Chapter 7

GAMs in Practice: mgcv

This chapter covers use of the generalized additive modelling functions provided by R package mgcv: the design of these functions is based largely on Hastie (1993), although to facilitate smoothing parameter estimation, their details have been modified. There are three main modelling functions: gam, already introduced in section 4.6 (p. 182); bam a version of gam designed for larger datasets; and gamm a version of gam for estimating generalized additive mixed models via package nlme, allowing access to a rich range of random effects and correlation structures.* In addition jagam provides an interface to JAGS for Bayesian stochastic simulation. Note that gam can be used with a wide range of distributions beyond the usual GLM exponential family. This generality is not available with bam and gamm. See section 7.12 for a brief overview of the other R packages available for smooth modelling.

When reading this chapter note that R and the mgcv package are subject to continuing efforts to improve them. Sometimes this may involve modifications of numerical optimization behaviour, which may result in noticeable, but (hopefully) statistically unimportant, differences between the output given in this chapter and the corresponding results with more recent versions. Sometimes the exact formatting of output can also change a little.

Familiarity with the mgcv help pages, covered in section 4.6.4 (p. 191), is likely to be helpful when reading this chapter.

7.1 Specifying smooths

For the most part, this chapter covers specification of models by example, but it is worth discussing some general points about the specification of smooths in model formulae up front. There are 4 types of smooth that can be used and mixed.

`s()` is used for univariate smooths (section 5.3, p. 201), isotropic smooths of several variables (section 5.5, 214) and random effects (section 3.5.2, 154).

`te()` is used to specify tensor product smooths constructed from any singly penalized marginal smooths usable with `s()`, according to section 5.6 (p. 227). Examples are provided in sections 7.2.3, 7.4 and 7.7.1, for example.

*gamm4 from the gamm4 package is similar, but using lmer and glmer from package lme4 as the fitting engines

`ti()` is used to specify tensor product interactions with the marginal smooths (and their lower order interactions) excluded, facilitating smooth ANOVA models as discussed in section 5.6.3 (p. 232), and exemplified in section 7.3.

`t2()` is used to specify the alternative tensor product smooth construction discussed in section 5.6.5 (p. 235), which is especially useful for generalized additive mixed modelling with the `gamm4` package described in section 7.7.

The first arguments to all these functions are the covariates of the smooth. Some further arguments control the details of the smoother. The most important are

`bs` is a short character string specifying the type of basis. e.g. `"cr"` for cubic regression spline, `"ds"` for Duchon spline, etc. It may be a vector in the tensor product case, if different types of basis are required for different marginals.

`k` is the basis dimension, or marginal basis dimension (tensor case). It can also be a vector in the tensor case, specifying a dimension for each marginal.

`m` specifies the order of basis and penalty, in a basis specific manner.

`id` labels the smooth. Smooths sharing a label all have the same smoothing parameter (assuming that they are of the same smoother type).

`by` is the name of a variable by which the smooth should be multiplied (metric case), or each level of which should have a separate copy of the smooth (factor case).

The last two items on the list require further explanation. An example formula with a smooth `id` is `y ~ s(x) + s(z1,id="foo") + s(z2,id="foo")`. This forces the smooths of `z1` and `z2` to have the same smoothing parameter: for this to really make sense they are also forced to have the same basis and penalty, provided this is possible.

`by` variables are the means for implementing 'varying coefficient models', such as that used in section 7.5.3. Suppose, for example, that we have metric variables x and z and want to specify a linear predictor term '$f(x_i)z_i$' where f is a smooth function. The model formula entry for this would be `s(x,by=z)`. Only one `by` variable is allowed per smooth, but any smooth with multiple covariates (specified by `s`, `te`, `ti` or `t2`) can also have a `by` variable. Note that, provided the `by` variable takes more than one value, such terms are identifiable without a sum-to-zero constraint, and so they are left unconstrained.

Metric `by` variables combined with a *summation convention* are the means by which linear functionals of smooths can be incorporated into the linear predictor. Examples are provided in sections 7.4.2 and 7.11.1. The idea is that if the covariates of the smooth and the `by` variable are all matrices, then a summation across rows is implied. For example if `X`, `Z` and `L` are all matrices then `s(X,Z,by=L)` specifies the term $\sum_k f(X_{ik}, Z_{ik})L_{ik}$ in the linear predictor. Tensor terms also support the convention.

`by` variables also facilitate 'smooth-factor' interactions, in which we have a separate smooth of one or more covariates for each level of a factor `by` variable. For example, suppose we have metric variables x and z and factor variable g with three levels. Let $g(i)$ denote the level of g corresponding to observation i. Then `te(z,x,by=g)` would contribute the terms '$f_{g(i)}(x_i, z_i)$' to the model linear pre-

dictor. That is, which of three separate smooth functions of x and z contributes to the linear predictor depends on which of the three levels of g applies for observation i. Again `s`, `te`, `ti` or `t2` terms all work in the same way regardless of the number of their covariates. To avoid confounding problems the smooths are all subject to sum to zero constraints, which usually means that the main effect of g should also be included in the model specification. For example, `g + te(z,x,by=g)`. Factor by variables can not be mixed with the summation convention.[†] When there are several factor `by` variables then identifiability can get tricky, and it can then be useful to employ *ordered factor* `by` variables. If a factor `by` is an ordered factor then no smooth is generated for its first level.

Often we would like all the smooths generated by a factor `by` to have the same smoothing parameter. The `id` mechanism allows this. For example `te(z,x,by=g,id="a")` causes the smooths for each level of `g` to share the same `id`, and hence all to have the same smoothing parameter.

7.1.1 How smooth specification works

As far as the `mgcv` estimation functions are concerned a smooth (or indeed a random effect) is just a set of model matrix columns and one or more corresponding penalties. This fact is exploited in the code design to make smooth implementation highly modular. Within the code, smooths are implemented via a smooth *constructor* method function. In addition each smooth has a *prediction matrix* method function, which will produce the matrix mapping the coefficients of the smooth to predictions at new covariate values. Finally, smooths can have a *plot* method function (although most use the generic smooth plot method).

The design uses the 'S3' object orientation built into R. `s()`, `te()`, `ti()` or `t2()` terms in a model formula give rise to a *smooth specification object*, with a class determining the type of smooth to be produced. This object is passed to the smooth constructor for its class (e.g., `smooth.construct.cr.smooth.spec` for the cubic regression spline basis `"cr"`). The constructor returns a *smooth object*, with its own class, containing a model matrix for the smooth, along with quadratic penalty matrices and other information. It also contains any information specific to the smooth that will be needed for prediction or plotting (knot locations, for example). See `?smooth.construct` for information on how to add a smooth class.

The smooth constructor and prediction matrix method functions are usually called via wrapper functions (`smoothCon()` and `PredictMat()`) that handle things like identifiability constraints, matrix arguments and `by` variables automatically for any smooth. Indeed it is quite easy to use `mgcv` only for the purpose of setting up smoothers and penalties. Here is an example setting up a thin plate regression spline basis and using it for unpenalized regression with `lm()`.

```
library(mgcv); library(MASS) ## load for mcycle data.
## set up a smoother...
```

[†] Separate summation convention smooths for each level of a factor are usually obtained by including a smooth for each level of the factor, with factor level specific versions of the `by` matrix set up to zero the term for all levels except the one to which the smooth applies.

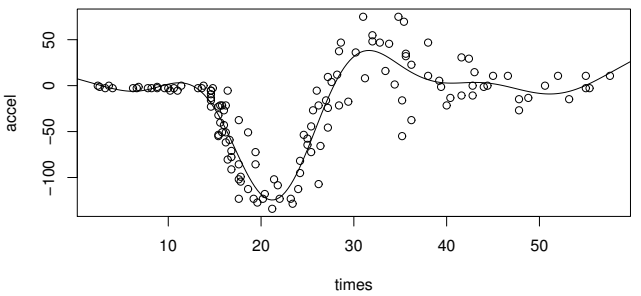

Figure 7.1 *Regression spline fit to the motorcycle data from the* MASS *library, illustrating how the* mgcv `smoothCon` *and* `PredictMat` *functions can be used to work with smoothers outside the* mgcv *modelling functions.*

```
sm <- smoothCon(s(times,k=10),data=mcycle,knots=NULL)[[1]]
## use it to fit a regression spline model...
beta <- coef(lm(mcycle$accel~sm$X-1))
with(mcycle,plot(times,accel)) ## plot data
times <- seq(0,60,length=200)  ## create prediction times
## Get matrix mapping beta to spline prediction at 'times'
Xp <- PredictMat(sm,data.frame(times=times))
lines(times,Xp%*%beta) ## add smooth to plot
```

The resulting plot is shown in figure 7.1.

7.2 Brain imaging example

Section 4.6 (p. 182) provided a basic introduction to mgcv. This section examines a more substantial example, in particular covering issues of model selection and checking in more detail. The data are from brain imaging by functional magnetic resonance scanning, and were reported in Landau et al. (2003). The data are available in data frame `brain`, and are shown in figure 7.2. Each row of the data frame corresponds to one voxel. The columns are: `X` and `Y`, giving the locations of each voxel; `medFPQ`, the brain activity level measurement (median 'Fundamental Power Quotient' over three measurements); `region`, a code indicating which region the voxel belongs to (0 for 'base' region; 1 for region of experimental interest and 2 for region subjected to direct stimulation) — there are some `NA`'s in this column; `meanTheta` is the average phase shift at each voxel, which we will not consider further. This section will consider models for `medFPQ` as a function of `X` and `Y`.

Clearly the `medFPQ` data are quite noisy, and the main purpose of the modelling in this section is simply to partition this very variable signal into a smooth trend component and a 'random variability' component, so that the pattern in the image becomes a bit clearer. For data such as these, where the discretization (into voxels) is

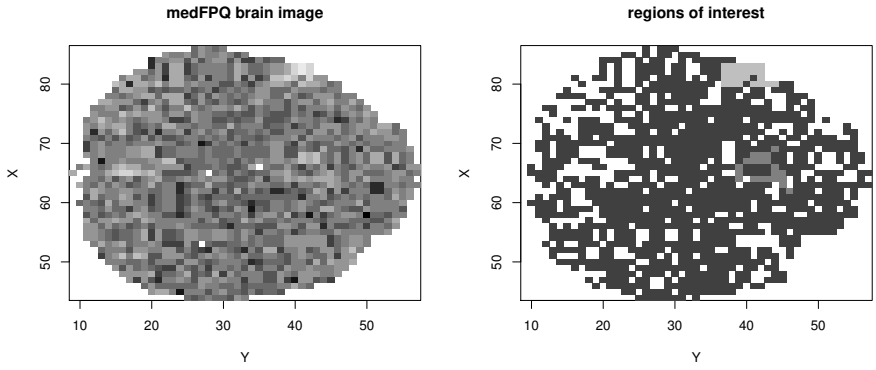

Figure 7.2 *The raw data for the brain imaging data discussed in section 7.2. The left hand plot shows the median of 3 measurements of Fundamental Power Quotient values at each voxel making up this slice of the scan: these are the measurements of brain activity. The right hand panel shows the regions of interest: the 'base region' is the darkest shade, the region directly stimulated experimentally is shown in light grey and the region of experimental interest is shown as dark grey. Unclassified voxels are white.*

essentially arbitrary, there is clearly a case to be made for employing a model which includes local correlation in the 'error' terms. In principle the correlation structures available in `gamm` could be used for this purpose (see, e.g., section 7.7.2, p. 371), but here we will proceed by treating the randomness as being independent between voxels, and let all between voxel correlation be modelled by the trend. This is not simply a matter of convenience, but relates closely to the purpose of the analysis: for these data the main interest lies in cleaning up this particular image: that is, in removing the component of variability that appears to be nothing more than random variation, at the level of the individual voxel. The fact that the underlying mechanism, generating features in the image, may include a component attributable to correlated noise across voxels is only something that need be built into the model if, for example, the objective is to be able to remove this component of the pattern from the image.

7.2.1 Preliminary modelling

An appropriate initial model structure would probably involve modelling `medFPQ` as a smooth function of `X` and `Y`, but before attempting to fit models, it is worth examining the data itself to look for possible problems. When this is done, 2 voxels appear problematic. These voxels have `medFPQ` values recorded as 3×10^{-6} and 4×10^{-7}, while the remaining 1565 voxels have values in the range 0.003 to 20. Residual plots from all attempts to model the data set including these two voxels consistently show them as grotesque outliers. Therefore they were excluded from the following analysis:

```
brain <- brain[brain$medFPQ>5e-3,] # exclude 2 outliers
```

The fairly skewed nature of the response data, medFPQ, and the fact that it is a necessarily positive quantity, suggest that some transformation may be required if a Gaussian error model is to be used. Attempting to use a Gaussian model without transformation confirms this:

```
> m0 <- gam(medFPQ ~ s(Y,X,k=100), data=brain)
> gam.check(m0)
Method: GCV   Optimizer: magic
Smoothing parameter selection converged after 6 iterations.
The RMS GCV score gradient at convergence was 6.236018e-05 .
The Hessian was positive definite.
Model rank =  100 / 100

Basis dim. (k) checking results. Low p-value (k-index<1) may
indicate that k is too low, especially if edf is close to k'.

          k'    edf k-index p-value
s(Y,X) 99.000 86.782   0.863       0
```

gam.check is a routine that produces some basic residual plots, and a little further information about the success or otherwise of the fitting process. Here it reports successful convergence of the GCV optimization and that the model has full rank. An informal basis dimension check is also included based on section 5.9 (p. 242): k' gives the maximum possible EDF for the smooth, to compare with the actual EDF; the k-index is the ratio of the model estimated scale parameter to an estimate based on differencing neighbouring residuals ('neighbouring' according to the arguments of the smooth); the p-value is for the residual randomization test described in section 5.9. Here the results are problematic, but of secondary importance, given the plots shown in figure 7.3. As explained in the figure caption, there are clear problems with the constant variance assumption: the variance is increasing with the mean. From the plots it is not easy to gauge the rate at which the variance of the data is increasing with the mean, but, in the absence of a good physical model of the mechanism underlying the relationship, some guidance can be obtained by a simple informal approach. If we assume that $\mathrm{var}(y_i) \propto \mu_i^\beta$, where $\mu_i = \mathbb{E}(y_i)$ and β is some parameter, then a simple regression estimate of β can be obtained as follows:

```
> e <- residuals(m0); fv <- fitted(m0)
> lm(log(e^2) ~ log(fv))

Call:
lm(formula = log(e^2) ~ log(fv))

Coefficients:
(Intercept)      log(fv)
     -1.961        1.912
```

i.e., $\beta \approx 2$. That is, from the residuals of the simple fit, it appears that the variance of the data increases with the square of the mean. This in turn suggests using the gamma distribution, which has this mean-variance relationship (see table 3.1, p. 104). Alternatively the apparent mean-variance relationship suggests using a log transform of medFPQ to stabilize the variance, but in practice a less extreme 4^{th} root transform

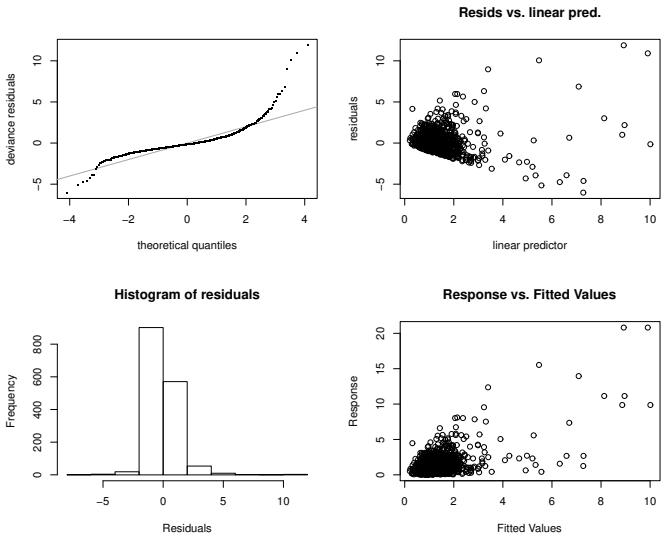

Figure 7.3 *Some basic model checking plots for model* `m0` *fitted to the brain scan data. The upper left normal QQ-plot clearly shows a problem with the Gaussian assumption. This is unsurprising when the upper right plot of residuals versus fitted values (linear predictor) is examined: the constant variance assumption is clearly untenable. The lower left histogram of residuals confirms the pattern evident in the QQ-plot: there are too many residuals in the tails and centre of the distribution relative to its shoulders. In this case the plot of response variable against fitted values, shown in the lower right panel, emphasizes the failure of the constant variance assumption.*

produces better residual plots. With the gamma model it might also be appropriate to use a log link, in order to ensure that all model predicted FPQ values are positive.

The following fits models based first on transforming the data and then on use of the gamma distribution.

```
m1 <- gam(medFPQ^.25 ~ s(Y,X,k=100), data=brain)
gam.check(m1)
m2<-gam(medFPQ~s(Y,X,k=100),data=brain,family=Gamma(link=log))
```

The plots from `gam.check` are shown in figure 7.4, and now show nothing problematic. `gam.check` plots for `m2` are equally good (but not shown). The informal basis dimension test still gives a low p-value, but since the EDF is some way below the basis dimension, and increasing the basis dimension barely changes the fitted values, let's stick with `k=100` here.

The major difference between `m1` and `m2` is in their biasedness on different scales. The model of the transformed data is approximately unbiased on the 4th root of the response scale: this means that it is biased downwards on the response scale itself. The log-gamma model is approximately unbiased on the response scale (only approximately because maximum penalized likelihood estimation is not generally unbiased, but is consistent). This can be seen if we look at the means of the fitted

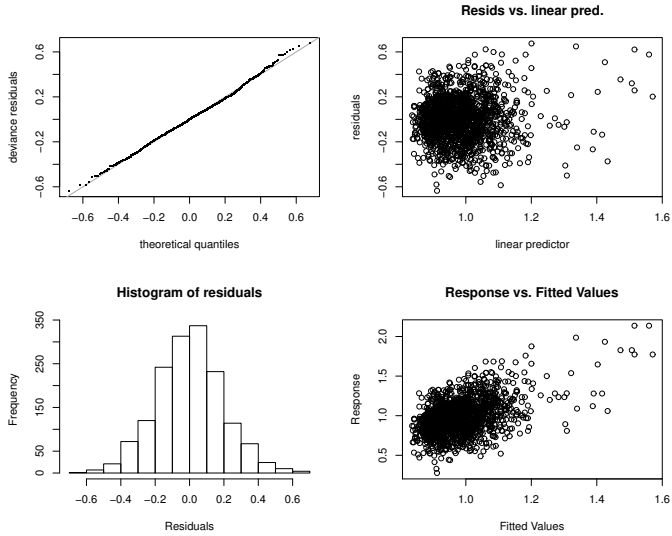

Figure 7.4 *Some basic model checking plots for model* `m1` *fitted to the transformed brain scan data. The upper left normal QQ-plot is very close to a straight line, suggesting that the distributional assumption is reasonable. The upper right plot suggests that variance is approximately constant as the mean increases. The histogram of residuals at lower left appears approximately consistent with normality. The lower right plot, of response against fitted values, shows a positive linear relationship with a good deal of scatter: nothing problematic.*

values (response scale) for the two models, and compare these to the mean of the raw data:

```
> mean(fitted(m1)^4);mean(fitted(m2));mean(brain$medFPQ)
[1] 0.985554   # m1 tends to under-estimate
[1] 1.211483   # m2 substantially better
[1] 1.250302
```

Clearly, if the response scale is the scale of prime interest then the gamma model is to be preferred to the model based on normality of the transformed data. So far the best model seems to be the gamma log-link model, `m2`, which should be examined a little further.

```
> m2

Family: Gamma
Link function: log

Formula:
medFPQ ~ s(Y, X, k = 100)

Estimated degrees of freedom:
60.6  total = 61.61
```

```
GCV score: 0.6216871
> vis.gam(m2,plot.type="contour",too.far=0.03,
+ color="gray",n.grid=60,zlim=c(-1,2))
```

A relatively complex fitted surface has been estimated, with 62 degrees of freedom. The function `vis.gam` provides quite useful facilities for plotting predictions from a `gam` fit against pairs of covariates, either as coloured perspective plots, or as coloured (or grey scale) contour plots. The plot it produces for `m2` is shown in figure 7.5(a). Notice how the activity in the directly stimulated region and the region of interest stand out clearly in this plot.

7.2.2 Would an additive structure be better?

Given the large number of degrees of freedom employed by the model `m2`, the question naturally arises of whether a different simpler model structure might achieve a more parsimonious fit. Since this is a book about GAMs, the obvious candidate is the additive model,

$$\log\{\mathbb{E}(\texttt{medFPQ}_i)\} = f_1(\texttt{Y}_i) + f_2(\texttt{X}_i), \quad \texttt{medFPQ}_i \sim \text{gamma}.$$

```
> m3 <- gam(medFPQ ~ s(Y,k=30) + s(X,k=30), data=brain,
+           family=Gamma(link=log))
> m3

Family: Gamma
Link function: log

Formula:
medFPQ ~ s(Y, k = 30) + s(X, k = 30)

Estimated degrees of freedom:
 9.58 20.20  total = 30.77

GCV score: 0.6453502
```

The GCV score is higher for this model, suggesting that it is not an improvement, and a comparison of explained deviances using `summary(m2)` and `summary(m3)` also suggests that the additive model is substantially worse. AIC comparison confirms this:

```
> AIC(m2,m3)
          df      AIC
m2 62.61062 3321.681 ## bivariate smooth
m3 31.77467 3393.738 ## additive model
```

Perhaps the most persuasive argument against the additive structure is provided by the plot of the predicted log activity levels provided in figure 7.5(b): the additive structure produces horizontal and vertical 'stripes' in the plot that have no real support from the data.

