ s(Height) + s(Girth),
           family=Gamma(link=log),data=trees)
```

This fits the generalized additive model

$$\log(\mathbb{E}[\texttt{Volume}_i]) = f_1(\texttt{Height}_i) + f_2(\texttt{Girth}_i) \quad \text{where } \texttt{Volume}_i \sim \text{gamma}$$

and the f_j are smooth functions. By default, the degree of smoothness of the f_j (within certain limits) is estimated by GCV. The results can be checked by typing the name of the fitted model object to invoke the `print.gam` print method, and by plotting the fitted model object. For example

```
> ct1

Family: Gamma
Link function: log

Formula:
Volume ~ s(Height) + s(Girth)

Estimated degrees of freedom:
1.00 2.42  total = 4.42

GCV score: 0.008082356
> plot(ct1,residuals=TRUE)
```

The resulting plot is displayed in the upper two panels of figure 4.13. Notice that the default print method reports the model distribution family, link function and formula, before displaying the effective degrees of freedom for each term (in the order that the terms appear in the model formula) and the whole model: in this case a nearly straight line, corresponding to about one degree of freedom, is estimated for the effect of height, while the effect of girth is estimated as a smooth curve with 2.4 degrees of freedom; the total degrees of freedom is the sum of these two, plus one degree of freedom for the model intercept. Finally, the GCV score for the fitted model is reported.

The plots show the estimated effects as solid lines/curves, with 95% confidence limits (strictly Bayesian credible intervals; see section 6.10, p. 293), based on (4.10), shown as dashed lines. The coincidence of the confidence limits and the estimated straight line, at the point where the line passes through zero on the vertical axis, is a result of the identifiability constraints applied to the smooth terms.[¶] The points shown on the plots are *partial residuals*. These are simply the Pearson residuals

[¶]The identifiability constraint is that the sum of the values of each curve, at the observed covariate values, must be zero: for a straight line, this condition exactly determines where the line must pass through zero, so there can be no uncertainty about this point.

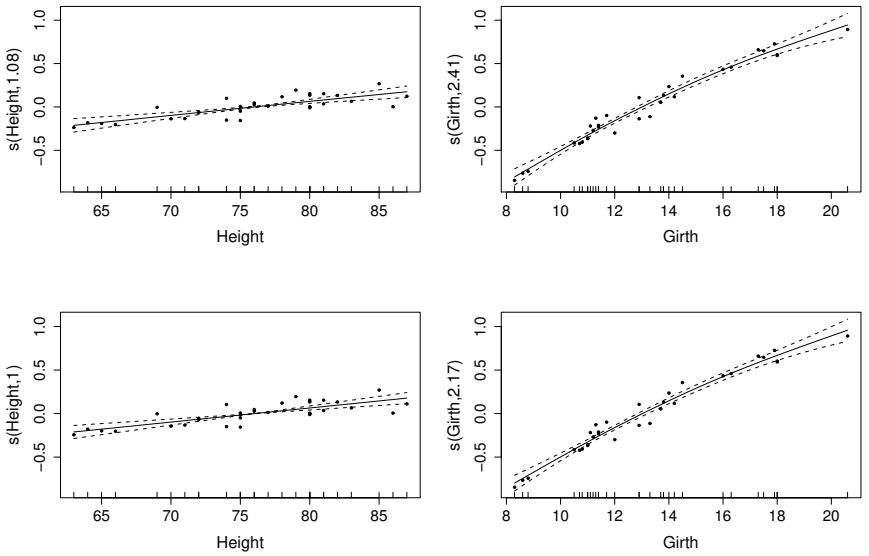

Figure 4.13 *Components of GAM model fits to the cherry tree data. The upper two panels are from* `ct1` *and the lower 2 from* `ct4`.

added to the smooth terms evaluated at the appropriate covariate values. For example, the residuals plotted in the top left panel of figure 4.13 are given by

$$\hat{\epsilon}_{1i}^{\text{partial}} = f_1(\texttt{Height}_\texttt{i}) + \hat{\epsilon}_i^p$$

plotted against $\texttt{Height}_i$. For a well fitting model the partial residuals should be evenly scattered around the curve to which they relate. The 'rug plots', along the bottom of each plot, show the values of the covariates of each smooth. The number in each y-axis caption is the effective degrees of freedom of the term being plotted.

4.6.1 Finer control of `gam`

The simple form of the `gam` call producing `ct1` hides a number of options that have been set to default values. The first of these is the choice of basis used to represent the smooth terms. The default is to use thin plate regression splines (section 5.5.1, p. 215), which have some appealing properties, but can be somewhat computationally costly for large data sets. The full range of available smoothers is covered in chapter 5. In the following, penalized cubic regression splines are selected using `s(...,bs="cr")`.