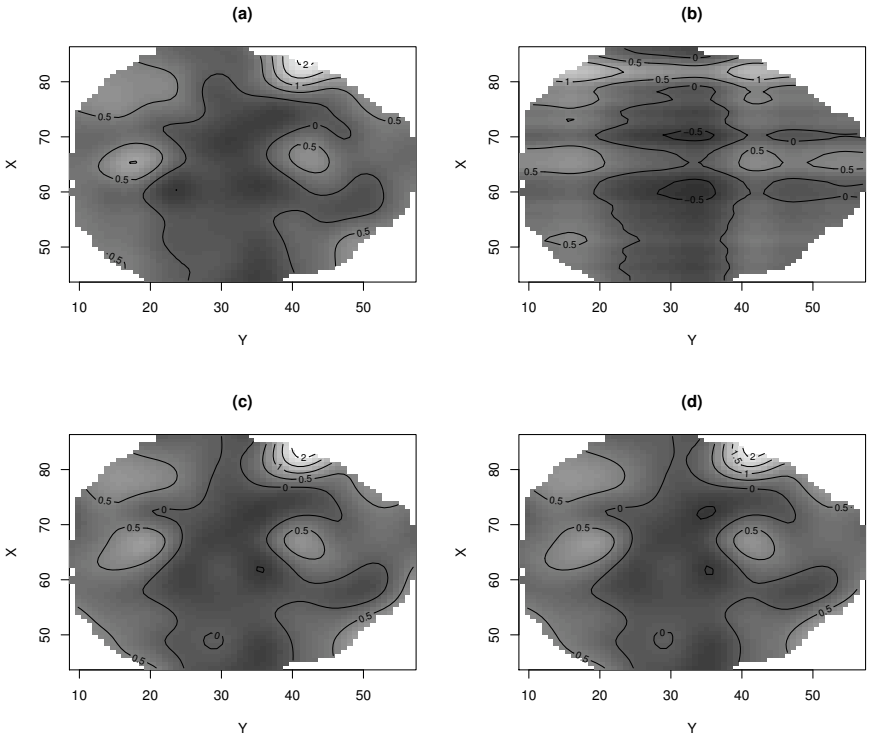

Figure 7.5 *Comparison of 4 models of the brain scan data. All plots show image plots and overlaid contour plots, on the scale of the linear predictor. (a) is model* `m2` *based on an isotropic smooth of* `Y` *and* `X`*. (b) is model* `m3` *based on a sum of smooth functions of* `Y` *and* `X`*. Notice the apparent artefacts in (b), relating to the assumption of an additive structure. (c) plots model* `tm` *and is basically as (a), but using a rank 100 tensor product smooth in place of the isotropic smooth. (d) is for model* `tm1`*, which is as model* `tm` *but with main effects and interaction estimated separately. All plots were produced with something like:* `vis.gam(m2,plot.type="contour",too.far=0.03,color="gray", n.grid=60,zlim=c(-1,2),main="(a)")`

7.2.3 Isotropic or tensor product smooths?

As discussed in chapter 5, isotropic smooths, of the sort produced by `s(Y,X)` terms, are usually good choices when the covariates of the smooth are naturally on the same scale, and we expect that the same degree of smoothness is appropriate with respect to both covariate axes. For the brain scan data these conditions probably hold, so the isotropic smooths are probably a good choice.

Nevertheless, it is worth checking what the results look like if we use a scale invariant tensor product smooth of `Y` and `X`, in place of the isotropic smooth (see sec-

tion 5.6, p. 227). Such smooths are computationally rather efficient (if the marginal bases of a tensor product smooth are cheap to construct, then so is the tensor product basis itself). In addition they provide another way of comparing an additive model to a bivariate smooth model via the `ti` interaction term construction of section 5.6.3 (p. 232). In particular we can fit the model

$$\log\{\mathbb{E}(\texttt{medFPQ}_i)\} = f_1(\texttt{Y}_i) + f_2(\texttt{X}_i) + f_3(\texttt{X}_i, \texttt{Y}_i), \quad \texttt{medFPQ}_i \sim \text{gamma},$$

in which there are main smooth effects and an interaction smooth effect, constructed so that functions of the form $f_1(\texttt{Y}_i) + f_2(\texttt{X}_i)$ are excluded from its basis.

Here is the code, first for the single tensor product smooth and then for the additive plus interaction version:

```
tm<-gam(medFPQ~te(Y,X,k=10),data=brain,family=Gamma(link=log))
tm1 <- gam(medFPQ ~ s(Y,k=10,bs="cr") + s(X,bs="cr",k=10) +
           ti(X,Y,k=10), data=brain, family=Gamma(link=log))
```

The single `s` terms could have been replaced by terms `ti(X,k=10)` resulting in the same model fit. Plots from `tm` and `tm1` are shown in figure 7.5(c) and (d), and residual plots and other checks show that both models are reasonable. However, an AIC comparison suggests that neither tensor product structure is quite as good as the isotropic model for these data:

```
> AIC(m2,tm,tm1)
          df      AIC
m2  62.61062 3321.681 ## isotropic
tm  60.34768 3332.817 ## tensor product
tm1 57.08366 3330.187 ## main + interaction
```

The AIC comparison slightly favours the main effects plus interaction model over the single tensor product smooth. One initially puzzling question is why the AICs are not identical for `tm` and `tm1`, given that they have the same smoothing basis, but simply partitioned differently. The reason is that they do not have the same penalty structure: `tm1` has extra separate penalties on the part of the basis representing the smooth main effects, $f_1(\texttt{Y}_i) + f_2(\texttt{X}_i)$.

Returning to the question of whether an additive model structure or bivariate structure is more appropriate, `tm1` can be examined using `summary` or `anova`.

```
> anova(tm1)

Family: Gamma
Link function: log

Formula:
medFPQ ~ s(Y, k = 10, bs = "cr") + s(X, bs = "cr", k = 10) +
    ti(X, Y, k = 10)

Approximate significance of smooth terms:
           edf Ref.df      F  p-value
s(Y)     8.258  8.750 14.526  < 2e-16
s(X)     7.494  8.314  6.959 2.37e-09
ti(X,Y) 39.332 49.891  2.291 1.15e-06
```

The p-value for `ti(X,Y)` (computed as in section 6.12.1, p. 305) clearly suggests that the smooth term is significantly non zero: the interaction is needed and an additive structure alone is insufficient.

7.2.4 Detecting symmetry (with `by` *variables)*

It is sometimes of interest to test whether the image underlying some noisy data is symmetric, and this is quite straightforward to accomplish, with the methods covered here. For the brain data we might want to test whether the underlying activity levels are symmetric about the mean `X` value of 64.5. The symmetry model in this case would be

$$\log\{\mathbb{E}(\texttt{medFPQ}_i)\} = f(\texttt{Y}_i, |\texttt{X}_i - 64.5|), \quad \texttt{medFPQ}_i \sim \text{ gamma}$$

and this could be compared with model `m2`, which does not assume symmetry, although for strict nesting of models we would need to be slightly more sophisticated and use an asymmetry model with a form such as

$$\log\{\mathbb{E}(\texttt{medFPQ}_i)\} = f(\texttt{Y}_i, |\texttt{X}_i - 64.5|) + f_r(\texttt{Y}_i, |\texttt{X}_i - 64.5|).\texttt{right}_i,$$

where f_r is represented using the same basis as f, and $\texttt{right}_i$ is a dummy variable, taking the values 1 or 0, depending on whether $\texttt{medFPQ}_i$ is from the right or left side of the brain.

The following code creates variables required to fit these two models, estimates them, and prints the results.

```
> brain$Xc <- abs(brain$X - 64.5)
> brain$right <- as.numeric(brain$X<64.5)
> m.sy <- gam(medFPQ~s(Y,Xc,k=100),data=brain,
+             family=Gamma(link=log))
> m.as <- gam(medFPQ~s(Y,Xc,k=100)+s(Y,Xc,k=100,by=right),
+             data=brain,family=Gamma(link=log))
> m.sy
[edited]
Estimated degrees of freedom:
51.4  total = 52.44
GCV score: 0.6489799
> m.as
[edited]
Estimated degrees of freedom:
50.5 44.7  total = 96.2
GCV score: 0.6176281
```

The GCV scores suggest that the asymmetric model is better, and the asymmetric model AIC is also 97 lower than the symmetric version. The same question can also be addressed by hypothesis testing:

```
> anova(m.as)
Family: Gamma
Link function: log
```

```
Formula:
medFPQ ~ s(Y, Xc, k = 100) + s(Y, Xc, k = 100, by = right)


Approximate significance of smooth terms:
                edf Ref.df     F  p-value
s(Y,Xc)       50.48  65.99 4.344  < 2e-16
s(Y,Xc):right 44.72  59.21 2.457 1.06e-08
```

If the symmetric model was adequate then the `s(Y,Xc):right` term should not be significantly different from zero, which is far from being the case here. Again this test uses section 6.12.1 (p. 305). The less well-justified approach of section 6.12.4 (p. 313) could be accessed using `anova(m.sy,m.as)`.

Plots of the different model predictions help to show why symmetry is so clearly rejected (figure 7.6).

```
vis.gam(m.sy,plot.type="contour",view=c("Xc","Y"),too.far=.03,
        color="gray",n.grid=60,zlim=c(-1,2),main="both sides")
vis.gam(m.as,plot.type="contour",view=c("Xc","Y"),
        cond=list(right=0),too.far=.03,color="gray",n.grid=60,
        zlim=c(-1,2),main="left side")
vis.gam(m.as,plot.type="contour",view=c("Xc","Y"),
        cond=list(right=1),too.far=.03,color="gray",n.grid=60,
        zlim=c(-1,2),main="right side")
```

7.2.5 Comparing two surfaces

The symmetry testing, considered in the last section, is an example of comparing surfaces — in that case one half of an image with a mirror image of the other half. In some circumstances it is also interesting to compare completely independent surfaces in a similar way. To see how this might work, a second set of brain scan data can be simulated, using the fitted model `m2`, and somewhat perturbed, as follows.

```
brain1 <- brain
mu <- fitted(m2)
n<-length(mu)
ind <- brain1$X<60 & brain1$Y<20
mu[ind] <- mu[ind]/3
set.seed(1)
brain1$medFPQ <- rgamma(rep(1,n),mu/m2$sig2,scale=m2$sig2)
```

Now the data sets can be combined, and dummy variables created to identify which data set each row of the combined data frame relates to.

```
brain2=rbind(brain,brain1)
brain2$sample1 <- c(rep(1,n),rep(0,n))
brain2$sample0 <- 1 - brain2$sample1
```

After which it is straightforward to fit a model with a single combined surface for both data sets, and a second model where the surfaces are allowed to differ. Note that in the latter case a single common surface is estimated, with a difference surface

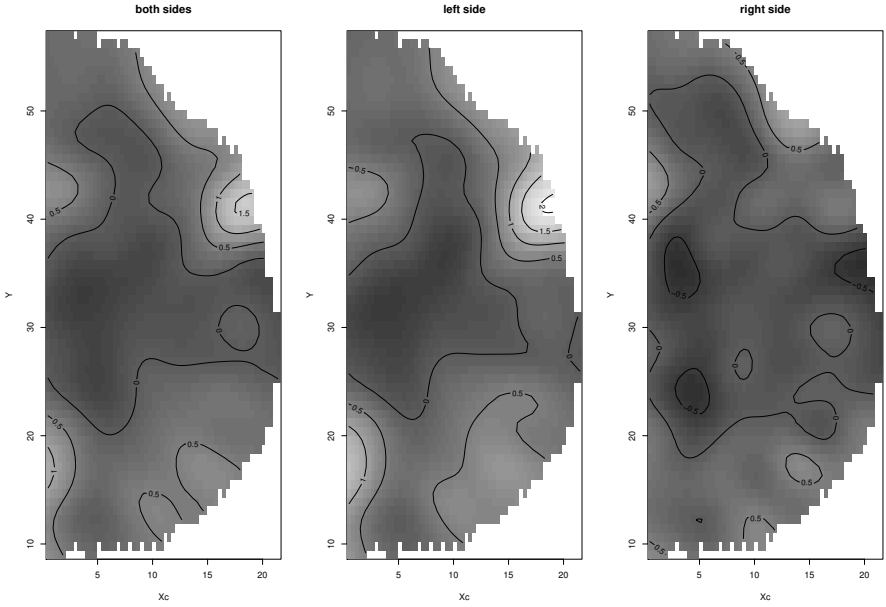

Figure 7.6 *Half brain images for the two models involved in examining possible symmetry in the brain image data. The left panel shows the predictions of log activity for the symmetry model: the left side of the image would be a mirror image of this plot. The middle plot shows a mirror image of the left side brain image under the asymmetry model, while the right plot shows the right side image under the asymmetry model. Clearly the asymmetry model suggests rather different activity levels in the two sides of the image.*

for the second sample. This approach is often preferable to a model with two completely separate surfaces, for reasons of parsimony. If there is no difference between the surfaces, it will still be necessary to estimate two quite complex surfaces, if we adopt separate surfaces for each data set. With the common surface plus difference approach, only one complex surface need be estimated: the difference being close to flat. Similarly, any time that the surfaces are likely to differ in a fairly simple way, it makes sense to use the common surface plus difference model, thereby reducing the number of effective degrees of freedom needed by the model.

```
m.same <- gam(medFPQ~s(Y,X,k=100),data=brain2,
            family=Gamma(link=log))
m.diff <- gam(medFPQ~s(Y,X,k=100)+s(Y,X,by=sample1,k=100),
            data=brain2,family=Gamma(link=log))
```

Examination of the GCV scores for the two models suggests that the second model `m.diff` is preferable to `m.same`. An AIC comparison, or a test of the significance of the difference surface in `mdiff`, confirm this:

```
> AIC(m.same,m.diff)
             df      AIC
m.same 65.09955 6529.403
m.diff 80.84258 6496.320
> anova(m.diff)

Family: Gamma
Link function: log

Formula:
medFPQ ~ s(Y, X, k = 100) + s(Y, X, by = sample1, k = 100)

Approximate significance of smooth terms:
                edf Ref.df     F  p-value
s(Y,X)        62.46  79.28 6.323  < 2e-16
s(Y,X):sample1 16.38  22.91 2.190 0.000936
```

In this example, the data for the two surfaces were available on exactly the same regular mesh, but the approach is equally applicable for irregular data where the data are not at the same covariate values for the two surfaces (although, of course, the regions of covariate space covered should overlap, for a comparison of this sort to be meaningful).

7.2.6 Prediction with `predict.gam`

The `predict` method function, `predict.gam`, enables a `gam` fitted model object to be used for prediction at new values of the model covariates. It is also used to provide estimates of the uncertainty of those predictions, and the user can specify whether predictions should be made on the scale of the response or of the linear predictor. For predictions on the scale of the linear predictor, `predict.gam` also allows predictions to be decomposed into their component terms, and it is also possible to extract the matrix, which, when multiplied by the model parameter vector, yields the vector of predictions at the given set of covariate values.

Usually the covariate values at which predictions are required are supplied as a data frame in argument `newdata`, but if this argument is not supplied then predictions/fitted values are returned for the original covariate values, used for model estimation. Here are some examples. Firstly on the scale of the linear predictor,

```
> predict(m2)[1:5]
        1         2         3         5         6
0.2988590 0.3353680 0.3432145 0.2786153 0.4482840
> pv <- predict(m2,se=TRUE)
> pv$fit[1:5]
        1         2         3         5         6
0.2988590 0.3353680 0.3432145 0.2786153 0.4482840
> pv$se[1:5]
        1         2         3         5         6
0.2582929 0.2124696 0.2120118 0.2193975 0.2234603
```

and then on the response scale,

```
> predict(m2,type="response")[1:5]
       1        2        3        5        6
1.348320 1.398455 1.409471 1.321299 1.565623
> pv <- predict(m2,type="response",se=TRUE)
> pv$se[1:5]
        1         2         3         5         6
0.3482614 0.2971292 0.2988245 0.2898897 0.3498546
```

For both sets of examples, predictions are produced for all 1564 voxels, but I have only printed the first 5. Note that the standard errors provided on the response scale are approximate, being obtained by the usual Taylor expansion approach.

Usually, predictions are required for new covariate values, rather than simply for the values used in fitting. Suppose, for example, that we are interested in two points in the brain scan image, firstly in the directly stimulated area, ($X = 80.1, Y = 41.8$), and secondly in the region of experimental interest, ($X = 68.3, Y = 41.8$). A data frame can be created, containing the covariate values for which predictions are required, and this can be passed to `predict.gam`, as the following couple of examples show.

```
> pd <- data.frame(X=c(80.1,68.3),Y=c(41.8,41.8))
> predict(m2,newdata=pd)
        1         2
1.2788054 0.5954536
> predict(m2,newdata=pd,type="response",se=TRUE)
$fit
       1        2
3.592346 1.813854

$se.fit
        1         2
0.5287998 0.2600005
```

It is also possible to obtain the contributions that each model term, excluding the intercept, makes to the linear predictor, as the following example shows. The additive model `m3` has been used to illustrate this, since it has more than one term.

```
> predict(m3,newdata=pd,type="terms",se=TRUE)
$fit
       s(Y)          s(X)
1 0.2501423  0.518684282
2 0.2501423 -0.003159924

$se.fit
        s(Y)       s(X)
1 0.05856406 0.09826015
2 0.05856406 0.08222815

attr(,"constant")
(Intercept)
   0.143576
```

As you can see, named arrays have been returned, the columns of which correspond

to each model term. There is one array for the predicted values, and a second for the corresponding standard errors.

Prediction with `lpmatrix`

Because the GAMs discussed in this book have an underlying parametric representation, it is possible to obtain a 'prediction matrix', $\mathbf{X}_p$, which maps the model parameters, $\hat{\boldsymbol{\beta}}$, to the predictions of the linear predictor, $\hat{\boldsymbol{\eta}}_p$. That is, to find $\mathbf{X}_p$ such that

$$\hat{\boldsymbol{\eta}}_p = \mathbf{X}_p \hat{\boldsymbol{\beta}}.$$

`predict.gam` can return $\mathbf{X}_p$, if its `type` argument is set to `"lpmatrix"`.

The following example illustrates this. Since the returned matrix is of dimension 2×100 (with default treatment of identifiability constraints), it has not been printed out, but rather it is demonstrated that it does indeed give the required linear predictor values, when multiplied by the coefficients of the fitted model.

```
> Xp <- predict(m2,newdata=pd,type="lpmatrix")
> fv <- Xp %*% coef(m2)
> fv
       [,1]
1 1.2788054
2 0.5954536
```

Why is $\mathbf{X}_p$ useful? A major use is in the calculation of variances for combinations of linear predictor values. Clearly, if $\hat{\mathbf{V}}_{\boldsymbol{\beta}}$ is the estimate of the parameter covariance matrix, then from standard probability theory, the estimated covariance matrix of $\boldsymbol{\eta}_p$ must be:

$$\hat{\mathbf{V}}_{\boldsymbol{\eta}_p} = \mathbf{X}_p \hat{\mathbf{V}}_{\boldsymbol{\beta}} \mathbf{X}_p^{\mathsf{T}}.$$

Now suppose that we are really interested in, for example, the difference, δ, between the linear predictor values at the points in the two regions. This difference could be written as, $\delta = \mathbf{d}^{\mathsf{T}} \boldsymbol{\eta}_p$, where $\mathbf{d}^{\mathsf{T}} = [1, -1]^{\mathsf{T}}$. In that case, standard theory says that:

$$\widehat{\mathrm{var}(\delta)} = \mathbf{d}^{\mathsf{T}} \hat{\mathbf{V}}_{\boldsymbol{\eta}_p} \mathbf{d} = \mathbf{d}^{\mathsf{T}} \mathbf{X}_p \hat{\mathbf{V}}_{\boldsymbol{\beta}} \mathbf{X}_p^{\mathsf{T}} \mathbf{d}.$$

The following code illustrates this.

```
> d <- t(c(1,-1))
> d %*% fv
          [,1]
[1,] 0.6833517
> d %*% Xp %*% vcov(m2) %*% t(Xp) %*% t(d)
           [,1]
[1,] 0.04184623
```

So, the ability to obtain an explicit 'predictor matrix' makes some variance calculations rather straightforward. Notice that in general some care must be taken with calculations like the preceding one, if `Xp` has a large number of rows. For example, if `Xp` had 10,000 rows and `d` was some corresponding weight vector then it is important to ensure that the calculation does not involve the explicit formation of a $10,000 \times 10,000$ matrix, `Xp%*%vcov(m2)%*%t(Xp)`,

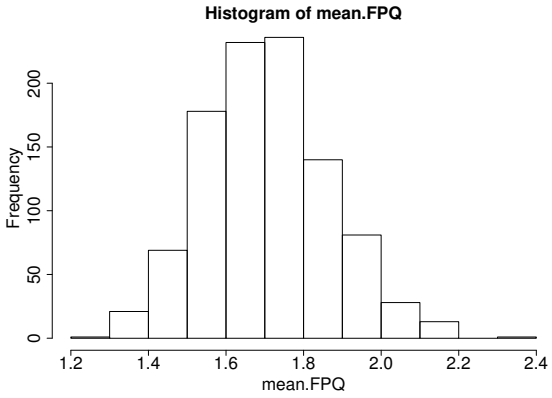

Figure 7.7 *Histogram of 1000 draws from the posterior distribution of average* `medFPQ` *in the region of experimental interest in the brain scan example.*

which would be costly in both computer time and memory. Fortunately, careful use of brackets will force R to order the operations in a way that avoids this: `(d%*%Xp)%*%vcov(m2)%*%(t(Xp)%*%t(d))` would do the trick.

A further trick along these lines is useful. Quite often we only need the diagonal of some covariance matrix, such as `diag(Xp%*%Vp%*%t(Xp))`. It turns out that this is the same as `rowSums(Xp*(Vp%*%Xp))`. The latter expression is much more efficient than the former as it does not compute the off-diagonal elements that are not of interest.

7.2.7 Variances of non-linear functions of the fitted model

The linear theory facilitating the calculations of the previous section is only applicable if we are interested in inference about linear functions of the linear predictor. As soon as the functions of interest become non-linear, as is generally the case when working on the response scale, we need different methods.

In fact, prediction matrices, in conjunction with simulation from the posterior distribution of the parameters, $\boldsymbol{\beta}$, give a simple and general method for obtaining variance estimates (or indeed distribution estimates), for any quantity derived from the fitted model, not simply those quantities that are linear in $\boldsymbol{\beta}$. The idea of this sort of posterior simulation was discussed in section 6.10 (p. 293).

As an example, suppose that we would like to estimate the posterior distribution of the average `medFPQ`, in the region of experimental interest. Since the quantity of interest is on the response scale, and not the scale of the linear predictor, it is not linear in $\boldsymbol{\beta}$ and the previous section's method is not applicable.

To approach this problem, a prediction matrix is first obtained, which maps the parameters to the values of the linear predictor needed to form the average.

```
ind <- brain$region==1& ! is.na(brain$region)
Xp <- predict(m2,newdata=brain[ind,],type="lpmatrix")
```

Next, a large number of replicate parameter sets are simulated from the posterior distribution of $\boldsymbol{\beta}$, using the `rmvn` function.

```
set.seed(8) ## for repeatability
br <- rmvn(n=1000,coef(m2),vcov(m2)) # simulate from posterior
```

Each column of the matrix `br` is a replicate parameter vector, drawn from the approximate posterior distribution of $\boldsymbol{\beta}$. Given these replicate parameter vectors and the matrix `Xp`, it is a simple matter to obtain the linear predictor implied by each replicate, from which the required averages can easily be obtained:

```
mean.FPQ <- rep(0,1000)
for (i in 1:1000) {
  lp <- Xp %*% br[i,]  # replicate linear predictor
  mean.FPQ[i] <- mean(exp(lp)) # replicate region 1 mean FPQ
}
```

Or, more efficiently, but less readably:

```
mean.FPQ <- colMeans(exp(Xp %*% t(br)))
```

So `mean.FPQ` now contains 1000 replicates from the approximate posterior distribution of the mean FPQ measurement in region 1. The results of `hist(mean.FPQ)` are shown in figure 7.7.

Clearly, this simulation approach is rather general: it is easy to obtain samples from the posterior distribution of any quantity that can be predicted from the fitted model. Notice also that, in comparison to bootstrapping, the approach is very computationally efficient. One disadvantage is that the smoothing parameters are treated as fixed at their estimates, rather than as uncertain. A way around this is to use the smoothing parameter uncertainty correction of section 6.11.1 (p. 302), which is available via `vcov(m,unconditional=TRUE)` provided that `m` is estimated using ML or REML smoothing parameter estimation.

7.3 A smooth ANOVA model for diabetic retinopathy

The `wesdr` data frame (originally from Chong Gu's `gss` package) contains clinical data on whether diabetic patients suffer from diabetic retinopathy or not. Three possible predictors are also available: duration of disease (years), body mass index (mass in kg divided by square of height in metres) and percent glycosylated hemoglobin in the blood. The latter is the percentage of hemoglobin bound to glucose, which reflects long term average blood glucose levels and should be below 6% for non-diabetics. Figure 7.8 shows boxplots of the covariates by disease status. The relationship of risk to the covariates is somewhat unclear.

To try and understand the data, a smooth additive logistic regression model could be employed. Since we might expect to find interactions between the covariates it might make sense to employ the sort of smooth ANOVA model introduced in section

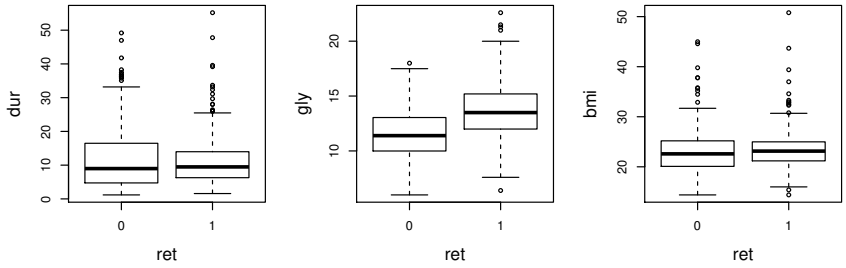

Figure 7.8 *Boxplots of disease duration, percent glycosylated hemoglobin and body mass index, by retinopathy status.*

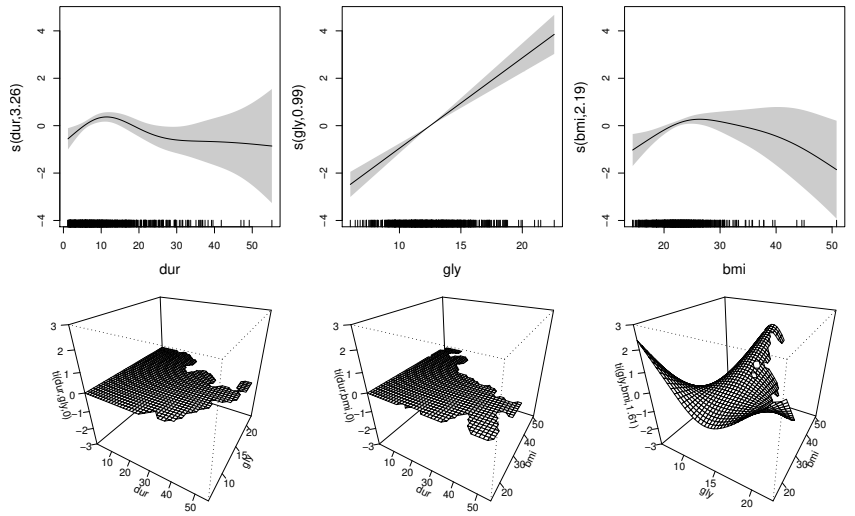

Figure 7.9 *Estimated effects for the smooth ANOVA logistic regression model for retinopathy risk. Notice how only the* `gly`-`bmi` *interaction is estimated to be non-zero.*

5.6.3 (p. 232). For example $\texttt{ret}_i \sim \text{binom}(1, \mu_i)$,

$$\text{logit}(\mu_i) = f_1(\texttt{dur}_i) + f_2(\texttt{gly}_i) + f_3(\texttt{bmi}_i) + f_4(\texttt{dur}_i, \texttt{gly}_i) + f_5(\texttt{dur}_i, \texttt{bmi}_i) + f_6(\texttt{bmi}_i, \texttt{gly}_i)$$

where f_1 to f_3 are smooth 'main effects', while f_4 to f_6 are smooth 'interactions' constructed to exclude the main effects, as described in section 5.6.3. The model is easily estimated, and for model selection the null space penalties of section 5.4.3 (p. 214) can be used.

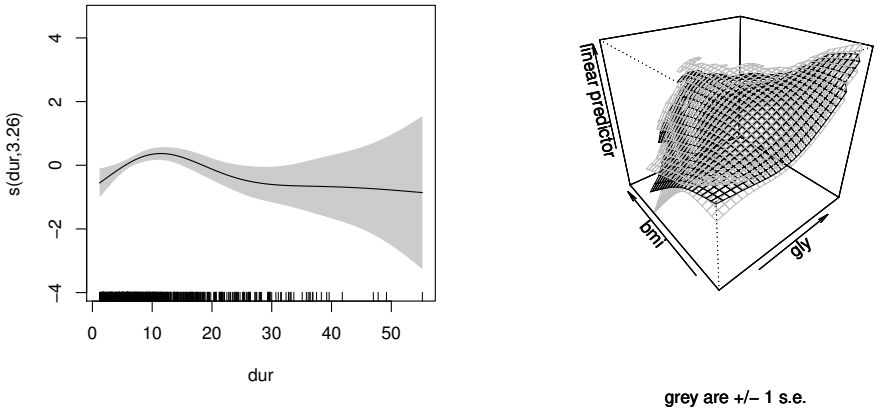

Figure 7.10 *Estimated duration effect and combined effect of body mass index and percent glycosylated hemoglobin from the retinopathy model. On the right, the grey surfaces are at plus or minus one standard error from the estimate. Notice how a higher level of glycosylated hemoglobin seems to be tolerated when body mass index is low.*

```
> b <- gam(ret ~ s(dur,k=k) + s(gly,k=k) + s(bmi,k=k) +
+          ti(dur,gly,k=k) + ti(dur,bmi,k=k) + ti(gly,bmi,k=k),
+      select=TRUE, data=wesdr, family=binomial(), method="ML")
> b
Family: binomial
Link function: logit

Formula:
ret ~ s(dur, k = k) + s(gly, k = k) + s(bmi, k = k) + ti(dur,
    gly, k = k) + ti(dur, bmi, k = k) + ti(gly, bmi, k = k)

Estimated degrees of freedom:
3.2553 0.9892 2.1866 0.0003 0.0001 1.6118  total = 9.04

ML score: 385.7904
```

Notice how f_4 and f_5 are effectively penalized out of the model, as the plots of the effects in figure 7.9 also show. f_6 is estimated to be non-zero and the model `summary` confirms that it is significantly so. Hence we need to interpret the main effects and interaction for `bmi` and `gly` together. One possibility would be to re-estimate the model with the simplified model formula `ret~s(dur,k=k)+te(gly,bmi,k=k)`, but this leads to a small increase in ML score and AIC (since the model is changed slightly by the change in smoothing penalties between the smooth ANOVA model and the simpler model). In figure 7.10 the joint effect of `bmi` and `gly` is therefore visualised using `vis.gam` (with options `se=1` and `colors="bw"`).

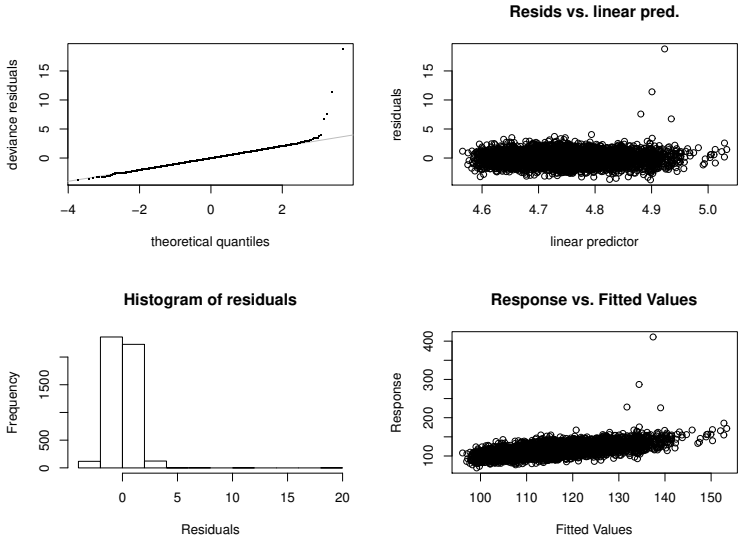

Figure 7.11 *Basic model checking plots for the* `ap0` *air pollution mortality model. For Possion data with moderately high means, the distribution of the standardized residuals should be quite close to normal, so that the QQ-plot is obviously problematic. As all the plots make clear, there are a few gross outliers that are very problematic in this fit.*

It seems that risk increases with increasing glycosylated hemoglobin, as expected, but initially increases less steeply for those with low BMI. Similarly, risk increases initially with disease duration, but there is then some evidence of decline: it could be that the patients only get to have high duration if they tend to have their disease under good control, or it could be that there are just some patients who were not going to get retinopathy, while those less fortunate tended to get it earlier rather than later in their disease progression. Note that not much checking has been done for this example since model checking with binary response data is difficult (see section 7.6 for one possibility), but the flexibility of the assumed model does offer some defence against misspecification problems.

7.4 Air pollution in Chicago

The relationship between air pollution and health is a moderately controversial topic, and there is a great deal of epidemiological work attempting to elucidate the links. In this section a variant of a type of analysis that has become quite prevalent in air-pollution epidemiology is presented. The data are from Peng and Welty (2004) and are contained in a data frame `chicago`. The response of interest is the daily death rate in Chicago, `death`, over a number of years. Possible explanatory variables for the observed death rate are levels of ozone, `o3median`, levels of sulpher dioxide, `so2median`, mean daily temperature, `tmpd`, and levels of particulate mat-

ter, pm10median (as generated by diesel exhaust, for example). In addition to these air quality variables, the underlying death rate tends to vary with time (in particular throughout the year), for reasons having little or nothing to do with air quality.

A conventional approach to modelling these data would be to assume that the observed numbers of deaths are Poisson random variables, with an underlying mean that is the product of a basic, time varying, death rate, modified through multiplication by pollution dependent effects. That is, the model would be

$$\log\{\mathbb{E}(\texttt{death}_i)\} = f(\texttt{time}_i) + \beta_1\texttt{pm10median}_i + \beta_2\texttt{so2median}_i + \beta_3\texttt{o3median}_i + \beta_4\texttt{tmpd}_i,$$

where $\texttt{death}_i$ follows a Poisson distribution, and f is a smooth function.

The model is easily fitted and checked.

```
ap0 <- gam(death~s(time,bs="cr",k=200)+pm10median+so2median+
           o3median+tmpd,data=chicago,family=poisson)
gam.check(ap0)
```

A cubic regression spline has been used for f to speed up computation a little. The checking plots are shown in figure 7.11, and show clear problems as a result of some substantial outliers. Plotting the estimated smooth with and without partial residuals emphasises the size of the outliers.

```
par(mfrow=c(2,1))
plot(ap0,n=1000)  # n increased to make plot smooth
plot(ap0,residuals=TRUE,n=1000)
```

The plots are shown in figure 7.12, and four gross outliers, in close proximity to each other, are clearly visible. Examination of the data indicates that the outliers are the four highest daily death rates occurring in the data, and that they occurred on consecutive days,

```
> chicago$death[3111:3125]
[1] 112  97 122 119 116 121 226 411 287 228 159 142 123 102  94
```

Plotting this section of data also indicates that this peak is associated with a period of very high temperatures and high ozone. One immediate possibility is that the model is simply too inflexible, and that some non-linear response of death rate to temperature and ozone is required. This might suggest replacing the linear dependencies on the air quality covariates with smooth functions, so that the model structure becomes:

$$\log\{\mathbb{E}(\texttt{death}_i)\} = f_1(\texttt{time}_i) + f_2(\texttt{pm10median}_i) + f_3(\texttt{so2median}_i) + f_4(\texttt{o3median}_i) + f_5(\texttt{tmpd}_i),$$

where the f_j are smooth functions. This model is easily fitted

```
ap1<-gam(death ~ s(time,bs="cr",k=200)+s(pm10median,bs="cr")+
    s(so2median,bs="cr")+s(o3median,bs="cr")+s(tmpd,bs="cr"),
    data=chicago,family=poisson)
```

but the gam.check plots are almost indistinguishable from those shown in figure 7.11. Figure 7.13 shows the estimated smooths, and indicates a problem with the distribution of pm10median values, in particular, which might be expected to cause

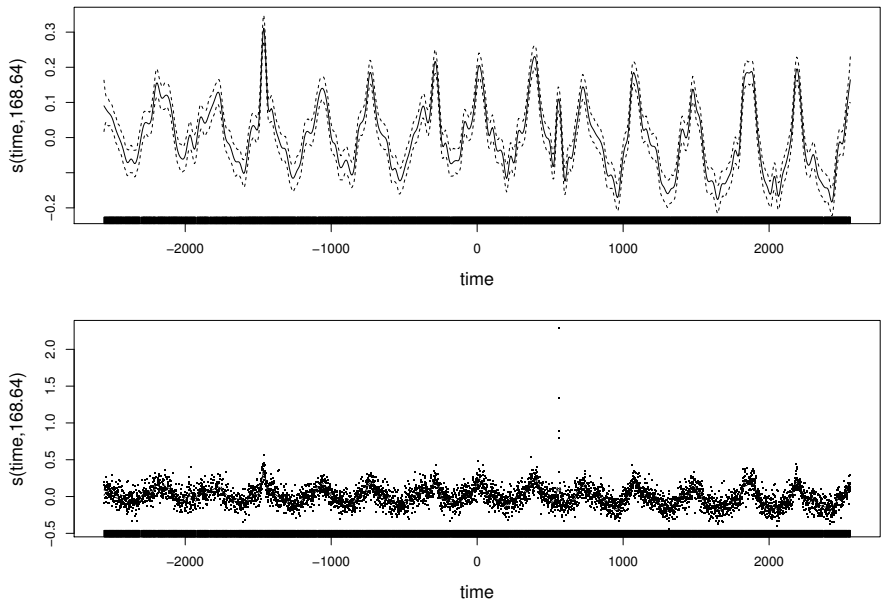

Figure 7.12 *The estimate of the smooth from model* `ap0` *shown with and without partial residuals. Note the 4 enormous outliers apparently in close proximity to each other.*

leverage problems. Similar plots, with partial residuals, again indicate a severe failure to fit the four day run of record death rates. Models with smooth interactions of the pollutant variables are no better.

More detailed examination of the data, surrounding the four day mortality surge, shows that the highest temperatures in the temperature record were recorded in the few days *preceding* the high mortalities, when there were also high ozone levels recorded. This suggests that average temperature and pollution levels, over the few days preceding a given mortality rate, might better predict it than the temperature and levels only on the day itself. On reflection, such a model might be more sensible on biological/medical grounds: the pollution levels and temperatures recorded in the data are not high enough to cause immediate acute disease and mortality, and it seems more plausible that any effects would take some time to manifest themselves via, for example, the aggravation of existing medical conditions.

There are then two types of model that might be appropriate. A *single index model* is specified in terms of smooth functions of weighted sums of covariates, where the weights are estimated as part of estimation. In the current case we would be interested in a weighted sum over lagged covariates. An alternative is a *distributed lag model* in which the response depends on a sum of smooth functions of lagged covariates. Usually the smooth functions at different lags vary smoothly with lag: we

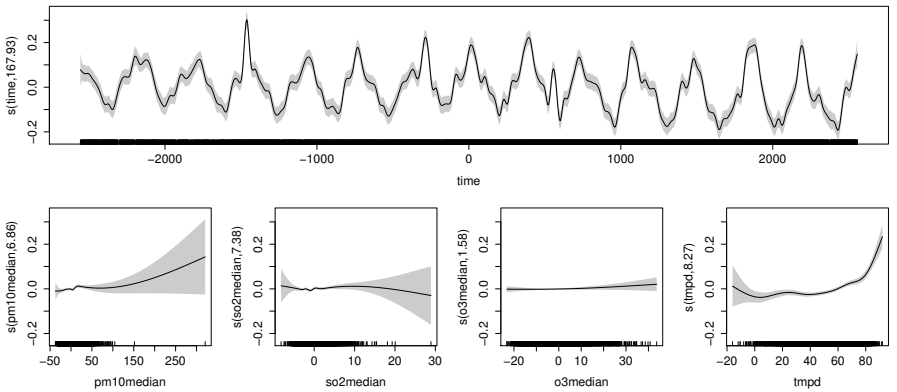

Figure 7.13 *The estimate of the smooth from model* `ap1` *shown without partial residuals. Note the gap in the* `pm10median` *values. Similar plots with partial residuals highlight exactly the same gross outliers as were evident in figure 7.12.*

do not expect the response to yesterday's pollution to have a completely different shape to the response to the day before yesterday's pollution, for example.

7.4.1 A single index model for pollution related deaths

Assuming that the data are arranged in order of increasing time, then an appropriate single index model is

$$\log\{\mathbb{E}(\texttt{death}_i)\} = f_1(\texttt{time}_i) + f_2\left(\sum_{k=0}^{5}\alpha_k \texttt{pm10}_{i-k}\right) + f_3\left(\sum_{k=0}^{5}\alpha_k \texttt{o3}_i, \sum_{k=0}^{5}\alpha_k \texttt{tmp}_i\right)$$

where the α_k are parameters to be estimated. Alternative models with an `so2` effect and/or a 3-way interaction of `pm10`, `o3` and `tmp` do not provide a better fit. Of course it would be possible to argue for different weights, α_k, for the different variables, and more lags could be included: the current structure was chosen to include more lags than seemed likely to have a strong effect while maintaining reasonable computational speed.

As a preliminary for model fitting it helps to prepare a set of 6-column matrices containing the lagged variables in separate columns. For example

```
lagard <- function(x,n.lag=6) {
  n <- length(x); X <- matrix(NA,n,n.lag)
  for (i in 1:n.lag) X[i:n,i] <- x[i:n-i+1]
  X
}
dat <- list(lag=matrix(0:5,nrow(chicago),6,byrow=TRUE))
dat$pm10 <- lagard(chicago$pm10median)
...
```

The easiest way to estimate the single index model is to write a function which accepts the weighting parameter as its first argument, internally re-weights the lagged covariates, and then performs the required GAM fitting call using the re-weighted data. Given such a function we can use R's built in `optim` function to find the weights optimizing the smoothing parameter criterion. Note that without some constraints the weights are not identifiable. They can be made identifiable by insisting that $\alpha_0 > 0$ and $\|\boldsymbol{\alpha}\| = 1$: the function below employs a re-parameterization to enforce this. It is also worth using `bam` (with default REML smoothing parameter estimation) in place of `gam` in the routine, in order to keep the computation time reasonable (using the `discrete=TRUE` option to `bam` would speed up a little more again). A suitable function (adapted from `?single.index`) is:

```
si <- function(theta,dat,opt=TRUE) {
## Return ML if opt==TRUE or fitted gam otherwise.
  alpha <- c(1,theta) ## alpha defined via unconstrained theta
  kk <- sqrt(sum(alpha^2)); alpha <- alpha/kk  ## ||alpha||=1
  o3 <- dat$o3 %*% alpha; tmp <- dat$tmp %*% alpha
  pm10 <- dat$pm10 %*% alpha ## re-weight lagged covariates
  b<- bam(dat$death~s(dat$time,k=200,bs="cr")+s(pm10,bs="cr")+
          te(o3,tmp,k=8),family=poisson) ## fit model
  cat(".") ## give user something to watch
  if (opt) return(b$gcv.ubre) else {
    b$alpha <- alpha  ## add alpha to model object
    b$J <- outer(alpha,-theta/kk^2) ## get dalpha_i/dtheta_j
    for (j in 1:length(theta)) b$J[j+1,j] <- b$J[j+1,j] + 1/kk
    return(b)
  }
} ## si
```

It remains to fit the model and extract the final fitted model.

```
> f1 <- optim(rep(1,5),si,method="BFGS",hessian=TRUE,dat=dat)
> apsi <- si(f1$par,dat,opt=FALSE)
> apsi$alpha
[1]  0.02497  0.66809  0.63504  0.26046  0.26051 -0.11851
```

The fit is fairly slow, taking around half an hour on the laptop computer used to write this, but the results are interesting. Almost no effect is estimated for the temperature on the day of death, the preceding two days are important and the two days before that less so. The fairly small negative coefficient for lag 5 could indicate that increases are somewhat harmful, but could also just be artefact: there might be a case for constraining the α_j to be non-negative here. The `gam.check` plots for `apsi` (not shown) are much improved relative to the previous models. Figure 7.14 shows the estimated effects and a single residual plot. Notice that the rug plot at the bottom of the time effect now has gaps: this is because observations with any NAs in the lagged variables now have to be dropped (the high observations that caused the original bad fits are not dropped this way). The most noticeable effect is the big increase in risk when the weighted sums of ozone and temperature are simultaneously high. The residual plot now looks much improved: there is one large negative outlier, but it is nothing like the magnitude of the previous models' outliers.

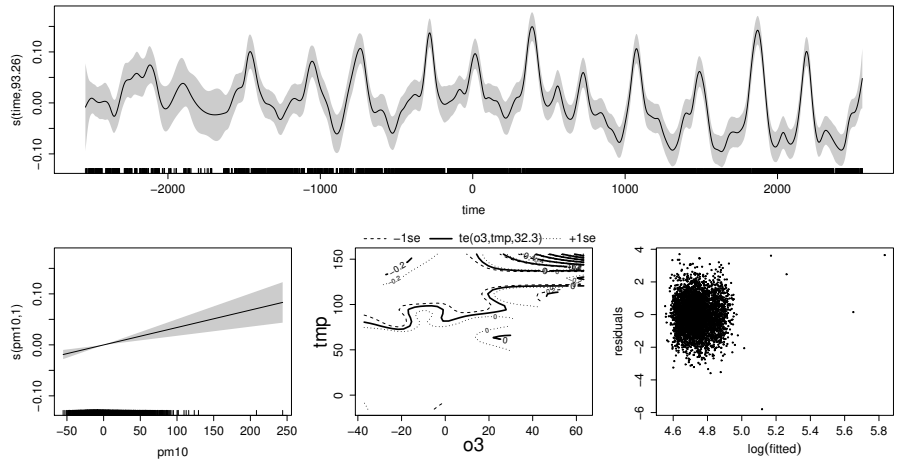

Figure 7.14 *The estimates from the single index model* `apsi`. *Notice the sharp increase in death rate at a combination of high temperature and high ozone over the preceding 4 days. The lower right residual plot for the same model is much improved on the model with only instantaneous daily effects.*

7.4.2 A distributed lag model for pollution related deaths

A distributed lag model represents the effect of a pollutant by a sum of smooth functions of the pollutant at a range of lags. For example the effect of `pm10` on the i^{th} day's deaths might be represented by $\sum_{k=0}^{5} f_k(\texttt{pm10}_{i-k})$, where the f_k are smooth functions to be estimated. This effect is easy to include in a model, via a model formula component something like `s(pm10)+s(pm10.1) ... + s(pm10.5)`, where `pm10.j` contains the `pm.10` observations lagged by j. Of course, we might not be happy that all these effects are estimated completely separately: something like `s(pm10,id=1)+s(pm10.1,id=1) ... + s(pm10.5,id=1)` would force the terms to have the same smoothing parameter. However, the effect estimates could still be wildly different between adjacent lags, which may not be plausible. Another alternative is to insist that the smooth functions themselves vary smoothly with lag. Suppose that f is a bivariate smooth function, while $\texttt{lag}_{ik} = k - 1$ and $\texttt{PM10}_{ik} = \texttt{pm10}_{i-k+1}$. Then an appropriate model for the contribution of `pm10` to the i^{th} day's deaths might be

$$\sum_{k=1}^{6} f(\texttt{PM10}_{ik}, \texttt{lag}_{ik}).$$

Using the 'summation convention' in `mgcv` (see section 7.1), such terms can be specified using a model formula component `te(PM10,lag)`, where `PM10` and `lag` are the six column matrices defined above.