```
> ct2 <- gam(Volume ~ s(Height,bs="cr") + s(Girth,bs="cr"),
+            family=Gamma(link=log),data=trees)
> ct2
```

scale	The value of the scale parameter, or a negative value if it is to be estimated. For method="GCV.Cp" then scale > 0 implies Mallows' C_p/UBRE/AIC is used. scale $< 0 \Rightarrow$ implies GCV is used. scale $= 0 \Rightarrow$ UBRE/AIC for Poisson or binomial, otherwise GCV.
gamma	This multiplies the model degrees of freedom in the GCV or UBRE/AIC criteria. Hence as gamma is increased from 1 the 'penalty' per degree of freedom increases in the GCV or UBRE/AIC criterion and increasingly smooth models are produced. Increasing gamma to around 1.5 can usually reduce over-fitting, without much degradation in prediction error performance.
sp	An array of supplied smoothing parameters. When this array is non-null, a negative element signals that a smoothing parameter should be estimated, while a non-negative value is used as the smoothing parameter for the corresponding term. This is useful for directly controlling the smoothness of some terms.
method	Selects the smoothing parameter selection criterion: GCV.Cp, GACV, ML or REML.

Table 4.1 *Main arguments to* gam *for controlling the smoothness estimation process.*

```
Family: Gamma
Link function: log

Formula:
Volume ~ s(Height, bs = "cr") + s(Girth, bs = "cr")

Estimated degrees of freedom:
 1.000126 2.418591   total =  4.418718

GCV score:  0.008080546
```

As you can see, the change in basis has made very little difference to the fit. Plots are almost indistinguishable to those for ct1. This is re-assuring: it would be unfortunate if the model depended very strongly on details like the exact choice of basis. However, larger changes to the basis, such as using P-splines (section 5.3.3, p. 204), can make an appreciable difference.

Another choice, hidden in the previous two model fits, is the *dimension*, k, of the basis used to represent smooth terms. In the previous two fits, the (arbitrary) default, $k = 10$, was used. The choice of basis dimensions amounts to setting the *maximum* possible degrees of freedom allowed for each model term. The actual effective degrees of freedom for each term will usually be estimated from the data, by GCV or another smoothness selection criterion, but the upper limit on this estimate is $k - 1$: the basis dimension minus one degree of freedom due to the identifiability constraint on each smooth term. The following example sets k to 20 for the smooth of Girth (and illustrates, by the way, that there is no problem in mixing different bases).

```
> ct3 <- gam(Volume ~ s(Height) + s(Girth,bs="cr",k=20),
+            family=Gamma(link=log),data=trees)
> ct3

Family: Gamma
Link function: log

Formula:
Volume ~ s(Height) + s(Girth, bs = "cr", k = 20)

Estimated degrees of freedom:
 1.000003 2.424226   total =  4.424229

GCV score:  0.00808297
```

Again, this change makes boringly little difference in this case, and the plots (not shown) are indistinguishable from those for `ct1`. This insensitivity to basis dimension is not universal, of course, and checking of this choice is covered in section 5.9 (p. 242). One quite subtle point is worth being aware of. This is that a space of functions of dimension 20 will contain a larger subspace of functions with effective degrees of freedom 5, than will a function space of dimension 10 (the particular numbers being arbitrary here). Hence it is often the case that increasing k will change the effective degrees of freedom estimated for a term, even though both old and new estimated degrees of freedom are lower than the original $k - 1$.

Another choice is the parameter `gamma` which can be used to multiply the model effective degrees of freedom in the GCV or UBRE scores in order to (usually) increase the amount of smoothing selected. The default value is 1, but GCV is known to have some tendency to overfitting on occasion, and it has been suggested that using $\gamma \approx 1.5$ can somewhat correct this without compromising model fit (e.g., Kim and Gu, 2004). See section 6.2.4 for one justification. Applying this idea to the current model results in the bottom row of figure 4.13 and the following output.