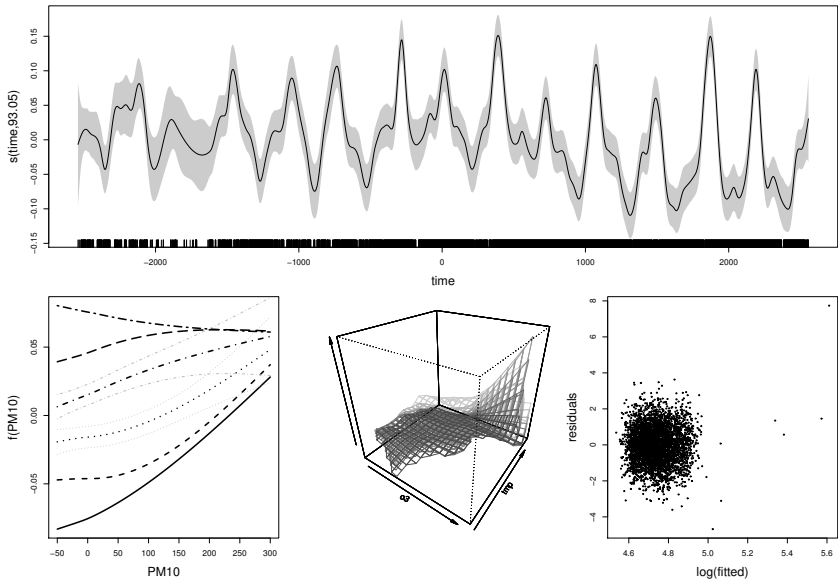

Figure 7.15 *Estimated effects for the distributed lag model of the Chicago air pollution and respiratory deaths data.* Top*: the estimated smooth for the background death rate.* Lower left*: the effect of PM_{10} for each lag from 0 to 5 days. Moving vertically from the bottom left of the plot, the curves are in order of increasing lag. One standard error bands are shown for lags 2 and 3. Notice how there is almost no effect by lags 4 and 5 (curve almost flat), but at shorter lags increased PM_{10} is associated with increased death rate.* Lower middle: *perspective plots of the ozone-temperature interaction by lag (z axis range -0.1 to 0.6). Lighter shades are for longer lags. The dominant feature is high risk at high ozone high temperature combinations, for the middle lags.* Lower right*: the residual plot shows one rather large positive outlier, but is otherwise reasonable.*

A useful model involving the same variables as the single index model is then

$$\log\{\mathbb{E}(\texttt{death}_i)\} = f_1(\texttt{time}_i) + \sum_{k=1}^{6} f_2(\texttt{PM10}_{ik}, \texttt{lag}_{ik}) + \sum_{k=1}^{6} f_3(\texttt{O3}_{ik}, \texttt{TMP}_{ik}, \texttt{lag}_{ik})$$

which can be estimated using

```
apl <- bam(death~s(time,bs="cr",k=200)+te(pm10,lag,k=c(10,5))+
      te(o3,tmp,lag,k=c(8,8,5)),family=poisson,data=dat)
```

A model with an additional distributed lag term for `so2` gives a p-value of 0.28 for the extra term, so there seems little justification for including it. Note that the model with and without `so2` are estimated with different numbers of data, because of missing `so2` data, so comparing them by AIC or GLRT (`anova`) is not valid, unless we re-fit the simpler model only to the subset of data with no missing `so2`.

The distributed lag model fits around 60 times faster than the single index model.

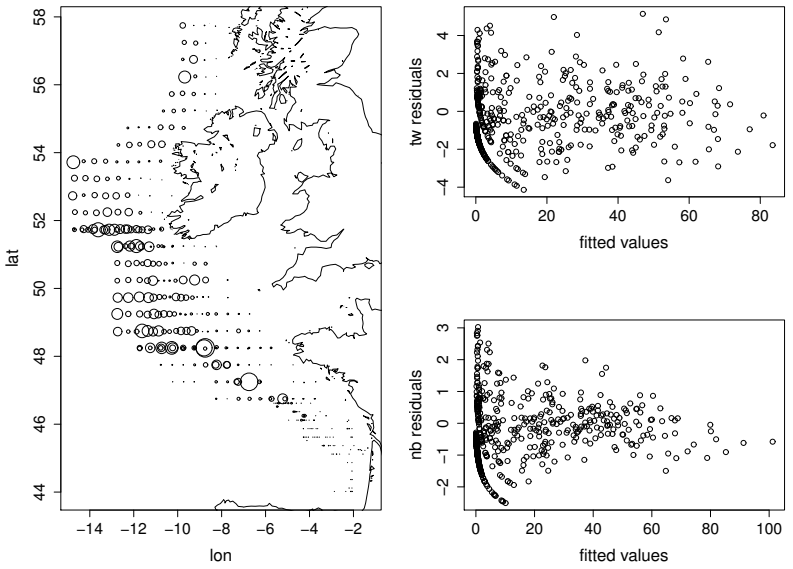

Figure 7.16 *The left hand plot shows the density of stage I mackerel eggs per square metre of sea surface as assessed by net samples in the 1992 mackerel egg survey. Circle areas are proportional to egg density and are centred on the location at which the egg samples were obtained. The right hand plots show residual plots for the Tweedie (upper) and negative binomial (lower) GAMs fitted to these data. The negative binomial deviance residuals appear to be decreasing with the predicted mean, which is problematic.*

It does not do quite as well on the very highest daily death count as the single index model, but does not have the one quite large negative outlier evident from that model. The distributed lag model has the lower AIC of the two. Compact visualization of distributed lag models is not always easy, and figure 7.15 offers a not completely satisfactory attempt (using `predict.gam` to generate the plotting data, and `par(new=TRUE)` in R to overlay perspective plots). The plots are rather consistent with the single index model, with the dominant pollution effect being a big increase in risk associated with high ozone-temperature combinations at intermediate lags.

7.5 Mackerel egg survey example

Worldwide, most commercially exploitable fish stocks are overexploited, with some stocks, such as Newfoundland cod, having famously collapsed. Effective management of stocks rests on sound fish stock assessment, but this is not easy to achieve. The main difficulty is that counting the number of catchable fish of any given species is all but impossible. A standard statistical sampling approach based on trying to catch fish fails, because, for large mobile net-avoiding organisms, there is no way of

relating what is caught to what was available to catch. To some extent, surveys with sonar avoid such catchability problems, but suffer from other problems: chiefly that it is often impossible to determine which fish species cause a given sonar signal. Another alternative is to attempt to reconstruct past fish stocks on the basis of records of what fish get landed commercially, an approach known as 'virtual population analysis', but this suffers from the problem that it tells you quite accurately what the stock was several years ago, but is imprecise about its current or recent state.

Egg production methods are a different means of assessing stocks. The idea is to assess the number of eggs produced by a stock, or the rate at which a stock is producing eggs, and then to work out the number (or more often mass) of adult fish required to produce this number or production rate. This works because egg production rates per kg of adult fish *can* be assessed from adults caught in trawls, while eggs can be sampled in an unbiased manner. Egg data are obtained by sending out survey ships to sample eggs at each station of some predefined sampling grid, over the area occupied by a stock. Eggs are usually sampled by hauling a fine meshed net up from the sea bed to the sea surface (or at least from well below the depth at which eggs are expected, to the surface). The number of eggs of the target species in the sample is then counted and the volume of water sampled is known.

7.5.1 Model development

To get the most out of the egg data it is helpful to model the egg distribution, and here GAMs can be useful. The example considered in this section concerns data from a 1992 mackerel egg survey. The data were first modelled using GAMs by Borchers et al. (1997) and are available in `gamair` dataset `mack`. The left hand panel of figure 7.16 shows the location at which samples were taken, and the egg densities found there. As well as longitude and latitude, a number of other possible predictors of egg abundance are available: `salinity`; surface temperature of the ocean, `temp.surf`; water temperature at a depth of 20 m, `temp.20m`; depth of the ocean, `b.depth`; and, finally, distance from the 200 m seabed contour, `c.dist`. The latter predictor reflects the biologists' belief that the fish like to spawn near the edge of the continental shelf, conventionally considered to end at a seabed depth of 200 m. At each sampling location, a net was hauled vertically through the water column from well below the depth at which eggs are found to the surface: the mackerel eggs caught in the net were counted, to give the response variable, `egg.count`.

A reasonable model for the mean egg counts is

$$\mathbb{E}(\texttt{egg.count}_i) = g_i \times (\text{net area})_i,$$

where g_i is the density of eggs, per square metre of sea surface, at the i^{th} sampling location. Taking logs of this equation we get

$$\log\{\mathbb{E}(\texttt{egg.count}_i)\} = d_i + \log\{(\text{net area})_i\},$$

where $d_i = \log(g_i)$ will be modelled as a function of predictor variables, using an additive structure, and $\log\{(\text{net area})_i\}$ will be treated as a model 'offset': that is,

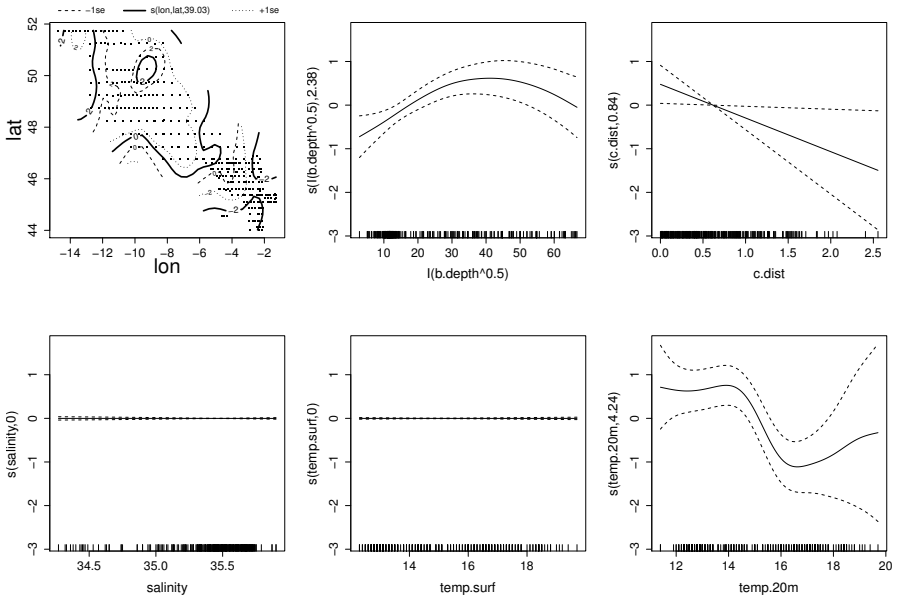

Figure 7.17 *Estimated model terms for the simple additive* `gmtw` *mackerel model.*

as a column of the model matrix with associated parameter fixed at 1. It remains to pick a distribution. Since the response is a count, it is tempting to start with the Poisson distribution, but any attempt to fit the sort of models used here with a Poisson response shows very strong evidence of *over-dispersion*: the residuals are much too large for consistency with a Poisson with unit scale parameter. An alternative is to use the `quasipoisson` family in R, which makes the Poisson-like assumption that $\text{var}(y_i) = \phi\mathbb{E}(y_i)$, but without fixing $\phi = 1$. However, to facilitate things like AIC model selection and REML smoothing parameter estimation[‡] it helps to use full distributions. The obvious alternatives are the Tweedie or negative binomial distributions (see section 3.1.9, p. 115).

For the purposes of this exercise, a simple additive structure will be assumed, with the first model being estimated as follows:

```
mack$log.net.area <- log(mack$net.area)

gmtw <- gam(egg.count ~ s(lon,lat,k=100) + s(I(b.depth^.5))+
       s(c.dist) + s(salinity) + s(temp.surf) + s(temp.20m)+
       offset(log.net.area),data=mack,family=tw,method="REML",
       select=TRUE)
```

[‡]RE/ML smoothing parameter selection has to use the extended quasi-likelihood of McCullagh and Nelder (1989) with `quasi` families, but given Fletcher (2012) it is unclear how reliable this will be in the presence of low expected counts.

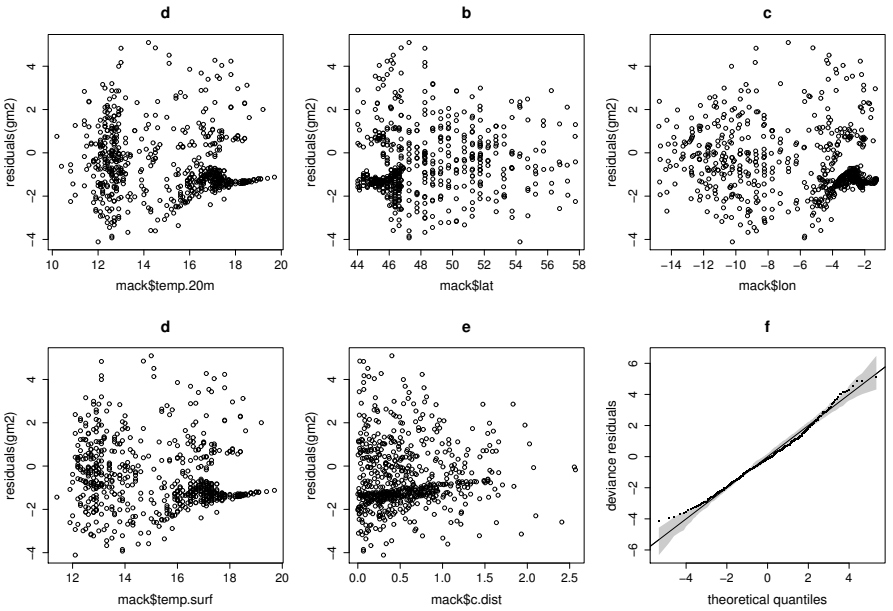

Figure 7.18 *Residual plots for model* `gm2`. f *is a QQ-plot produced by* `qq.gam` *showing the ordered deviance residuals against their theoretical quantiles, if the model is correct. The grey band is a reference band.*

Here the model selection penalties discussed in section 5.4.3 (p. 214) are used by setting argument `select` to `TRUE`. Tweedie and negative binomial distributions with automatic estimation of their extra parameters are only available with REML or ML smoothing parameter estimation, but these also tend to work better than prediction error criteria with selection penalties, because of reduced tendency to under-smooth. The square root transform of `b.depth` is to reduce leverage from very high values. The `tw` family has a log link as default, and will estimate the Tweedie p and scale parameters as part of fitting: the p estimate is 1.33 in this case. A similar call with `nb` in place of `tw` will fit a negative binomial based model (θ is estimated as 1.37).

The basic residual plots on the right hand side of figure 7.16 show acceptable behaviour for the deviance residuals of the Tweedie model, but a problematic decline in the variance of the deviance residuals with fitted value for the negative binomial model, indicating that the negative binomial assumes too sharp an increase in the variance with the mean. AIC (using, e.g., `AIC(gmtw)`) also gives a much smaller value for the Tweedie model. Figure 7.17 shows the effect plots for the Tweedie model, `gmtw`: the `salinity` and `surf.temp` effects have been penalized out of the model altogether. Note however, that there are many missing values for `salinity`, so that dropping it allows a much larger set of data to be used for estimation. Clearly then, we should not drop `surf.temp` before refitting without

the salinity effect. In fact, doing this still gives a fully penalized `temp.surf` effect, so that is dropped too, to arrive at

```
> gm2 <- gam(egg.count ~ s(lon,lat,k=100) + s(I(b.depth^.5)) +
          s(c.dist) + s(temp.20m) + offset(log.net.area),
          data=mack,family=tw,method="REML")
> gm2

Family: Tweedie(p=1.326)
Link function: log

Formula:
egg.count ~ s(lon, lat, k = 100) + s(I(b.depth^0.5)) +
    s(c.dist) + s(temp.20m) + offset(log.net.area)

Estimated degrees of freedom:
54.48  3.26  1.00  6.21  total = 65.95

REML score: 1566.855
```

Some residual plots for `gm2` are shown in figure 7.18. The plots appear reasonable, with nothing to indicate much wrong with the assumed mean-variance relationship, nor other suspicious patterns. The small dense block of residuals on each plot relates to the densely sampled low abundance block in the south of the survey area. Figure 7.19 plots the non-linear effects for this model. In some respects the high degrees of freedom estimated for the spatial smooth is disappointing: biologically it would be more satisfactory for the model to be based on predictors to which spawning fish might be responding directly. Spatial location can really only be a proxy for something else, or the result of a process in which much of the pattern is driven by spatial correlation.

7.5.2 Model predictions

The purpose of this sort of modelling exercise is assessment of the total stock of eggs, which means that predictions are required from the model. In the first instance a simple map of predicted densities is useful. The data frame `mackp` contains the model covariates on a regular grid, over the survey area, as well as an indexing column indicating which grid square the covariates belong to, in an appropriate 2D array. The following code produces the plot on the left hand side of figure 7.20.

```
mackp$log.net.area <- rep(0,nrow(mackp))
lon <- seq(-15,-1,1/4); lat <- seq(44,58,1/4)
zz<-array(NA,57*57); zz[mackp$area.index]<-predict(gm2,mackp)
image(lon,lat,matrix(zz,57,57),col=gray(0:32/32),
      cex.lab=1.5,cex.axis=1.4)
contour(lon,lat,matrix(zz,57,57),add=TRUE)
lines(coast$lon,coast$lat,col=1)
```

Notice the substantial problem that the egg densities remain high at the western boundary of the survey area.

Typically, uncertainty estimates are required for quantities derived from fitted

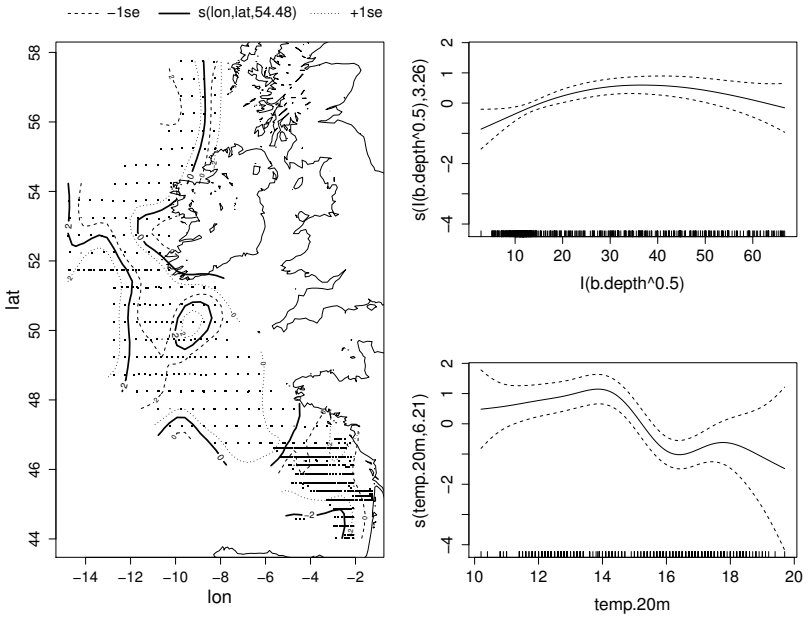

Figure 7.19 *Estimated smooth terms for the mackerel model* `gm2`. *The left panel shows the smooth of location. The upper right panel shows the smooth of seabed depth and the lower right panel shows the smooth of temperature at 20 metres depth. The* `c.dist` *effect is a straight line similar to that shown in figure 7.17.*

model predictions, and as in the brain imaging example, these can be obtained by simulation from the posterior distribution of the model coefficients. For example, the following obtains a sample from the posterior distribution of the mean density of mackerel eggs across the survey area, shown in the upper right panel of figure 7.20.

```
set.seed(4) ## make reproducible
br1 <- rmvn(n=1000,coef(gm2),vcov(gm2))
Xp <- predict(gm2,newdata=mackp,type="lpmatrix")
mean.eggs1 <- colMeans(exp(Xp%*%t(br1)))
hist(mean.eggs1)
```

A disadvantage of such simulations is that they are conditional on the estimated smoothing parameters, $\hat{\boldsymbol{\lambda}}$. That is, we are simulating from the posterior $f(\boldsymbol{\beta}|\mathbf{y}, \hat{\boldsymbol{\lambda}})$, when we would really like to simulate from $f(\boldsymbol{\beta}|\mathbf{y})$. One approach is to use the simple smoothing parameter uncertainty correction of section 6.11.1 (p. 302) to account for this. All we need to do is to substitute `vcov(gm2,unconditional=TRUE)` for `vcov(gm2)` in the above code. Let `mean.eggs` contain the result of this: its histogram is shown at the lower right of figure 7.20. The unconditional distribution is a little wider than the fixed smoothing parameter version.

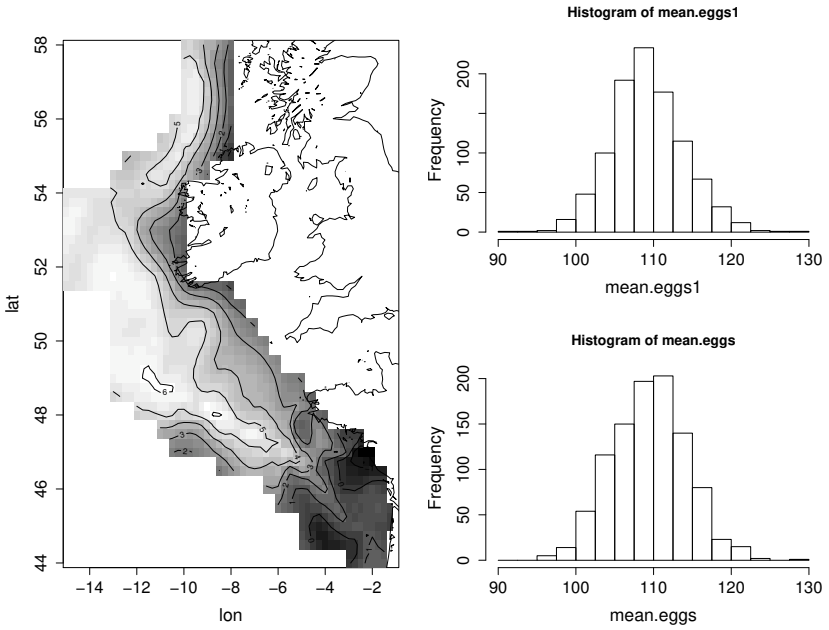

Figure 7.20 *The left hand panel shows predicted log densities of mackerel eggs over the 1992 survey area, according to the model* `gm2`. *The upper right panel shows the posterior distribution of mean egg densities per square metre sea surface, conditional on the estimated smoothing parameters. The lower left panel is the same, but unconditional.*

7.5.3 Alternative spatial smooths and geographic regression

If we were being very fussy, we might object that longitude and latitude are not really an isotropic co-ordinate system, so that a thin plate spline is not appropriate. This view carries an implicit assumption that our model is a much better representation of reality that is likely to be the case, but we could address it anyway, by using a spline on the sphere, as discussed in section 5.5.3 (p. 222). All we need do is to replace the spatial smooth in `gm2` with `s(lon,lat,bs="sos",k=100)`. The AIC gets slightly worse when this is done, but predictions from the model in figure 7.21(a) are almost indistinguishable from the original `gm2` model.