```
> ct4 <- gam(Volume ~ s(Height) + s(Girth),
+            family=Gamma(link=log),data=trees,gamma=1.4)
> ct4

Family: Gamma
Link function: log

Formula:
Volume ~ s(Height) + s(Girth)

Estimated degrees of freedom:
 1.00011 2.169248   total =  4.169358

GCV score:  0.00922805

> plot(ct4,residuals=TRUE)
```

The heavier penalty on each degree of freedom in the GCV score has resulted in

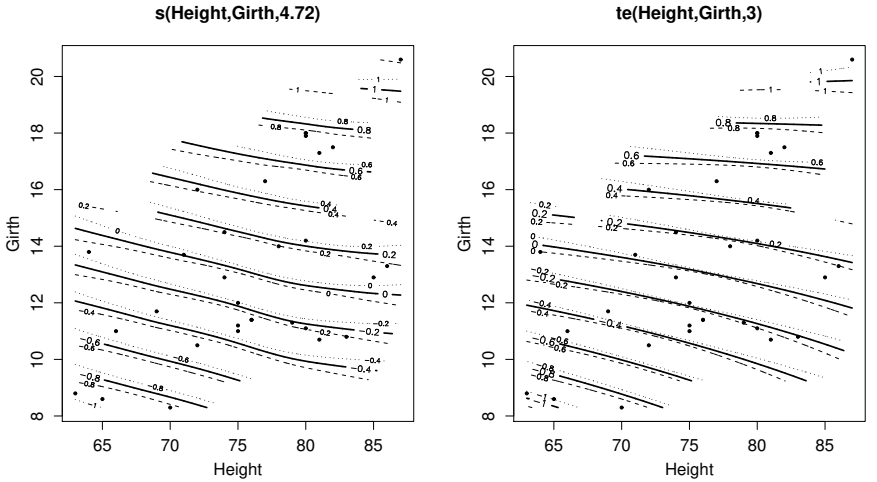

Figure 4.14 *Smooth functions of height and girth fitted to the cherry tree data, with degree of smoothing chosen by GCV. The left hand panel shows a thin plate regression spline fit (*`ct5`*), while the right panel shows a tensor product spline fit (*`ct6`*). For both plots the bold contours show the estimate of the smooth; the dashed contours show the smooth plus the standard error of the smooth and the dotted contours show the smooth less its standard error. The symbols show the locations of the covariate values on the height–girth plane. Parts of the smooths that are far away from covariate values have been excluded from the plots using the* `too.far` *argument to* `plot.gam`.

a model with fewer degrees of freedom, but the figure indicates that the change in estimates that this produces is barely perceptible.

4.6.2 Smooths of several variables

`gam` is not restricted to models containing only smooths of one predictor. In principle, smooths of any number of predictors are possible via two types of smooth. Within a model formula, `s()` terms, using the `"tp"`, `"ds"` or `"gp"` bases,[‖] produce isotropic smooths of multiple predictors, while `te()` terms produce smooths of multiple predictors from tensor products of *any* singly penalized bases available for use with `s()` (including mixtures of different bases). The tensor product smooths are invariant to linear rescaling of covariates, and can be quite computationally efficient. Alternative versions `t2()` and `ti()` are available for different sorts of functional ANOVA decomposition. Section 5.7 (p. 237) compares isotropic and tensor product smoothers.

[‖] Or indeed `"sos"` or `"so"` bases.

By way of illustration, the following code fragments both fit the model

$$\log(\mathbb{E}[\texttt{Volume}_i]) = f(\texttt{Height}_i, \texttt{Girth}_i) \text{ where } \texttt{Volume}_i \sim \text{gamma},$$

and f is a smooth function. Firstly an isotropic thin plate regression spline is used:

```
> ct5 <- gam(Volume ~ s(Height,Girth,k=25),
+            family=Gamma(link=log),data=trees)
> ct5

Family: Gamma
Link function: log

Formula:
Volume ~ s(Height, Girth, k = 25)

Estimated degrees of freedom:
 4.668129   total =  5.668129

GCV score:  0.009358786

> plot(ct5,too.far=0.15)
```

yielding the left hand panel of figure 4.14. Secondly a tensor product smooth is used. Note that the `k` argument to `te` specifies the dimension for each marginal basis: if different dimensions are required for the marginal bases then `k` can also be supplied as an array. The basis dimension of the tensor product smooth is the product of the dimensions of the marginal bases.

```
> ct6 <- gam(Volume ~ te(Height,Girth,k=5),
+            family=Gamma(link=log),data=trees)
> ct6

Family: Gamma
Link function: log

Formula:
Volume ~ te(Height, Girth, k = 5)

Estimated degrees of freedom:
 3.000175   total =  4.000175

GCV score:  0.008197151

> plot(ct6,too.far=0.15)
```

Notice how the tensor product model has fewer degrees of freedom and a lower GCV score than the TPRS smooth. In fact, with just 3 degrees of freedom, the tensor product smooth model amounts to

$$\log(\mathbb{E}[\texttt{Volume}_i]) = \beta_0 + \beta_1 \texttt{Height}_i + \beta_2 \texttt{Girth}_i + \beta_3 \texttt{Height}_i \texttt{Girth}_i,$$

the 'wiggly' components of the model having been penalized away altogether.

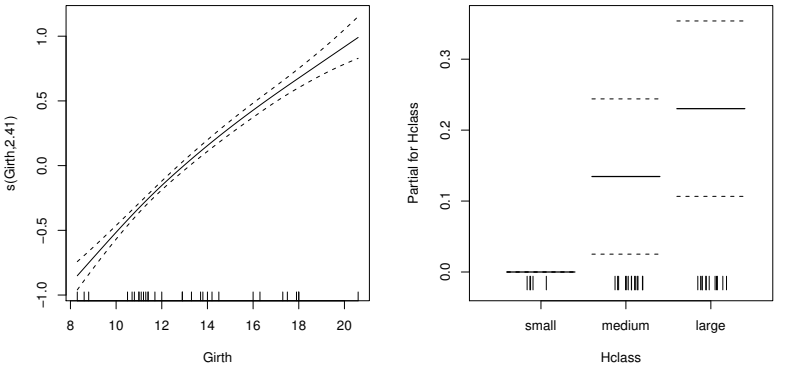

Figure 4.15 *Plot of model* `ct7`*, a* semi-parametric *model of cherry tree volume, with a factor for height and a smooth term for the dependence on girth. The left plot shows the smooth of girth, with 95% confidence interval, while the right panel shows the estimated effect, for each level of factor* `Hclass`*. The effect of being in the* `small` *height class is shown as zero, because the default contrasts have been used here, which set the parameter for the first level of each factor to zero.*

4.6.3 Parametric model terms

So far, only models consisting of smooth terms have been considered, but there is no difficulty in mixing smooth and parametric model components. For example, given that the model `ct1` smooth of height is estimated to be a straight line, we might as well fit the model:

```
gam(Volume ~ Height+s(Girth),family=Gamma(link=log),data=trees)
```

but to make the example more informative, let us instead suppose that the `Height` is actually only measured as a categorical variable. This can easily be arranged, by creating a factor variable which simply labels each tree as small, medium or large:

```
trees$Hclass <- factor(floor(trees$Height/10)-5,
                  labels=c("small","medium","large"))
```

Now we can fit a generalized additive model to these data, using the `Hclass` variable as a factor variable, and plot the result (figure 4.15).

```
ct7 <- gam(Volume ~ Hclass + s(Girth),
           family=Gamma(link=log), data=trees)
par(mfrow=c(1,2)); plot(ct7,all.terms=TRUE)
```

Often, more information about a fitted model is required than is supplied by plots or the default print method, and various utility functions exist to provide this. For example the `anova` function can be used to investigate the approximate significance of model terms.