A more useful alternative might be to replace the spatial smooth with a Gaussian process smooth (section 5.8.2, p. 241) in which the autocorrelation function is forced to drop to zero at some point (one degree separation, for example). Replacing the spatial smooth in `gm2` with `s(lon,lat,bs="gp",k=100,m=c(1,1))` achieves this, selecting a spherical correlation model with maximum range 1. A motivation for doing this is that we might avoid the spatial smoother representing long range auto-correlation that would be better represented using the covariates. In fact the AIC only drops by 2 and the estimated remaining effects are almost unchanged.

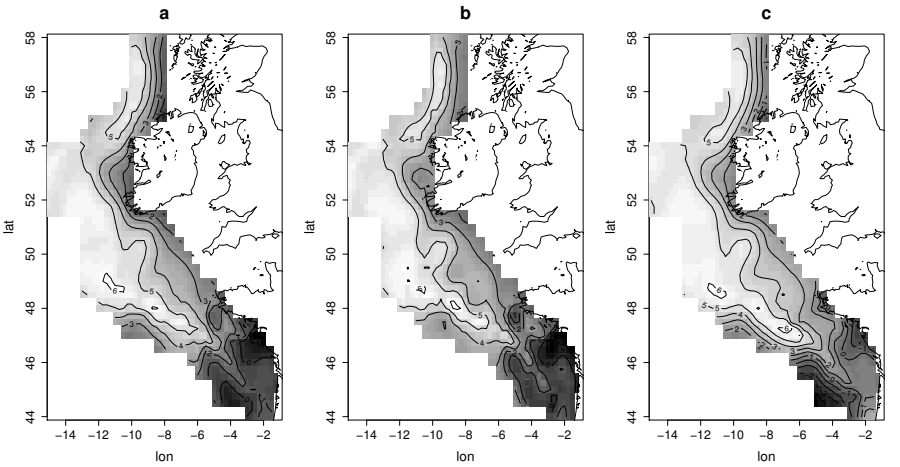

Figure 7.21 (a). *Predictions from the mackerel model with a spline on the spehere used for the spatial effect, to deal 'correctly' with lon-lat locations.* (b). *Predictions from the mackerel model with a Gaussian process smooth of location, using a spherical correlation function.* (c). *Predictions from a geographic regression version of the mackerel model.*

The predictions from the model are shown in figure 7.21(b): they are similar to the original predictions of `gm2`, but noticeably rougher as a result of the suppression of long-range autocorrelation.

A more radical change in structure would be to use a *geographic regression* model. This is a type of *varying coefficient* model in which each covariate is assumed to have a linear influence on the linear predictor for the response, but the slope parameter of that linear dependence varies smoothly with geographic location. So the log egg density term in the mackerel model might become

$$d_i = f_1(\texttt{lon}_i, \texttt{lat}_i) + f_2(\texttt{lon}_i, \texttt{lat}_i)\texttt{temp.20m}_i + f_3(\texttt{lon}_i, \texttt{lat}_i)\sqrt{\texttt{b.depth}_i}$$

where $f_1 - f_3$ are smooth functions of location (letting the coefficient for location derived variable `c.dist` vary with location does not seem like a good idea). This model can by estimated using

```
gmgr <- gam(egg.count ~s(lon,lat,k=100)+s(lon,lat,by=temp.20m)
        +s(lon,lat,by=I(b.depth^.5)) +offset(log.net.area),
        data=mack,family=tw,method="REML")
```

It has an AIC value 22 lower than `gm2`, so is worth serious consideration as an alternative model. Its predictions, shown in figure 7.21c, are noticeably smoother than the previous models, although not radically different in general appearance.

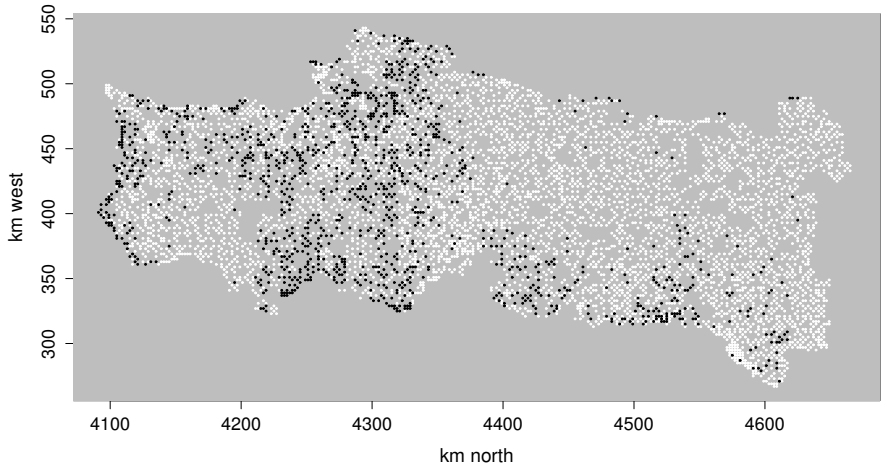

Figure 7.22 *Presence (black) and absence (white) of crested lark in sampled 2 km by 2 km squares in Portugal.*

7.6 Spatial smoothing of Portuguese larks data

Figure 7.22 shows data on the presence or absence of crested lark in each of a set of sampled 2km $\times$ 2km squares, gathered as part of the compilation of the Portuguese Atlas of Breeding Birds. The whole of Portugal was divided into 10km $\times$ 10km squares, each square was further subdivided into 25, 2km $\times$ 2km 'tetrads', and several tetrads (usually 6) were selected from each square. Each selected tetrad was then surveyed, to establish which bird species were breeding within it: crested lark is one of the species surveyed (although it should be noted that there are some problems distinguishing crested lark from Thekla lark).

The compilers of the atlas would like to summarize the information in figure 7.22, as a map, showing how the probability that a tetrad contains breeding crested larks varies over Portugal. An obvious approach is to model the presence – absence data, c_i, as

$$\text{logit}(\mu_i) = f(e_i, n_i)$$

where $\text{logit}(\mu_i) = \log\{\mu_i/(1-\mu_i)\}$, f is a smooth function of location variables e and n, and $c_i \sim \text{binomial}(1, \mu_i)$. The geographic nature of the data suggests using an isotropic smoother to represent f. Given the rather large number of data, it speeds things up to base the TPRS on rather fewer spatial locations than those contained in the entire data set. This is done automatically by the thin plate spline constructor, which will randomly select 2000 of the data locations to use as knots[§] and then build

[§]The same seed is used for exact reproducibility, but without affecting the state of R's random number generator. The number of knots and the random number seed can be changed: see `?tprs`. Alternatively knots can be specified directly via the `knots` argument to `gam`.

the thin plate eigen basis using these. The original data contain spatial locations in metres west and north of some origin. Before fitting let's convert these to kilometres.

```
library(gamair); data(bird)
bird$n <- bird$y/1000;bird$e <- bird$x/1000
m1 <- gam(crestlark ~ s(e,n,k=100),data=bird,family=binomial,
          method="REML")
```

The model is fitted using REML smoothing parameter estimation and a relatively complex model is selected (note that the reported REML score is the negative of the restricted log likelihood, so lower is better).

```
> m1

Family: binomial
Link function: logit

Formula:
crestlark ~ s(e, n, k = 100)

Estimated degrees of freedom:
74.5  total = 75.51

REML score: 2497.513

> plot(m1,scheme=3,contour.col=c1,too.far=0.03,rug=FALSE)
```

produces figure 7.23(a) showing the linear predictor of the model. An equivalent on the response (probability) scale is easily produced using `vis.gam`.

An obvious question with spatial data is whether an alternative spatial smoother might give different, or improved, results. In particular, since Portugal is long and thin, there is quite a big area that is close to the boundary, so we might worry about boundary effects. One way to investigate these is to change the smoothing penalty: for example, a Duchon spline based on first derivative penalisation, with parameter $s = 1/2$ (see section 5.5.2, p. 221), will penalize towards the constant function, which tends to lead to rather restrained boundary behaviour. The model call for this model (`m2`, say) is as the `m1` model call, but with the spatial smoother now set to `s(e,n,bs="ds",m=c(1,.5),k=100)`. Argument `m[1]` specifies a first order derivative penalty, and `m[2]` specifies $s = 1/2$. Figure 7.23(b) shows the estimated linear predictor. There are some differences to `m1`, particularly on the boundary, but the changes are small.

Another alternative would be to use a Gaussian process smoother (see section 5.8.2, p. 241). Judged by REML and AIC the Matérn with $\kappa = 1.5$ seems to fit best in this case. Rather than simply assuming some value for the Matérn range parameter, ρ, it is easy to search for the best fit from a set of plausible alternatives. For example

```
REML <- r <- 1:10*10
for (i in 1:length(r)) {
  mt <- gam(crestlark ~ s(e,n,bs="gp",m=c(3,r[i]),k=100),
            data=bird, family=binomial, method="REML")
  REML[i] <- mt$gcv.ubre
}
```

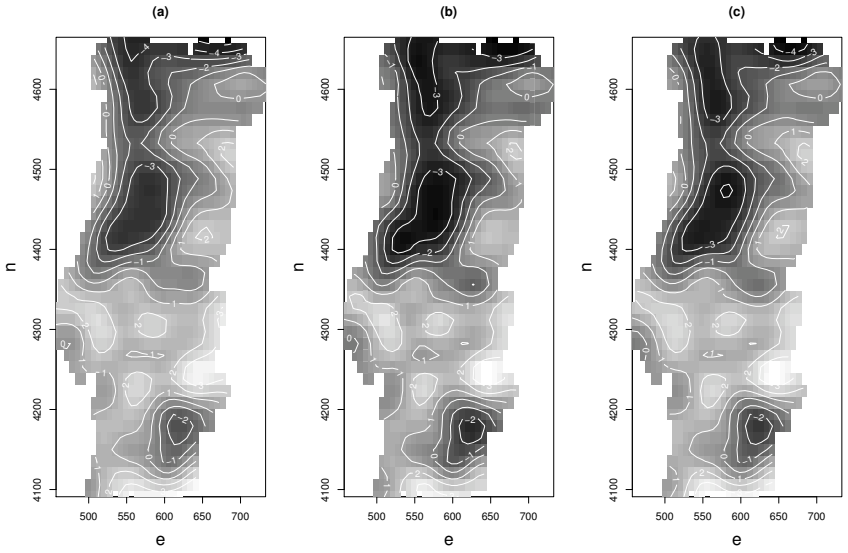

Figure 7.23 (a). *The linear predictor of the crested lark model using a thin plate spline.* (b). *The equivalent to (a) using a Duchon spline with a first derivative penalty.* (c). *The equivalent for a Matérn based Gaussian process smoother.*

shows that there is a minimum at $\rho = 30$. The REML scores at $\rho = 20$ and 40 are both less than 1 different from the value at $\rho = 30$, so there is no need to search on a finer grid. Estimates from the optimum model (`m3`, say) are plotted in figure 7.23c. The plot is rather similar to that from the thin plate spline based model. In fact the thin plate spline model has the lowest REML score of the three alternatives, and AIC also favours this model:

```
> AIC(m1,m2,m3)
          df      AIC
m1 76.07381 4845.400
m2 77.86097 4866.461
m3 75.77660 4863.057
```

Model checking with binary data is somewhat awkward (see exercise 2 in chapter 3), especially for spatial data, but for this application we can simplify matters considerably by choosing to model the data at the 10 km by 10 km square level, in order to obtain a binomial response with easier to interpret residuals. It turns out that the estimated models are almost indistinguishable in terms of predicted probabilities, whether we model raw data or the aggregated data. The `bird` data frame contains a column `QUADRICULA` identifying which 10 km square each tetrad belongs to, so aggregation is easy.

```
bird$tet.n <- bird$N <- rep(1,nrow(bird))
bird$N[is.na(as.vector(bird$crestlark))] <- NA
ba <- aggregate(data.matrix(bird), by=list(bird$QUADRICULA),
                FUN=sum, na.rm=TRUE)
```

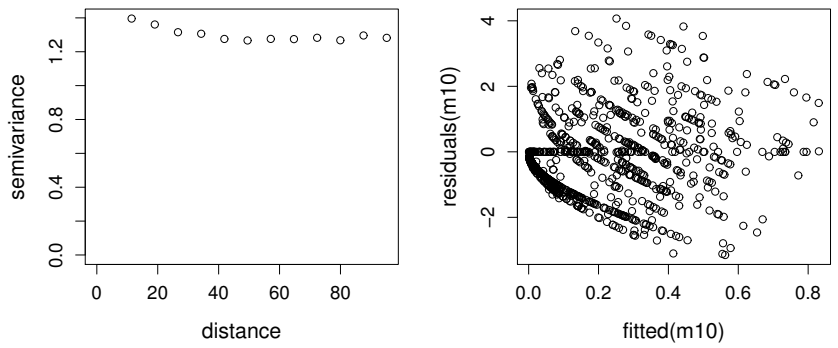

Figure 7.24 *Left: the variogram for the deviance residuals from* `m10`*, the aggregated crested lark binomial model. It is flat and apparently unproblematic. Right: deviance residuals against fitted probabilities, again showing no problems, except slight over-dispersion.*

```
ba$e <- ba$e/ba$tet.n; ba$n <- ba$n/ba$tet.n
```

The model can now be estimated.

```
m10 <- gam(cbind(crestlark,N-crestlark) ~ s(e,n,k=100),
           data=ba, family=binomial, method="REML")
```

A plot looks very similar to that from `m1`.

As usual, the (deviance) residuals for `m10` should be plotted against fitted values to check model assumptions, but conventional residual plots are unlikely to pick up one potential problem with spatial data: namely spatial auto-correlation in the residuals and consequent violation of the independence assumption. To check this, it is useful to examine the *variogram* of the residuals, and the `geoR` package has convenient functions for doing this.

```
library(geoR)
coords<-matrix(0,nrow(ba),2);coords[,1]<-ba$e;coords[,2]<-ba$n
gb <- list(data=residuals(m10,type="d"),coords=coords)
plot(variog(gb,max.dist=100))
plot(fitted(m10),residuals(m10))
```

The plotted variogram is shown on the left of figure 7.24. Uncorrelated residuals should give a more or less flat variogram, while un-modelled spatial auto-correlation usually results in a variogram which increases sharply before eventually plateauing. In the current case, the variogram suggests no autocorrelation, or even slight negative autocorrelation, which might suggest very slight overfitting (although see exercise 4).

The residual versus fitted value plot, shown on the right of figure 7.24, is very good for binomial data, although there is a suggestion of overdispersion: that is the data seem slightly more variable than truly binomial data. Refitting `m10` with a `quasibinomial` family tends to confirm this, yielding a scale parameter estimate around 2.

7.7 Generalized additive mixed models with R

Simple random effects can be included in a GAM in `mgcv` using `s(...,bs="re")` as introduced in section 3.5.2 (p. 154), based on the equivalence of smooths and random effects covered in sections 4.2.4 (p. 172) and 5.8 (p. 239). However, if a model requires a more complex random effect structure than can be handled with simple i.i.d. effects, then it makes sense to convert the smooths in the model to a random effects representation that can be estimated with standard mixed modelling functions, as covered in section 6.8 (p. 288). Another ground for doing this is when there are very large numbers of random effects: mixed modelling functions are typically set up to compute efficiently using the sparsity structure of the random effects in a way that `gam` is not.

`mgcv` includes a `gamm` function which fits GAMMs based on linear mixed models as implemented in the `nlme` library. The function performs the reparameterizations described in section 5.8 (p. 239) and calls `lme` to estimate the re-parameterized model (see section 4.2.4), either directly, or as part of a PQL iteration. It then unscrambles the returned `lme` object so that it looks like a `gam` object, returning a two item list containing the `lme` object for the working mixed model, and the un-scrambled `gam` object.

`gamm` gives full access to the random effect and correlation structures available in `lme`, but it should be noted that it seems to work `lme` quite hard, and it is not difficult to specify models which cause numerical problems in estimation, or failure of the PQL iterations in the generalized case. This is particularly true when explicitly modelling correlation in the data, probably because of the inherent difficulty in separating correlation from trend, when the trend model is itself rather complex. Note also that changes in the underlying optimization methods may lead to slight differences between the results obtained with different `mgcv` and `nlme` versions: these should not be statistically important.

The `gamm4` package provides the equivalent to `gamm` using the `lme4` package. This does not have correlation structures available, but does not rely on PQL for GLMM estimation and is much more efficient than `lme` with 'crossed' (as opposed to nested) random effects. The only other limitation is that only the `t2` construction for tensor product smoothing (see 5.6.5, p. 235) can be used with `gamm4`, because of limitations in the way random effects can be specified. Another option is to use Bayesian stochastic simulation: the `jagam` function provides one option for doing this.

Sometimes a random effects model is required with a large number of smooth curves, which really are random effects. For example we might have a random smooth curve per subject in some experiment. These can be set up to be efficiently computed in `gamm` or `gamm4` using the `"fs"` basis. See `?factor.smooth.interaction` and section 7.7.4.

7.7.1 A space-time GAMM for sole eggs

To start with, let us revisit the Bristol channel sole data one more time. The GLMs and GLMMs considered in sections 3.3.5 and 3.5 were rather unwieldy, as a result of

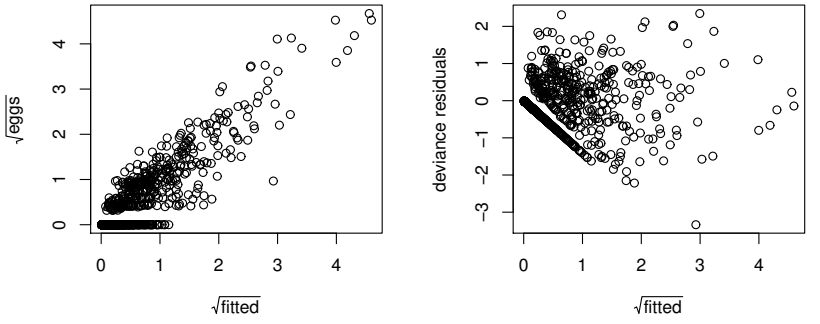

Figure 7.25 *Residual plots for the GAMM of the Bristol channel sole data, fitted by* `bam`*. The left panel shows the relationship between raw data and fitted values. The right panel shows deviance residuals against fitted values. Both plots appear to be reasonable.*

the large number of polynomial terms involved in specifying the models. If nothing else, a GAMM offers a way of reducing the clumsiness of the model formulation. It is straightforward to write the basic sole model (3.15), plus random effect, as a GAMM,

$$\log(\mu_i) = \log(\Delta_i) + f_1(\mathtt{lo}_i, \mathtt{la}_i, \mathtt{t}_i) - f_2(\mathtt{t}_i)\bar{a}_i + b_k,$$

if observation i is from sampling station k. Here f_1 and f_2 are smooth functions, $b_k \sim N(0, \sigma_b^2)$ are the i.i.d. random effects for station and, as before, μ_i is the expected value for $\mathtt{egg}_i$ (given $\mathbf{b}$), while Δ_i and $\bar{a}_i$ are the width and average age of the corresponding egg class. `lo`, `la` and `t` are location and time variables. For simplicity the mortality rate term, f_2, is assumed to depend only on time: this assumption could be relaxed, but it is unlikely that the data really contain enough information to say much about spatial variation in mortality rate.

We need to decide what sort of smooths to use to represent f_1 and f_2. For f_1, a tensor product of a thin plate regression spline of `lo` and `la`, with a thin plate regression spline (or any other spline) of `t`, is probably appropriate: isotropy is a reasonable assumption for spatial dependence (although in that case we should really use a more isotropic co-ordinate system than longitude and latitude), but not for the space-time interaction. For f_2 and the default TPRS suffices. The multiplication of f_2 by $\bar{a}_i$ is achieved using the `by` variable mechanism.

Following on from the previous analyses, here is the call used to fit the model using `gamm`. Note that, unlike `lme`, `gamm` will *only* accept the list form of the `random` argument. Following the previous analyses a `quasipoisson` family is used. We first have to create a `station` factor variable, from the unique sampling station locations.

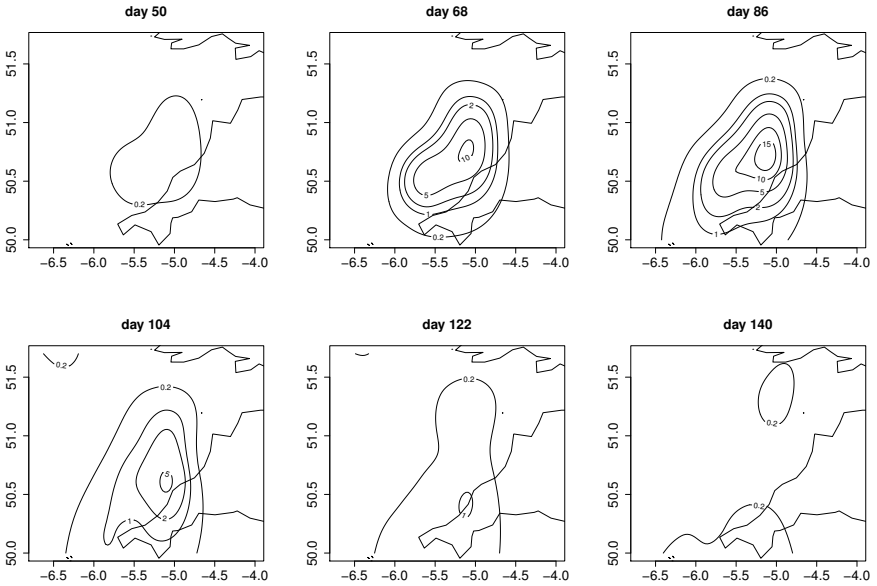

Figure 7.26 *Predicted spawning rates at various times, using the GAMM of Bristol channel sole eggs, presented in section 7.7.1. Note how plots are more peaked and of rather different shape to those from figure 3.19.*

```
solr$station <- factor(with(solr,paste(-la,-lo,-t,sep="")))
som <- gamm(eggs~te(lo,la,t,bs=c("tp","tp"),k=c(25,5),d=c(2,1))
            +s(t,k=5,by=a)+offset(off), family=quasipoisson,
            data=solr,random=list(station=~1))
```

`gamm` returns a list with two components: `lme` is the object returned by `lme`; `gam` is an incomplete object of class `gam`, which can be treated like a `gam` object for prediction, plotting, etc. For example,

```
> som$gam

Family: quasipoisson
Link function: log

Formula:
eggs ~ te(lo, la, t, bs = c("tp", "tp"), k = c(25, 5),
      d = c(2, 1)) + s(t, k = 5, by = a) + offset(off)

Estimated degrees of freedom:
49.77  4.38  total = 55.15
```

A somewhat quicker alternative for fitting, given the modest number of sampling stations, is to use function `bam` as follows

```
som1 <- bam(eggs~te(lo,la,t,bs=c("tp","tp"),k=c(25,5),d=c(2,1))
           + s(t,k=5,by=a)+offset(off)+s(station,bs="re"),
        family=quasipoisson,data=solr)
```

Fitting now takes about one third of the time that `gamm` takes, and the default is to use REML for the working model estimation. Fitting using `gam` is also possible: while slower than `bam` or `gamm` this would also allow negative binomial or Tweedie fits with estimation of all parameters.