```
> anova(ct7)

Family: Gamma
```

```
Link function: log

Formula:
Volume ~ Hclass + s(Girth)

Parametric Terms:
       df     F p-value
Hclass  2 7.076 0.00358

Approximate significance of smooth terms:
           edf Est.rank     F  p-value
s(Girth) 2.414    9.000 54.43 1.98e-14
```

Clearly there is quite strong evidence that both height and girth matter (see section 6.12, for information on the p-value calculations for the smooth terms). Similarly, an approximate AIC value can be obtained for the model (see section 6.11, p. 301):

```
> AIC(ct7)
[1] 154.9411
```

The `summary` method provides considerable detail.

```
> summary(ct7)

Family: Gamma
Link function: log

Formula:
Volume ~ Hclass + s(Girth)

Parametric coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.12693    0.04814  64.949  < 2e-16 ***
Hclassmedium  0.13459    0.05428   2.479 0.020085 *
Hclasslarge   0.23024    0.06137   3.752 0.000908 ***

Approximate significance of smooth terms:
           edf Est.rank     F  p-value
s(Girth) 2.414    9.000 54.43 1.98e-14 ***

R-sq.(adj) =  0.967   Deviance explained = 96.9%
GCV score = 0.012076   Scale est. = 0.0099671  n = 31
```

Notice that, in this case, the significance of individual parameters of the parametric terms is given, rather than whole term significance. Other measures of fit are also reported, such as the adjusted r^2 and percentage deviance explained, along with the GCV score, an estimate of the scale parameter of the model, and the number of data fitted.

4.6.4 The mgcv *help pages*

mgcv has quite extensive help pages, both documenting functions and attempting to provide overviews of a topic. The easiest way to access the pages is via the HTML versions, by typing `help.start()` in R, then navigating to the mgcv pages and browsing. Several pages are well worth knowing about:

- `mgcv-package` offers an overview of the package and what it offers.
- `family.mgcv` gives an overview of the distributions available.
- `smooth.terms` gives an overview of the smooths types available.
- `random.effects` is an overview of random effects in mgcv.
- `gam.models` reviews some aspects of model specification; `gam.selection` covers model selection options.
- `gam`, `bam`, `gamm` and `jagam` cover the main modelling functions.

4.7 Exercises

1. This question is about illustrating the problems with polynomial bases. First run

   ```
   set.seed(1)
   x<-sort(runif(40)*10)^.5
   y<-sort(runif(40))^0.1
   ```

   to simulate some apparently innocuous x, y data.

   (a) Fit 5^{th} and 10^{th} order polynomials to the simulated data using, e.g., `lm(y~poly(x,5))`.

   (b) Plot the x, y data, and overlay the fitted polynomials. (Use the `predict` function to obtain predictions on a fine grid over the range of the x data: only predicting at the data fails to illustrate the polynomial behaviour adequately).

   (c) One particularly simple basis for a cubic regression spline is $b_1(x) = 1$, $b_2(x) = x$ and $b_{j+2}(x) = |x - x_j^*|^3$ for $j = 1 \ldots q-2$, where q is the basis dimension, and the x_j^* are knot locations. Use this basis to fit a rank 11 cubic regression spline to the x, y data (using `lm` and evenly spaced knots).

   (d) Overlay the predicted curve according to the spline model, onto the existing x, y plot, and consider which basis you would rather use.

2. Polynomial models of the data from question 1 can also provide an illustration of why orthogonal matrix methods are preferable to fitting models by solution of the 'normal equations' $\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{\beta} = \mathbf{X}^\mathsf{T}\mathbf{y}$. The bases produced by `poly` are actually orthogonal polynomial bases, which are a numerically stable way of representing polynomial models, but if a naïve basis is used then a numerically badly behaved model can be created.

   ```
   form<-paste("I(x^",1:10,")",sep="",collapse="+")
   form <- as.formula(paste("y~",form))
   ```

   produces the model formula for a suitably ill-behaved model. Fit this model using `lm`, extract the model matrix from the fitted model object using `model.matrix`, and re-estimate the model parameters by solving the 'normal equations' given