Residual plots for `som1` are shown in figure 7.25: these are residuals in which the random effects are at their predicted values. By default, residuals for `som$gam` from `gamm` would have the random effects set to zero, while those for `som$lme` would have the random effects set to their predicted values. This difference also carries over to prediction. Using `predict` with `som$gam` will give predictions in which the random effects are set to zero. By contrast `predict` with `som1`, fitted by `bam` (`gam` would be the same), sets the random effects to their predicted values. However, `predict.gam` also allows terms to be excluded from predictions, which is equivalent to setting them to zero. For example `predict(som1,...,exclude="s(station)")` would ensure that the station effect was set to zero in predictions. This feature has been used to produce the predicted spawning rate plots in figure 7.26.

Estimates of the random effect variances are extractable from both `gamm` and `bam`/`gam` fits. For example:

```
> gam.vcomp(som1)

Standard deviations and 0.95 confidence intervals:

               std.dev      lower     upper
...
s(station)   0.9677711 0.82946738 1.1291353
scale        0.5351120 0.51526441 0.5557242
```

or equivalently:

```
> som$lme
...
 Formula: ~1 | station %in% g.0 %in% g
        (Intercept)  Residual
StdDev:   0.9710766 0.5340049
...
```

`g.0` and `g` are dummy variables associated with re-casting the smooths as random effects and can be ignored. The important information is the station random effect standard deviation (0.97) and the residual standard deviation (0.53).

Soap film improvement of boundary behaviour

An obvious objection to the results in figure 7.26 is that the model predicts high fish egg densities on land, and in one or two places near the edge of the survey area predicts low densities where there are almost certainly no eggs at all (but there was no sampling because of the certainty of finding nothing). In practice it could be argued

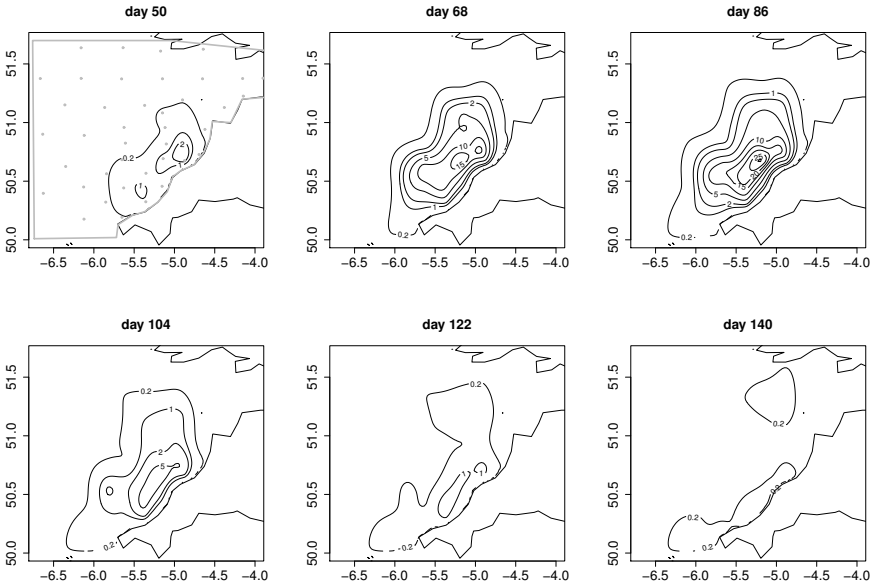

Figure 7.27 *Predicted spawning rates at various times, using the soap film smooth based GAMM of Bristol channel sole eggs, presented in section 7.7.1. Note how the contour plots now descend to near zero at the coast. The top left panel shows the boundary region and knots in grey.*

that this matters little, and that for assessing the total spawning we would anyway only integrate over the sea, and could easily exclude areas where eggs should not be. On the other hand the statistician might have difficulty selling this argument to the scientists who have spent several weeks, often in bad weather, collecting the egg data at considerable expense.[†] In particular one might worry about a model that predicted high egg densities right up to the beach, when in reality density will decline to near zero there.

One way of achieving a model with better boundary behaviour is to use a finite area soap film smoother (see section 5.5.4, p. 223), in place of the thin plate spline of location, to allow some control of the boundary. In particular we could set things up so that the boundary smooth of the soap film is almost constant over space and time, so that the data will then force the boundary density to be modelled as close to zero. The space time smoother in our model is a tensor product of a spatial smoother and a smooth in time. To use the soap film as the spatial smooth marginal in `mgcv` requires that we split the doubly penalized soap smooth into two singly penalized

[†]The author's misspent youth included just one week of doing just that, leading to some sympathy with the scientists' position.

components: the boundary interpolating soap film, and the smooth departures from this interpolating film in the interior of the boundary. These can then be made into two separate tensor product smooths with time. mgcv supplies "sf" ('soap film') and "sw" ('soap wiggly') smooths for this purpose. In fact for our current model, where we want the boundary smooth to be constant in space and time, we can drop the tensor product with "sf" altogether (since a constant smooth subject to a sum-to-zero identifiability constraint is simply zero).

Soap film smoothers require that polygons describing the boundary are supplied as the bnd component of the xt argument to the smooth specification. bnd contains a list of lists. There is a list for each boundary defining polygon, and each polygon is a list of two named arrays defining co-ordinates of polygon vertices: the names must match the smoother arguments. For a simple boundary there is only one polygon defined in the list, but if a region also contains islands/ holes then there may be more polygons, defining these. See ?soap for examples. To obtain a suitable boundary polygon for the sole egg data I produced a plot of the sampling station locations and coast, and then used R's locator function to define a suitable simple boundary. Note that this is also a good approach when a boundary is rather complicated, since a simple boundary 'enclosing' the real boundary of interest is usually sufficient, and less likely to produce artefacts than a highly convoluted boundary. The top left panel of figure 7.27 shows the boundary in grey.

We also have to choose the knots of the soap film smoother.‡ I simply selected a nicely spread out subset of the sampling station locations for one sampling occasion (shown in grey in the top left panel of figure 7.27). The fitting code is as follows.

```
sole$station <- solr$station
som2 <- bam(eggs ~ te(lo,la,t,bs=c("sw","cr"),k=c(40,5),
            d=c(2,1),xt=list(list(bnd=bnd),NULL)) +
            s(t,k=5,by=a) + offset(off) + s(station,bs="re"),
            knots=knots, family=quasipoisson, data=sole)
```

The te statement defining the space-time interaction of the interior soap film and time now requires a list of xt objects to be passed to the marginal smooths. The first item in the list is the important one, containing the soap boundary; the second item can be NULL as the "cr" basis does not require an xt. The rest of the model specification is as before, except that there is now a knots argument supplying the soap film knots.

Figure 7.27 shows the predicted egg density distributions for the new model. The distributions are somewhat more biologically realistic now. The term wise effective degrees of freedom for this model are very similar to the previous model (around 50 for the space time smooth), and both models have an explained deviance of 88%. AIC or REML comparison of the models is not available here: it would be possible to re-fit using a Tweedie distribution to gain access to AIC, but let's move on.

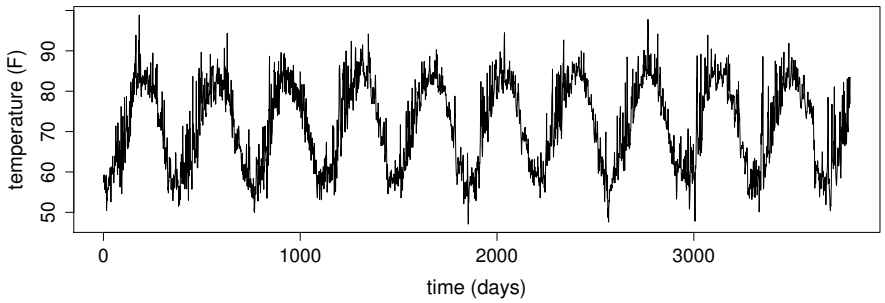

Figure 7.28 *Daily air temperature in Cairo, Egypt from January 1st 1995.*

7.7.2 The temperature in Cairo

Figure 7.28 shows daily temperature in Cairo over nearly a decade. The data are from `http://www.engr.udayton.edu/weather/citylistWorld.htm`. It is clear that the data contain a good deal of noisy short term auto-correlation, and a strong yearly cycle. Much less clear, given these other sources of variation, is whether there is evidence for any increase in average mean temperature over the period: this is the sort of uncertainty that allows climate change sceptics to bamboozle the ill-informed. A reasonable model of these data might therefore be

$$\texttt{temp}_i = f_1(\texttt{time.of.year}_i) + f_2(\texttt{time}_i) + e_i \tag{7.1}$$

where $e_i = \phi e_{i-1} + \epsilon_i$, the ϵ_i being i.i.d. $N(0, \sigma^2)$ random variables. f_1 should be a 'cyclic' function, with value and first 2 derivatives matching at the year ends. The basic idea is that if we model time of year and auto-correlation properly, then we should be in a good position to establish whether there is a significant overall temperature trend.

Model (7.1) is easily fitted:

```
ctamm <- gamm(temp~s(day.of.year,bs="cc",k=20)+s(time,bs="cr"),
         data=cairo,correlation=corAR1(form=~1|year))
```

Note a couple of details: the data cover some leap years, where `day.of.year` runs from 1 to 366. This means that by default the cyclic smooth matches at day 1 and 366, which is correct in non-leap years, but not quite right in the leap years themselves: a very fussy analysis might deal more carefully with this. Secondly, I have nested the AR model for the residuals within year: this vastly speeds up computation, but is somewhat arbitrary.

Examine the `gam` part of the fitted model first:

```
> summary(ctamm$gam)
```

[‡]The boundary file and knots used here can be found in the `gamair` help file `?sole`.

```
Family: gaussian
Link function: identity

Formula:
temp ~ s(day.of.year, bs = "cc", k = 20) + s(time, bs = "cr")

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  71.6581     0.1518   472.2   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms:
                edf Ref.df       F p-value
s(day.of.year) 9.392 18.000 222.803 < 2e-16 ***
s(time)        1.382  1.382   9.633 0.00224 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) =  0.849
  Scale est. = 16.519    n = 3780
```

It seems that there is evidence for a long term trend in temperature, and that the model fits fairly closely. We can also extract things from the `lme` representation of the fitted model, for example 95% confidence intervals for the variance parameters.

```
> intervals(ctamm$lme,which="var-cov")
Approximate 95% confidence intervals

 Random Effects:
  Level: g
                lower      est.     upper
sd(Xr - 1) 0.06064995 0.1450556 0.3477507
  Level: g.0
                    lower         est.     upper
sd(Xr.0 - 1) 2.817526e-07 0.0003473937 0.9780106

 Correlation structure:
        lower      est.     upper
Phi 0.6598207 0.6840085 0.7067803
attr(,"label")
[1] "Correlation structure:"

 Within-group standard error:
   lower     est.    upper
3.913727 4.064353 4.220775
```

The confidence interval for ϕ is easily picked out, and provides very strong evidence that the AR1 model is preferable to an independence model ($\phi = 0$), while the interval for σ is (3.92,4.23). Note that confidence intervals for the smoothing parameters are also available. Under the `Random Effects` heading the interval for `g.1` re-

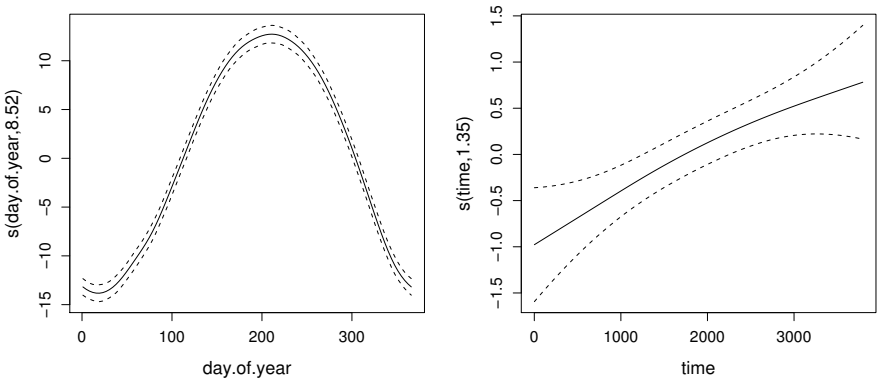

Figure 7.29 *GAMM terms for daily air temperature in Cairo, Egypt from 1st January 1995. The left panel is the estimated annual cycle: note that it has a fatter peak and thinner trough than a sinusoid. The right pattern is the estimated long term trend: there appears to have been a rise of around 1.5 F over the period of the data.*

lates to the smoothing parameter for the first smooth, while that for `g.2` relates to the second smooth. The parameterization is not completely obvious: the reported parameters are σ^2/λ_i, as the following confirms

```
ctamm$gam$sig2/ctamm$gam$sp
s(day.of.year)         s(time)
  0.1450556262   0.0003473937
```

We can plot the estimated model terms using

```
plot(ctamm$gam,scale=0)
```

which results in figure 7.29. The temperature increase appears to be quite marked. The simulation techniques covered in section 7.2.7 can be used to simulate from the posterior distribution of the modelled temperature rise, if required.

The `bam` function also allows a simple AR1 model on the residuals (in data frame order). It has the small advantage that we do not need to nest the AR1 correlation structure within year, but the disadvantage that estimation of the correlation parameter ρ is not automatic. However, we can easily search for a REML optimal ρ as follows:

```
REML <- rho <- 0.6+0:20/100
for (i in 1:length(rho)) {
  ctbam <-bam(temp~s(day.of.year,bs="cc",k=20)+s(time,bs="cr"),
              data=cairo,rho=rho[i])
  REML[i] <- ctbam$gcv.ubre
}
```

The minimum REML is at $\rho = 0.69$, and with this correlation parameter the fitted effects are indistinguishable from those shown in figure 7.29, and the conclusions are identical.

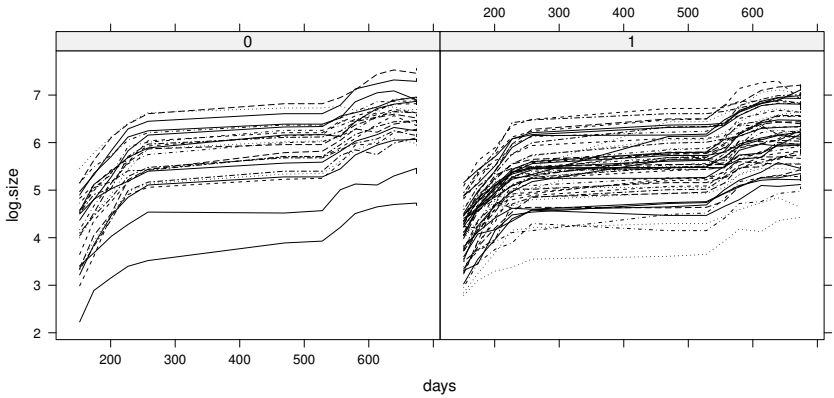

Figure 7.30 *Growth trajectories for 79 Sitka spruce trees, grown in elevated ozone (right) and control (left) conditions.*

7.7.3 Fully Bayesian stochastic simulation: `jagam`

The duality between smooths and random effects/random fields (section 5.8, p. 239) means that we can also estimate generalized additive mixed models in a fully Bayesian way. All that is required, in terms of extra model formulation, is to put priors on the smoothing parameters and any variance parameters, and (usually) on any un-penalized model coefficients (such as those representing the linear part of a cubic spline). There are specialist packages for doing this, in particular `R2BayesX` and `INLA`, but in this section we will consider using the general purpose Gibbs sampling package `JAGS` (Plummer, 2003), via its `rjags` R interface.

Figure 7.30 shows 79 growth trajectories of Sitka spruce trees grown in either elevated ozone or control conditions. They are available in the `sitka` data frame from the `SemiPar` package. A simple smooth additive mixed model would be

$$\texttt{log.size}_i = f(\texttt{day}_i) + d_{\texttt{id}_i} + \epsilon_i, \quad d_{\texttt{id}} \sim N(0, \sigma_b^2), \quad \epsilon_i \sim N(0, \sigma^2),$$

where $\texttt{id}_i$ is the identity of tree to which the i^{th} measurement belongs. The model could easily be fitted with `gamm`, `gam` or `bam`, but can also be used to illustrate the use of `jagam`, which opens up the possibility of much more complicated random effect structures.

A `jagam` call is much like a `gam` call, but rather than fitting a model it writes the `JAGS` code to perform Bayesian simulation from the model. Any random effects structure is not specified as part of the `jagam` call, but is instead added into the `JAGS` code template produced by `jagam`. In this way the rich random effects modelling structure available in `JAGS` can be accessed. `rjags` can then be used to simulate from the model. Functions are also provided to interpret the results after simulation. For the `sitka` model

```
jd <- jagam(log.size ~ s(days) + ozone, data=sitka,
            file="sitka0.jags", diagonalize=TRUE)
```

produces a template file of JAGS code, "sitka0.jags", specifying the Sitka model excluding the random effects. It also returns a list, jd, containing the variables referred to in the code. The code can then be edited to add the random effects, yielding the following ("sitka.jags"). All comments except those beginning '!# ADDED' are autogenerated by jagam.

```
model {
  mu0 <- X %*% b ## expected response
  for (i in 1:n) { mu[i] <- mu0[i] + d[id[i]]} #! ADDED r.e.
  for (i in 1:n) { y[i] ~ dnorm(mu[i],tau) } ## response
  scale <- 1/tau ## convert tau to standard GLM scale
  tau ~ dgamma(.05,.005) ## precision parameter prior
  for (i in 1:nd) {d[i] ~ dnorm(0,taud)} #! ADDED r.e. dist
  taud ~ dgamma(.05,.005) #! ADDED r.e. precision prior
  ## Parametric effect priors CHECK tau=1/58^2 is appropriate!
  for (i in 1:2) { b[i] ~ dnorm(0,3e-04) }
  ## prior for s(days)...
  for (i in 3:10) { b[i] ~ dnorm(0, lambda[1]) }
  for (i in 11:11) { b[i] ~ dnorm(0, lambda[2]) }
  ## smoothing parameter priors CHECK...
  for (i in 1:2) {
    lambda[i] ~ dgamma(.05,.005)
    rho[i] <- log(lambda[i])
  }
}
```

So in this case three lines were needed to add the random effect to the model. Notice that JAGS requires all priors to be proper (although they can be very vague). Hence priors have been put on parametric fixed effects, while the improper priors equivalent to smoothing penalties have to be made proper to be usable. The simplest way to achieve this is via the null space penalties of section 5.4.3 (p. 214), and this is the approach taken by jagam. The added random effects code requires variables id and nd, which must be added to jd:

```
jd$jags.data$id <- sitka$id.num
jd$jags.data$nd <- length(unique(sitka$id.num))
```

Now everything is ready to simulate from the model from within R.

```
library(rjags); load.module("glm")
jm <- jags.model("sitka.jags", data=jd$jags.data,
      inits=jd$jags.ini, n.chains=1)
sam <- jags.samples(jm, c("b","rho","scale","mu"),
       n.iter=10000, thin=10)
jam <- sim2jam(sam, jd$pregam)
```

jags.model compiles the model specification in sitka.jags and initializes the model. jags.samples then simulates from the model by Gibbs sampling, returning every 10^{th} simulated value of the model (fixed + spline) coefficients (b), the log smoothing parameter (rho), etc. A total of 10000 iterations is performed. Finally the sampler output is used to create a simple gam object, jam, for further investigation.

Figure 7.31 shows some results from the model, produced by the following code:

```
plot(jam) ## plot the estimated mean growth trajectory
```

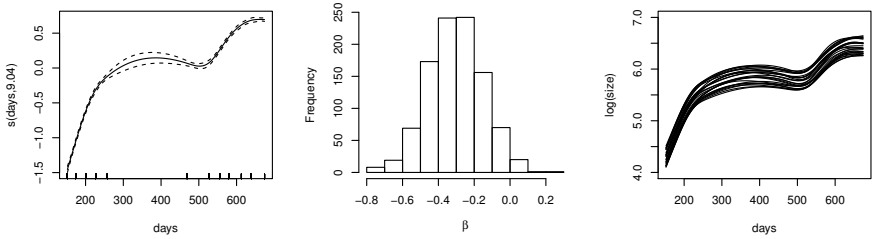

Figure 7.31 *Left: smooth mean Sitka growth currve with 95% credible interval. Middle: histogram of 1000 draws from the posterior of the* `ozone` *effect; there is good evidence for ozone surpressing growth. Right: random growth curves for control trees drawn from the posterior distribution of the model.*

```
hist(sam$b[2,,1]) ## histogram of ozone effect parameter
pd <- data.frame(days=152:674,ozone=days*0)
Xp <- predict(jam,newdata=pd,type="lpmatrix")
ii <- 1:25*20+500 ## draws to select
for (i in ii) { ## draw growth curves
  fv <- Xp%*%sam$b[,i,1] ## growth curve from posterior
  if (i==ii[1]) plot(pd$days,fv,type="l") else lines(pd$days,fv)
}
```

More detail can be found in Wood (2016a). Clearly the real strength of this approach is when a model requires complicated random effects that can be specified in `JAGS` but are difficult to deal with otherwise.

7.7.4 Random wiggly curves

An obvious alternative model for the `sitka` data would have a separate random curve for each tree, in addition to the main average growth curve. These random curves would have no un-penalized null space, instead being shrunk towards zero. The `"fs"` basis provides such random curves: one for each level of a factor. The following code would estimate such a model:

```
sitka$id.num <- as.factor(sitka$id.num)
b <- gamm(log.size ~ s(days) + ozone + ozone:days +
          s(days,id.num,bs="fs",k=5), data=sitka)
plot(b$gam,pages=1)
```

An extra slope parameter as well as an intercept has been allowed for the ozone effect. `s(days,id.num,bs="fs",k=5)` specifies a rank 5 cubic spline (default) for each level of `id.num`. The splines all share the same 3 smoothing parameters: one for the usual spline penalty, and one for each term in the penalty null space. `"fs"` smooths are coded somewhat trickily to work especially efficiently with `gamm` (although there are restrictions: see `?factor.smooth.interaction`), but we could also have estimated the model (more slowly) using `gam` with REML or ML

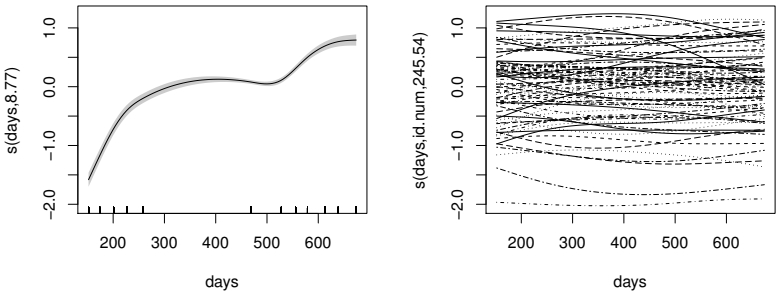

Figure 7.32 *Left: smooth mean Sitka growth currve with 95% credible interval. Right: estimated random departures from mean growth trajectory for each tree.*

smoothing parameter estimation. In either case there is a significant negative slope for enhanced ozone, and the model estimated effects are as shown in figure 7.32.

7.8 Primary biliary cirrhosis survival analysis

Primary biliary cirrhosis is a disease caused by autoimmune damage to the bile ducts of the liver. It can eventually lead to cirrhosis of the liver. The `survival` package in R (Therneau, 2015; Therneau and Grambsch, 2000) contains data frames `pbc` and `pbcseq` from a trial conducted at the Mayo clinic between 1974 and 1984 testing the drug D-penicillamine against a placebo. In the `pbc` dataset a number of baseline measurements are available for each patient, while in the `pbcseq` data frame time varying measurements are available. 6% of the patients exited the trial by receiving a transplant – here these will be treated as censored.[§]

The covariates include patient `id`, `age`, `sex`, treatment (`trt`) and disease `stage`, classified as 1 to 4, where 1 corresponds to subtle bile damage and other early signs and 4 is cirrhosis. The other covariates of interest here are blood measurements of quantities related to liver function. `albumin` is an important blood protein produced in the liver; alkaline phosphotase (`alk.phos`) is an enzyme particularly concentrated in the liver and bile ducts; aspartate aminotransferase (`ast`) is an enzyme commonly used as a biomarker for liver health; `bili`rubin is a product of the breakdown of aged red blood cells processed in the liver and excreted in bile and urine; count of `platelets`, which are the component of blood responsible for clotting, but can be destroyed in a damaged liver; `protime` is a standardised blood clotting time measure related to the previous covariate.

An initial model includes all the preceding covariates, with age and the blood test measurements included as additive smooth effects.

```
pbc$status1 <- as.numeric(pbc$status==2)
```

[§] Actually 3/4 of patients had stage 4 disease by time of transplant, so this sort of 'censoring' is really somewhat informative.

```
pbc$stage <- factor(pbc$stage)
b0 <- gam(time ~ trt+sex+stage+s(sqrt(protime))+s(platelet)+
          s(age)+s(bili)+s(albumin)+s(sqrt(ast))+s(alk.phos),
          weights=status1,family=cox.ph,data=pbc)
```

`protime` and `ast` have been square rooted to reduce their skew. The `alk.phos` effect is estimated to be almost flat with wide confidence intervals, and a high p-value. Dropping it also reduces the model AIC. The `ast` effect then continues to look marginal and has a p-value of 0.16: dropping it increases the AIC only marginally, so it might as well be omitted. `stage` then has a p-value of 0.2, its omission increases the AIC slightly, but it is dropped. The `anova` of the final model is then

```
> anova(b)

Family: Cox PH
Link function: identity

Formula:
time ~ trt + sex + s(sqrt(protime)) + s(platelet) + s(age) +
    s(bili) + s(albumin)

Parametric Terms:
    df Chi.sq p-value
trt  1  0.120  0.7294
sex  1  3.439  0.0637


Approximate significance of smooth terms:
                   edf Ref.df Chi.sq  p-value
s(sqrt(protime)) 1.000  1.001 13.337 0.000261
s(platelet)      1.001  1.002  5.789 0.016141
s(age)           6.043  7.172 29.417 0.000145
s(bili)          4.264  5.223 89.545  < 2e-16
s(albumin)       1.000  1.000 31.086 2.47e-08
```

This provides no evidence for a treatment effect. The smooth effects are plotted in figure 7.33 along with a residual plot of the deviance residuals against the 'risk scores' (simply the linear predictor values), using

```
plot(b); plot(b$linear.predictors,residuals(b))
```

The wedge shaped block of residuals at the lower left of the residual plot is caused by the censored observations and is typical of such plots. The couple of high points at the top left are individuals who died despite relatively low risk scores, and may be slight outliers here. Otherwise the plot appears reasonable. There seems to be a slow upward drift in hazard with age, while the other effects are in line with expectations. Hazard increases with blood clotting time and reduced platelet count. There is an initial sharp increase with serum bilirubin which then levels off, consistent with a poorly functioning liver failing to clear bilirubin, while hazard decreases with serum albumin, presumably as ability to produce albumin increases with liver function.

We can easily produce model predicted survival functions for any set of covariate values, using `predict.gam` with `type="response"`. `predict.gam` uses

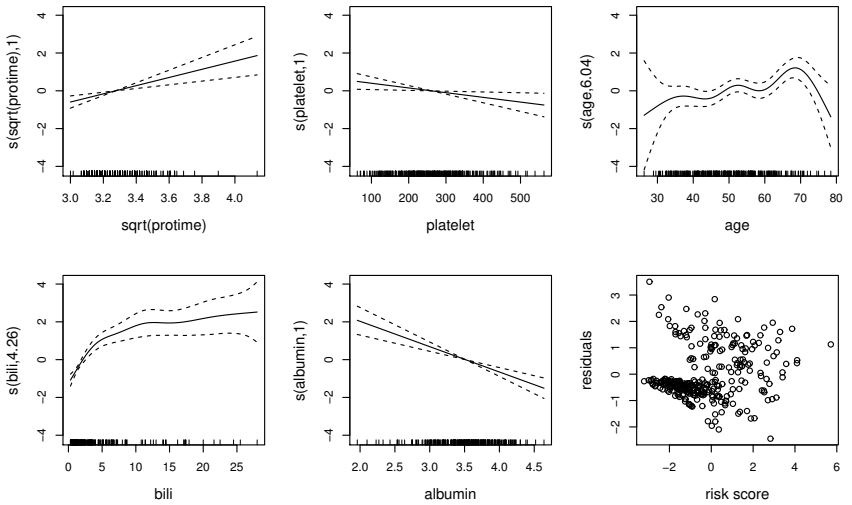

Figure 7.33 *Smooth effects for the primary biliary cirrhosis fixed covariates model along with a corresponding residual plot (lower right).*

(6.25) from section 6.6.2 (p. 287) to compute standard errors on the survivor function scale. These are not ideal for plotting survival function intervals as they can lead to intervals outside $[0, 1]$. However, it is easy to transform to standard errors on the cumulative hazard scale, and then transform cumulative hazard intervals to survival intervals. The following example code plots the estimated survival function and both types of interval (using one standard error) for patient 25's covariates.

```
## create prediction data frame...
np <- 300
newd <- data.frame(matrix(0,np,0))
for (n in names(pbc)) newd[[n]] <- rep(pbc[[n]][25],np)
newd$time <- seq(0,4500,length=np)
## predict and plot the survival function...
fv <- predict(b,newdata=newd,type="response",se=TRUE)
plot(newd$time,fv$fit,type="l",ylim=c(0.,1),xlab="time",
     ylab="survival",lwd=2)
## add crude one s.e. intervals...
lines(newd$time,fv$fit+fv$se.fit,col="grey")
lines(newd$time,fv$fit-fv$se.fit,col="grey")
## and intervals based on cumulative hazard s.e...
se <- fv$se.fit/fv$fit
lines(newd$time,exp(log(fv$fit)+se))
lines(newd$time,exp(log(fv$fit)-se))
```

Figure 7.34 shows such plots for three patients.

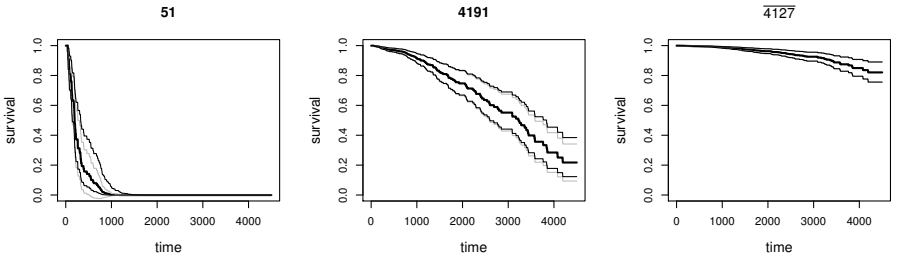

Figure 7.34 *Survival functions and one standard error bands for patients 10, 66 and 25 according to the primary biliary cirrhosis fixed covariates model. The figures above each panel are the patient's actual survival or censoring (barred) time. The thin black lines show the bands produced via transformation from the cumulative hazard scale, whereas the grey lines are approximations directly on the survival scale.*

7.8.1 Time dependent covariates

Now consider the `pbcseq` data, in which the covariates are measured at several times for each patient. The `cox.ph` family is designed only for the fixed covariate case, but the equivalent Poisson model trick of section 3.1.10 (p. 116) can readily deal with this situation, and can just as well be used with penalized smooth models.

A modelling decision must now be made of how to estimate the covariate values at each non-censored event time, as required for the time dependent version of the Cox proportional hazards model. We certainly do not have direct measurements for each patient at each event time. Here let us make the crude assumption that the measurement at an event time is whatever it was the previous time it was measured. This stepwise interpolation is very easy to carry out, but is obviously rather crude: linear or spline interpolation might be preferable.

The first task, then, is to convert the `pbcseq` data frame into a data frame of artificial Poisson data, with the interpolated time varying covariate values. The `?cox.pht` help file example in `mgcv` includes a routine `tdpois` for performing this task, which can be applied as follows (the defaults are for `pbcseq` here!).

```
pbcseq$status1 <- as.numeric(pbcseq$status==2) ## deaths
pb <- tdpois(pbcseq) ## conversion
pb$tf <- factor(pb$futime) ## add factor for event time
```

The resulting `pb` data frame has 28,380 rows — this approach is quite computationally costly. In the fixed covariate case fitting the equivalent Poisson model using `gam` with REML smoothness selection produces the same results as using the `cox.ph` family, but can be quite time consuming, because of the data set size and number of levels of the `tf` factor. Here let's tolerate some approximation and fit using `bam` with the `discrete==TRUE` option (which would still be feasible for millions of data).

Starting with the same dependence on covariates as in the baseline covariates case, backwards model selection drops `alk.phos`, `sex` and `stage` in that order, on the basis of each sequentially being the term with the highest p-value. Each reduc-

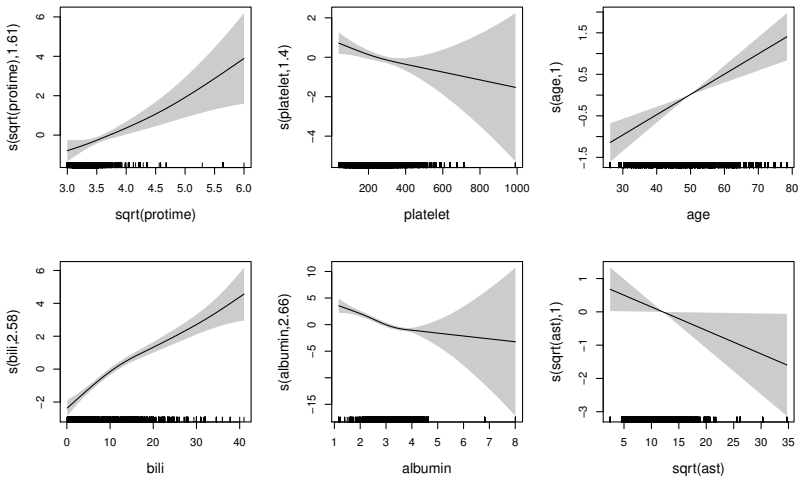

Figure 7.35 *Smooth effects for the primary biliary cirrhosis variable covariates model.*

tion of the model also reduced the AIC. As the trial variable of interest, `trt` was not considered as a candidate for dropping. The fitting call for the selected model was

```
b <- bam(z ~ tf - 1  +  trt + s(sqrt(protime)) + s(platelet) +
         s(age) + s(bili) + s(albumin) + s(sqrt(ast)),
         family=poisson,data=pb,discrete=TRUE,nthreads=2)
```

where as in the `glm` case, '`tf-1`' is the first term in the linear predictor to ensure that we get one coefficient estimated for each event time, rather than an intercept and difference terms. The option `nthreads=2` tells `bam` to use two CPU cores when possible. `anova(b)` or `summary(b)` gives a p-value of about 0.7 for `trt`, so again there is no evidence for a treatment effect.

Residuals (as defined in section 6.6.2, p. 286) are quite easily computed for the time dependent model. The cumulative hazard per subject simply needs to take into account the covariate variability, but this is straightforward.

```
chaz <- tapply(fitted(b),pb$id,sum) ## cum. hazard by subject
d <- tapply(pb$z,pb$id,sum) ## censoring indicator
mrsd <- d - chaz ## Martingale residuals
drsd <- sign(mrsd)*sqrt(-2*(mrsd + d*log(chaz))) ## deviance
```

Figure 7.35 is the result of `plot(b,pages=1,scale=0,scheme=1)`. Compared to the baseline covariates model there is slightly more non-linearity evident in the estimated effects now, although the estimated effect of age is a simple straight line. The direction and broad interpretation of the effects are as before.

Survival function prediction is essentially the same as in section 3.3.3 (p. 136) following the calculations described in section 3.1.10 (p. 116). Here is some code to plot the survivor function for subject 25 again:

```
te <- sort(unique(pb$futime)) ## event times
di <- pbcseq[pbcseq$id==25,] ## data for subject 25
```

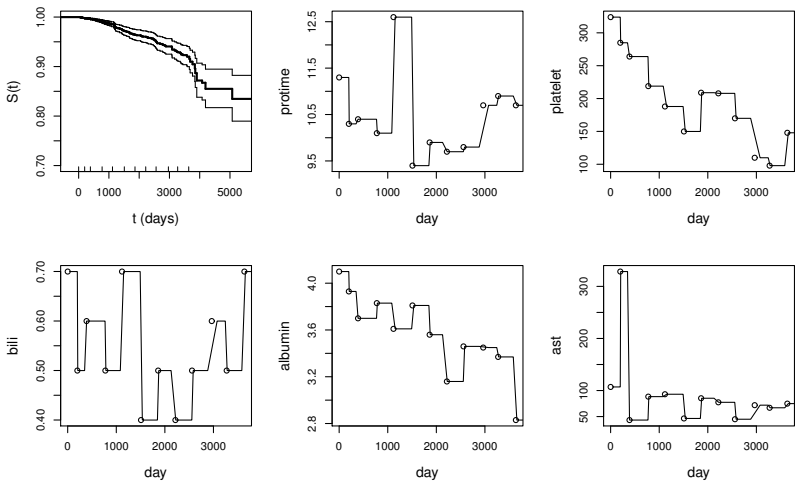

Figure 7.36 *Top left: survival function and one standard error bands for subject 25, according to the primary biliary cirrhosis variable covariates model. Remaining panels: the covariates through time.*

```
## interpolate to te using app from ?cox.pht...
pd <- data.frame(lapply(X=di,FUN=app,t=di$day,to=te))
pd$tf <- factor(te)
X <- predict(b,newdata=pd,type="lpmatrix")
eta <- drop(X%*%coef(b)); H <- cumsum(exp(eta))
J <- apply(exp(eta)*X,2,cumsum)
se <- diag(J%*%vcov(b)%*%t(J))^.5
plot(stepfun(te,c(1,exp(-H))),do.points=FALSE,ylim=c(0.7,1),
     ylab="S(t)",xlab="t (days)",main="",lwd=2)
lines(stepfun(te,c(1,exp(-H+se))),do.points=FALSE)
lines(stepfun(te,c(1,exp(-H-se))),do.points=FALSE)
rug(pbcseq$day[pbcseq$id==25]) ## measurement times
```

This results in the top left panel of figure 7.36. It is also of interest to relate the survivor function to the changes in covariates over time. The following extracts the time varying covariate record and plots it, also overlaying the interpolated version used for fitting, to produce plots like the remaining panels in figure 7.36

```
er <- pbcseq[pbcseq$id==25,]
plot(er$day,er$ast);lines(te,pd$ast)
```

Notice how the reduction in the survival function relative to the constant baseline covariate model is probably driven by deterioration in serum albumin and platelet counts in particular, and some increase in clotting time towards the end of the data.

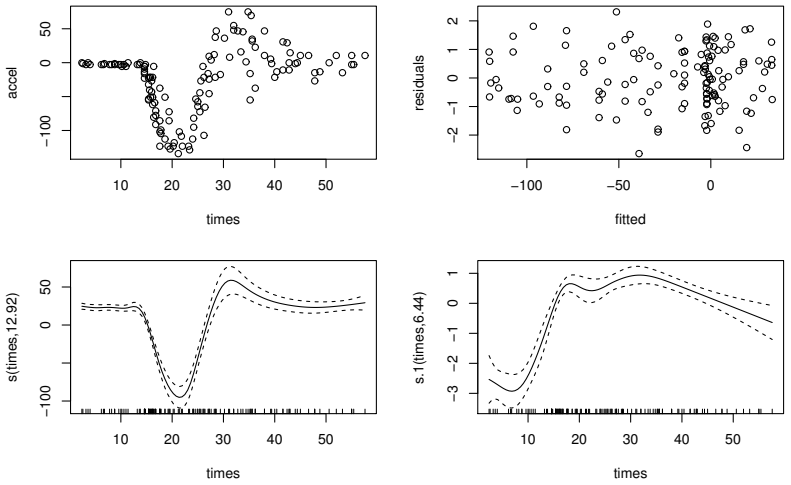

Figure 7.37 *Top left: acceleration of the head of a crash test dummy against time. Top right: residuals of a Gaussian location-scale model against predicted acceleration. Bottom left: estimated smooth for the mean acceleration. Bottom right: estimated smooth for the log (shifted) standard deviation.*

7.9 Location-scale modelling

Sometimes it is useful to allow several parameters of the response distribution to each depend on covariates. When the dependence is via smooth functions, Rigby and Stasinopoulos (2005) call the resulting models 'generalized additive models for location scale and shape' (GAMLSS), while Klein et al. (2015) refer to 'distributional regression'. The methods of section 6.6 (p. 280) handle such models as a special case (see Wood et al., 2016), and some models are available in `mgcv`.[¶]

A very simple example is a Gaussian location-scale model for the `mcycle` data from the `MASS` library, shown at the top left of figure 7.37. These data are measurements of the acceleration of the head of a crash test dummy in a simulated motorcycle accident, against time. Given the variability in both the mean and the variance of the data then the following model might be appropriate:

$$\texttt{accel}_i \sim N(\mu_i, \sigma_i^2), \quad \mu_i = f_1(\texttt{time}_i), \quad \log(\sigma_i - b) = f_2(\texttt{time}_i),$$

where f_1 and f_2 are smooth functions and b is a small lower bound on σ_i ensuring that we avoid singularities in the likelihood. The adaptive smooths of section 5.3.5 (p. 207) might be useful here, so the following code will fit the model.

```
library(MASS);library(mgcv)
b <- gam(list(accel~s(times,bs="ad"),~s(times,bs="ad")),
         family=gaulss,data=mcycle)
```

[¶] At time of writing the `gaulss`, `gevlss`, `multinom` and `ziplss` families.

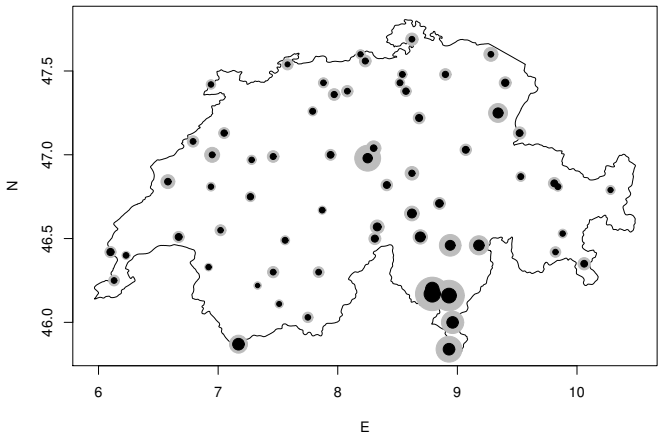

Figure 7.38 *Locations of 65 Swiss weather stations. Grey circle sizes are proportional to the largest 12-hour rainfall seen at the station (1981-2015), while the black circle size shows the average annual maximum.*

Notice how `gam` now requires a list of formulae, with the first specifying the response and the linear predictor for the mean, and the next specifying the linear predictor for the log (shifted) standard deviation. The fitted values for the resulting model will now be a two column matrix. The first column is the predicted mean ($\hat{\mu}_i$), while the second is the square root of the precision (i.e., $1/\hat{\sigma}_i$). `predict` will also return matrix results for `type "response"` and `"link"`. The top right panel of figure 7.37 shows the residuals against fitted mean, while the lower left panel is $\hat{f}_1$ and the lower right is $\hat{f}_2$. See `?gaulss` for more detail. Notice that the variance parameter in models of this class is being estimated by penalized maximum likelihood, and therefore suffers the usual downward bias at modest sample sizes discussed in section 2.4.5 (p. 83).

7.9.1 Extreme rainfall in Switzerland

The `swer` data frame contains the highest rainfall recorded in any 12-hour period each year for 65 Swiss weather stations, with covariates, from 1981 to 2015. Figure 7.38 shows where they are (using package `mapdata` for the outline of Switzerland), and gives some impression of the data variability. The generalized extreme value distribution is often used as a model for 'batch extreme' data of this sort. Its p.d.f. is

$$t(y)^{\xi+1}e^{-t(x)}/\sigma \text{ where } t(y) = [1 + \{(y-\mu)/\sigma\}\xi]^{-1/\xi}$$

($\xi \neq 0$) and $t(y) = \exp\{-(y-\mu)/\sigma\}$ ($\xi = 0$). The `mgcv` version uses the simple approximation that $\xi = \pm\epsilon$ for $|\xi| < \epsilon$, where ϵ is a small constant.

The `gevlss` family in `mgcv` allows μ, $\log\sigma$ and ξ to depend on smooth linear predictors. Maximum likelihood estimation for the GEV distribution is consistent when $\xi > -1$, and the GEV distribution has finite variance when $\xi < 1/2$, so `mgcv` uses a modified logit link to keep ξ in the interval $(-1, 1/2)$.

The initial model fit was as follows

```
library(mgcv);library(gamair);data(swer)
b0 <- gam(list(exra ~ s(nao)+ s(elevation)+ climate.region+
                      te(N,E,year,d=c(2,1),k=c(20,5)),
      ~ s(year)+ s(nao)+ s(elevation)+ climate.region+ s(N,E),
      ~ s(elevation)+ climate.region),family=gevlss,data=swer)
```

That is, the annual maximum 12 hourly rainfall, `exra`, was modelled using a GEV distribution, in which the location parameter, μ, depends on a space-time interaction term, which of 11 `climate.regions` the station belongs to, station `elevation` in metres and the annual North Atlantic Oscillation index `nao`. The `nao` measures pressure anomalies between the Azores and Iceland, and is somewhat predictive of whether western Europe is dominated by warm wet weather from the Atlantic, or colder continental weather. The scale parameter, $\log\sigma$, was modelled as depending on `year` and location (`N` and `E`) additively, but was otherwise as the location parameter. The shape parameter ξ was modelled as depending on `elevation` and `climate.region`. The choice of initial model reflects the fact that the data tend to be more informative about location than scale, and about scale than shape.

Model selection proceeded by first replacing the space time interaction term, `te(N,E,year)`, with an additive structure, `s(N,E)+s(year)`, which decreased the model AIC. After that a backward selection approach was used in which `anova` was used to obtain p-values for each term, and the term with the highest p-value (over 0.05) was dropped, provided it also decreased the AIC. For this example the results were rather clear cut, ending up with the following.

```
b <- gam(list(exra~ s(nao)+s(elevation)+climate.region+s(N,E),
         ~ s(year)+ s(elevation)+ climate.region+ s(N,E),
        ~ climate.region),family=gevlss,data=swer)
plot(b,scale=0,scheme=c(1,1,3,1,1,3),contour.col="white")
```

The result of the `plot` call is shown in the top two rows of figure 7.39. Notice the increase in location and scale in the west of the country, not in the rain shadow of high mountains. As expected, elevation generally increases both location and scale, although the complex estimate for the dependence on elevation is quite likely to result from confounding with hidden variables, such as aspect (e.g., west slope versus east slope). The NAO index effect is interesting, with the location parameter apparently having a minimum when the index is around zero: Switzerland is, of course, a long way inland, so this effect is likely to be quite complex. Finally there is a weak year effect evident in the variability.