above (see `?solve`). Compare the estimated coefficients in both cases, along with the fits. It is also instructive to increase the order of the polynomial by one or two and examine the results (and to decrease it to 5, say, in order to confirm that the QR and normal equations approaches agree if everything is 'well behaved'). Finally, note that the singular value decomposition (see B.10) provides a reliable way of diagnosing the linear dependencies that can cause problems when model fitting. `svd(X)$d` obtains the singular values of a matrix `X`. The largest divided by the smallest gives the 'condition number' of the matrix — a measure of how ill-conditioned computations with the matrix are likely to be.

3. Show that the $\boldsymbol{\beta}$ minimizing (4.6), in section 4.2.2, is given by (4.7).
4. Let $\mathbf{X}$ be an $n \times p$ model matrix, $\mathbf{S}$ a $p \times p$ penalty matrix, and $\mathbf{B}$ any matrix such that $\mathbf{B}^\mathsf{T}\mathbf{B} = \mathbf{S}$. If
$$\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{X} \\ \mathbf{B} \end{bmatrix}$$
is an augmented model matrix, show that the sum of the first n elements on the leading diagonal of $\tilde{\mathbf{X}}(\tilde{\mathbf{X}}^\mathsf{T}\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^\mathsf{T}$ is $\mathrm{tr}\left(\mathbf{X}(\mathbf{X}^\mathsf{T}\mathbf{X}+\mathbf{S})^{-1}\mathbf{X}^\mathsf{T}\right)$.
5. The 'obvious' way to estimate smoothing parameters is by treating smoothing parameters just like the other model parameters, $\boldsymbol{\beta}$, and to choose λ to minimize the residual sum of squares for the fitted model. What estimate of λ will such an approach always produce?
6. Show that for any function f, which has a basis expansion
$$f(x) = \sum_j \beta_j b_j(x),$$
it is possible to write
$$\int f''(x)^2 dx = \boldsymbol{\beta}^\mathsf{T}\mathbf{S}\boldsymbol{\beta},$$
where the coefficient matrix $\mathbf{S}$ can be expressed in terms of the known basis functions b_j (assuming that these possess at least two (integrable) derivatives). As usual $\boldsymbol{\beta}$ is a parameter vector with β_j in its j^{th} element.
7. Show that for any function f which has a basis expansion
$$f(x,z) = \sum_j \beta_j b_j(x,z),$$
it is possible to write
$$\int \left(\frac{\partial^2 f}{\partial x^2}\right)^2 + 2\left(\frac{\partial^2 f}{\partial x \partial z}\right)^2 + \left(\frac{\partial f^2}{\partial z^2}\right)^2 dxdz = \boldsymbol{\beta}^\mathsf{T}\mathbf{S}\boldsymbol{\beta},$$
where the coefficient matrix $\mathbf{S}$ can be expressed in terms of the known basis functions b_j (assuming that these possess at least two (integrable) derivatives w.r.t. x and z). Again, $\boldsymbol{\beta}$ is a parameter vector with β_j in its j^{th} element.
8. The additive model of section 4.3 can equally well be estimated as a mixed model.

(a) Write a function which converts the model matrix and penalty returned by `tf.XD` into mixed model form. Hint: because of the constraints the penalty null space is of dimension 1 now, leading to a slight modification of $\mathbf{D}_+$.

(b) Using your function from part (a) obtain the model matrices required to fit the two term additive tree model, and estimate it using `lme`. Because there are now two smooths, two `pdIdent` terms will be needed in the `random` list supplied to `lme`, which will involve two dummy grouping variables (which can just be differently named copies of the same variable).

(c) Produce residual versus fitted volume and raw volume against fitted volume plots.

(d) Produce plots of the two smooth effect estimates with partial residuals.

9. Following on from question 8, we can also estimate a GAM as a GLMM. This is particularly easy to implement using the PQL method of section 3.4.2 (p. 149).

(a) Modify the function `gam.fit` from section 4.4, so that in place of the call to `am.fit` there is an appropriate call to `lme` to estimate the coefficients and smoothing parameters of a working linear mixed model. The modified function should take a response vector and the model matrices from the previous question as inputs, and return the `lme` fitted model object for the working model at convergence.

(b) Use your function to fit the Gamma additive model of section 4.4 to the `trees` data.

(c) Produce plots of measured volume against predicted volume, and of residuals against the linear predictor of the model.

(d) Plot the smooth effects with partial residuals.