The bottom row of figure 7.39 plots residuals and raw data against the expected annual maximum 12-hour rainfall. The expected value for the GEV distribution is $\mu + \sigma\{\Gamma(1-\xi) - 1\}/\xi$, so the following code computes it from the three columns of the `fitted` matrix for the model:

```
mu <- fitted(b)[,1];rho <- fitted(b)[,2]; xi <- fitted(b)[,3]
fv <- mu + exp(rho)*(gamma(1-xi)-1)/xi
```

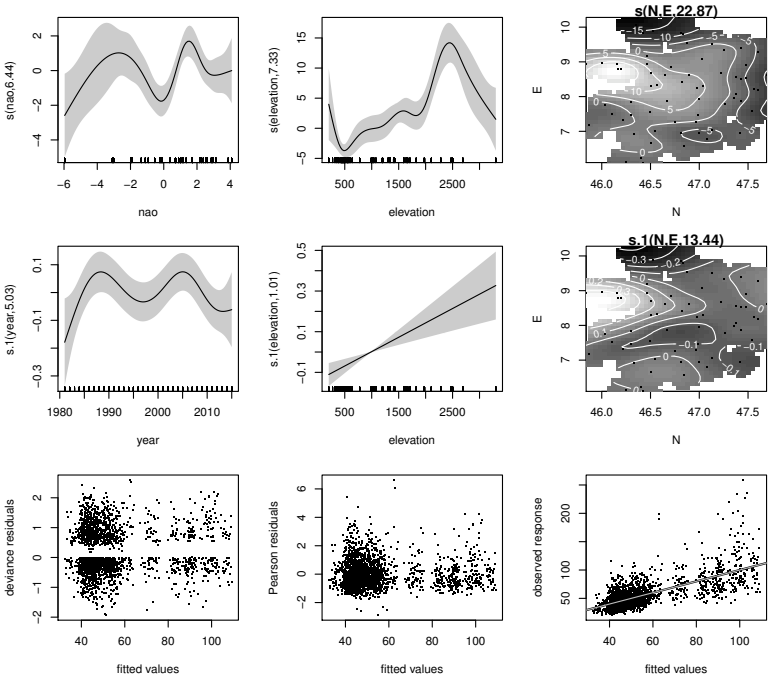

Figure 7.39 *The top two rows show the estimated effects for the Swiss 12-hour extreme rainfall data. The bottom row shows deviance residuals, Pearson residuals and raw data plotted against the model predicted expected annual maximum 12-hour rainfall. The r^2 for the lower right panel is 0.5.*

The `residuals` function computes the deviance and Pearson residuals, as usual.

When modelling extremes some quantile of the fitted distribution is often of interest. For any set of covariates, this can be computed from the quantile function of the GEV given predictions of μ, σ and ξ, but here let's consider a slightly more complicated target requiring some simulation. Specifically, let's target the 98th percentile of the posterior distribution of annual maximum of 12-hour rainfall for each station over its period of operation. This is easily done by simulating from the approximate posterior distribution of the model coefficients, $\boldsymbol{\beta}$, and then simulating replicate data from the resulting GEV. The following code does this for 1000 replicates.

```
Fi.gev <- function(z,mu,sigma,xi) { ## GEV inverse cdf.
  xi[abs(xi)<1e-8] <- 1e-8 ## approximate xi=0, by small xi
  x <- mu + ((-log(z))^-xi-1)*sigma/xi
}
mb <- coef(b);Vb <- vcov(b) ## posterior mean and cov
b1 <- b ## copy fitted model object to modify
n.rep <- 1000; br <- rmvn(n.rep,mb,Vb) ## posterior sim
n <- length(fitted(b))
sim.dat <- cbind(data.frame(rep(0,n*n.rep)),swer$code)
```

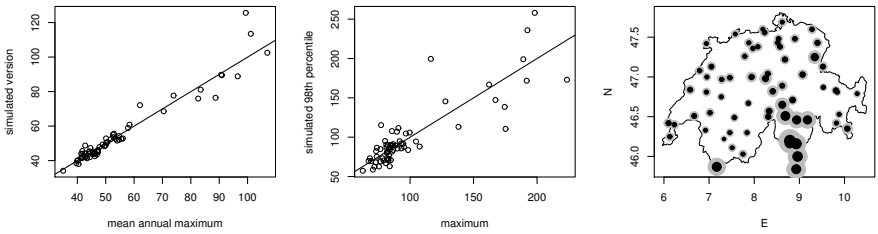

Figure 7.40 *Results of simulating from the posterior annual 12-hour maximum rainfall distribution. The left plot is simulation mean against actual mean, by station. The middle plot is the* 98^{th} *percentile of the annual 12-hour maximum posterior distribution, against maximum observed between 1981 and 2015, by station. The right plot corresponds to figure 7.38, but with grey circle size proportional to the simulated* 98^{th} *percentile and black to the simulated mean.*

```
for (i in 1:n.rep) {
  b1$coefficients <- br[i,] ## copy sim coefs to gam object
  X <- predict(b1,type="response");ii <- 1:n + (i-1)*n
  sim.dat[ii,1] <- Fi.gev(runif(n),X[,1],exp(X[,2]),X[,3])
}
```

So the idea is to simulate 1000 replicate parameter vectors from the posterior for the coefficients, and then for each to predict the corresponding GEV parameters for each year at each station. Given these parameters a value is simulated from the corresponding GEV distribution (by evaluating the quantile function/inverse c.d.f. at a uniform random value). The preceding code is somewhat inefficient for readability purposes. A more efficient version would call `predict` with the `type="lpmatrix"` argument. This produces a single matrix, consisting of a block of columns for each of the three linear predictors. Which matrix column belongs to which linear predictor is given by the `"lpi"` attribute of the returned matrix — a list of three vectors of column indices.

Given `sim.dat` it is now easy to obtain the mean and 98th percentile of the annual maximum for each station.

```
stm <- tapply(sim.dat[,1],sim.dat[,2],mean)
st98 <- tapply(sim.dat[,1],sim.dat[,2],quantile,probs=0.98)
```

Figure 7.40 shows plots of the results of this exercise, indicating that the model is doing a reasonable job in this case.

Chavez-Demoulin and Davison (2005) and Yee and Stephenson (2007) are foundational references for extreme value GAMs (but use backfitting approaches).

7.10 Fuel efficiency of cars: Multivariate additive models

The `mpg` data frame contains data on fuel efficiency of cars along with characteristics of the cars. For each car there is a city driving and highway driving fuel efficiency

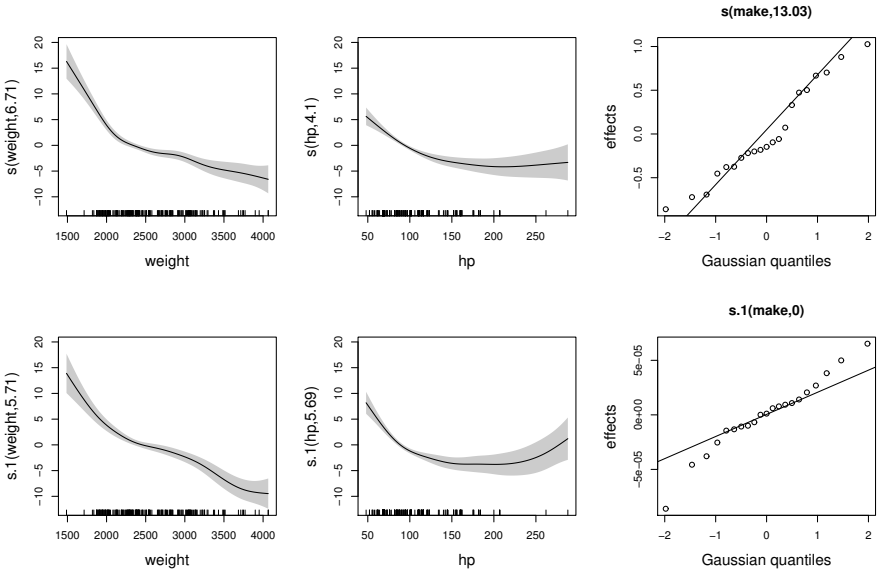

Figure 7.41 *Multivariate additive model for the* `mpg` *data. Note that terms from the second linear predictor are labelled with an extra '*`.1`*' on the term name.*

given in miles per gallon (the data are from the US). It makes sense to treat the fuel efficiency measurements as a bivariate response. If we assume bivariate normality then the `mvn` family in `mgcv` can be used. This implements a simply multivariate additive model in which the mean of each component of the multivariate response has its own linear predictor, and there is the possibility to share terms between linear predictors. The correlations between the response components are also estimated (but do not depend on covariates).

From physical principles (see, e.g., the excellent MacKay, 2008) we would expect that highway fuel consumption would be dominated by aerodynamic factors, and hence would be best modelled in terms of car dimensions (frontal area in particular), while city fuel consumption should be dominated by weight and other engine characteristics. Unfortunately the physically based models I tried all did worse than a rather boring model involving weight and engine power (`hp`) and using the same variables for city and highway. Here is the model fitting code

```
library(mgcv); library(gamair); data(mpg)
b <- gam(list(city.mpg ~ fuel +style +drive +s(weight) +s(hp)
                         + s(make,bs="re"),
              hw.mpg ~ fuel +style +drive +s(weight) +s(hp)
                         + s(make,bs="re")),
             family = mvn(d=2) , data = mpg)
```

`fuel` is diesel or petrol (gasoline), `style` is a factor for style of car, `drive` is a

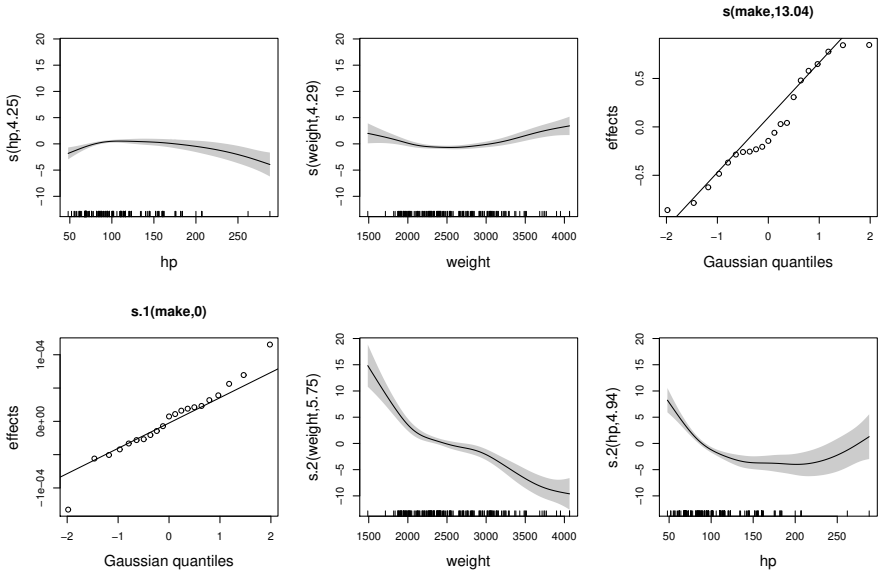

Figure 7.42 *Shared term multivariate additive model for the* `mpg` *data. The effects specified in the third shared model formula are labelled with an extra '*`.2`*'.*

factor for front, rear or all wheel drive. `make` is a factor for manufacturer — a random effect is assumed for this. There is one formula for each component of the bivariate response. Figure 7.41 shows the results. The random effect of `make` is estimated as effectively zero for highway driving, whereas `make` seems to matter for city driving. The smooths for `weight` and `hp` have strikingly similar shapes for city and highway, and the obvious question arises of whether we could simplify the model, by using the same smooths for both components. To address this question a model can be fitted that uses the same smooths of `weight` and `hp` for both `mpg` measurements, but has an extra smooth of both for the `city.mpg`. If the joint smooths are adequate then the extra smooths should turn out to be unnecessary.

The `gam` call for this model now has three formulae.

```
b1 <- gam(list(city.mpg ~ fuel +style +drive +s(hp) +s(weight)
                          + s(make,bs="re"),
               hw.mpg ~ fuel +style +drive +s(make,bs="re"),
                  1+2 ~ s(weight) +s(hp) -1),
          family = mvn(d=2) , data = mpg)
```

The third formula specifies the shared terms. Its left hand side contains numbers and the '+' symbol, in order to indicate which components of the response the formula relates to: here the same `s(weight)` and `s(hp)` will be used in both the `city.mpg` and `hw.mpg` linear predictors. For models with more components then we could have further shared component formulae of this sort (see `?formula.gam`). This

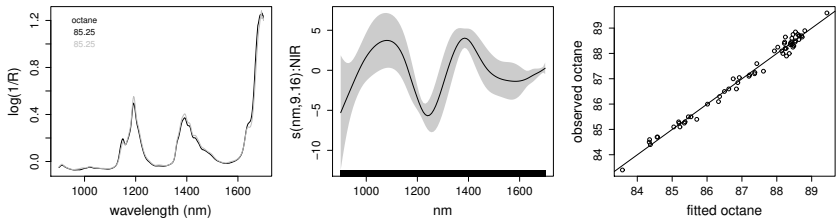

Figure 7.43 *Left: two example near infra-red spectra for gasoline samples with their octane ratings. Middle: estimated coefficient function for the scalar on function regression. Right: observed versus model fitted octane measurements.*

same mechanism can also be used for other multi-formula families. The results of this second model are shown in figure 7.42. Both the figure and a look at the model summary indicate that there are some significant (non constant) differences between the city and highway responses to `weight` and `hp`. The way that city efficiency initially goes up with `hp`, relative to highway, is initially counter intuitive, but probably reflects some relative inefficiency of low powered cars in highway driving. The initial extra effect of `weight` is as expected – extra weight adds disproportionately more consumption in city driving. In fact `b1` has a slightly higher AIC than `b`, so there is not really any good reason to prefer it to the original model.

7.11 Functional data analysis

Functional data analysis (Ramsay and Silverman, 2005) is concerned with models for data where some of the observations involved are best treated as functions. The methods are very closely related to those presented in this book, especially the use of basis expansions and penalties. This section will merely scratch the surface of the topic by presenting a couple of models of this type that can be directly estimated in `mgcv`. See the `refund` package in R (Goldsmith et al., 2016) for much more.

7.11.1 Scalar on function regression

The left hand panel of figure 7.43 shows near infra-red spectra measured for two gasoline/petrol samples (out of 60 available), along with the octane rating of the sample. Octane rating measures the 'knocking resistance' of fuel: the higher the octane rating the higher can be the compression ratio of the engine that burns it. Traditionally, octane rating measurement is somewhat complicated, requiring a variable compression test engine and standardized reference fuel. In comparison obtaining the near infra-red spectrum of fuel is much quicker and cheaper, so it would be good to be able to predict octane rating from a measured spectrum. The `gas` data contain a vector of `octane` measurements, along with a matrix `NIR`, each row of which contains a spectrum for the corresponding `octane` measurement. Matrix `nm` is the corresponding matrix of wavelengths.

A simple model (sometimes known as a 'signal regression model', e.g., Marx and Eilers, 2005) for predicting octane rating from the spectra might be:

$$\texttt{octane}_i = \int f(\nu)k_i(\nu)d\nu + \epsilon_i$$

where $k_i(\nu)$ is a spectrum as a function of frequency, ν, and the ϵ_i are i.i.d. Gaussian errors. f is a smooth function to estimate. Because integration is a linear operation, this model can be estimated by exactly the machinery used for other GAMs. Replacing the integral by a discrete sum ('quadrature') approximation, such a model can be specified and estimated in `mgcv` using the summation convention that applies when a smooth term has matrix arguments. Generically, suppose that a model contains the term `s(X,by=Z)` where `X` and `Z` are matrices of identical dimensions. The contribution to the i^{th} element of the linear predictor from such a term is $\sum_j f(X_{ij})Z_{ij}$, where f is the smooth function. $\mathbf{Z}$ is assumed to be a matrix of ones, if it is not supplied. The summation convention can be used to include a discrete version of any (bounded) linear functional of a function in the linear predictor of a model, by suitable choice of $\mathbf{X}$ and $\mathbf{Z}$.

The following code fits the model and produces the middle and right hand panels of figure 7.43.

```
b <- gam(octane~s(nm,by=NIR,k=50),data=gas)
plot(b,scheme=1,col=1)
plot(fitted(b),gas$octane)
```

From the right panel it is clear that the model works rather well in this case.

Note that these functional regression terms can be mixed with normal smooth terms in a model, and that the summation convention also allows for the smooth function to vary with other covariates. For example suppose that we would like to use a model

$$y_i = \int f_1(\nu, x_i)k_i(\nu)d\nu + f_2(z_i) + \epsilon_i,$$

In such a case three matrices would be needed: `K` containing $k_i(\nu)$ measurements in each row (possibly multiplied by quadrature weights); the matrix `nu`, each row of which contains the ν values at which $k_i(\nu)$ has been measured (usually each row is identical); and `X`, each row of which simply contains the required x_i value repeated for each column (so the columns of `X` are all identical). Then the model formula would be `y ~ te(nu,X,by=K) + s(z)` (assuming default `k` values). A tensor product smooth usually makes sense in such cases, since ν and x are usually not naturally on the same scale.

Prostate cancer screening

Figure 7.44 shows protein mass spectra for blood samples from 9 patients: 3 with each of a healthy prostate gland, an enlarged gland and prostate cancer. Blood testing being much less invasive than biopsy, it is desirable to be able to predict prostate condition from the spectra. Data described in Adam et al. (2002) are available in the `prostate` dataset which contains 654 prostate classifications (`type`: 1, healthy;

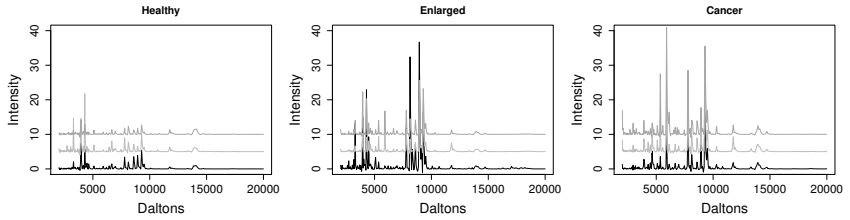

Figure 7.44 *Example protein mass spectra for 3 subjects with healthy, enlarged and cancerous prostate gland. The spectra are vertically shifted for visibility.*

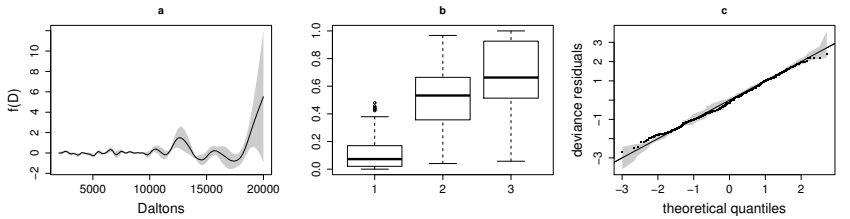

Figure 7.45 (a). *The estimated smooth function and confidence band for the ordered categorical prostate diagnosis model.* (b). *Box plots of the predicted probability of cancer for each subject, by actual status.* (c). *Sorted deviance residuals for the model against simulated theoretical quantiles, with reference band in grey.*

2, enlarged; 3, cancerous) along with the corresponding spectral (`intensity`) for each. `intensity` is a 624×264 matrix, with one spectrum per row. The protein masses at which the spectra were measured are in the matrix `MZ`.

One model to try might be an ordered categorical scalar on function model. The idea is that the linear predictor determines the mean value of a latent random variable, whose value relative to some cut points determines the category observed for a subject. The linear predictor itself is given by the integral of the observed spectrum, multiplied by a coefficient function, to be estimated. So we have that

$$y_i = \begin{cases} 1 & z_i < \theta_1 \\ 2 & \theta_1 \leq z_i < \theta_2 \\ 3 & \theta_2 \leq z_i \end{cases}$$

where z_i is a logistically distributed random variable with mean

$$\mu_i = \int f(m) k_i(m) dm.$$

$k_i(m)$ is the i^{th} spectrum, while $f(m)$ is a smooth function to be estimated. θ_1 is set to -1 for identifiability, but θ_2 is estimated as part of model fitting. In reality the integral is replaced by a discrete sum, of course.

For the current example the fitting code is simply

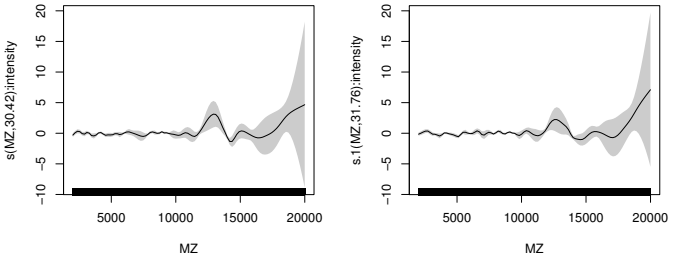

Figure 7.46 *Estimated coefficient functions for the multinomial prostate screening model. Left is for the enlarged class while right is for the cancerous class.*

```
b <- gam(type ~ s(MZ,by=intensity,k=100),family=ocat(R=3),
         data=prostate,method="ML")
```

`ocat` specifies the ordered categorical model with `R=3` classes (see Wood et al., 2016, for full details). The estimated smooth function is quite complex, but also quite robust to the choice of basis and basis dimension. For example, the fit is very similar using an adaptive smoother via `s(MZ,by=intensity,k=100,bs="ad")`. The following code plots the estimated function, shows box plots of the predicted probability of cancer by actual class, and plots a QQ-plot and reference bands for the model's deviance residuals.

```
plot(b,rug=FALSE,scheme=1,xlab="Daltons",ylab="f(D)",
     cex.lab=1.6,cex.axis=1.4)
pb <- predict(b,type="response") ## matrix of class probs
plot(factor(prostate$type),pb[,3])
qq.gam(b,rep=100,lev=.95)
```

The results are shown in figure 7.45, but the middle figure is somewhat disappointing. The model does not distinguish cancer from non-cancer very sharply, and the problem-specific algorithm described in Adam et al. (2002) does better in this case.

A multinomial prostate screening model

Perhaps the problem is the way that the model treats the health status categories as ordered. This approach does not have a strong clinical basis, so we might be better off treating the categories as un-ordered. One model for this is the multinomial logistic regression model implemented in `mgcv` in the `multinom` family.[‖] The idea is that each response observation, y_i, is one of $K+1$ category labels, $0, 1, \ldots, K$. For identifiability reasons category 0 is treated as a reference category, and there is one linear predictor $\eta^{[k]}$ associated with each of the remaining K categories. The linear predictors can contain parametric or smooth terms in the usual way. The probability of y_i taking the value 0 is then $1/\{1+\sum_k \exp(\eta_i^{[k]})\}$, while the probability that it

[‖] Technically this family has the structure of the location scale and shape models of section 7.9 and is implemented in the same way. Of course, the distribution parameters are in a sense all location parameters.

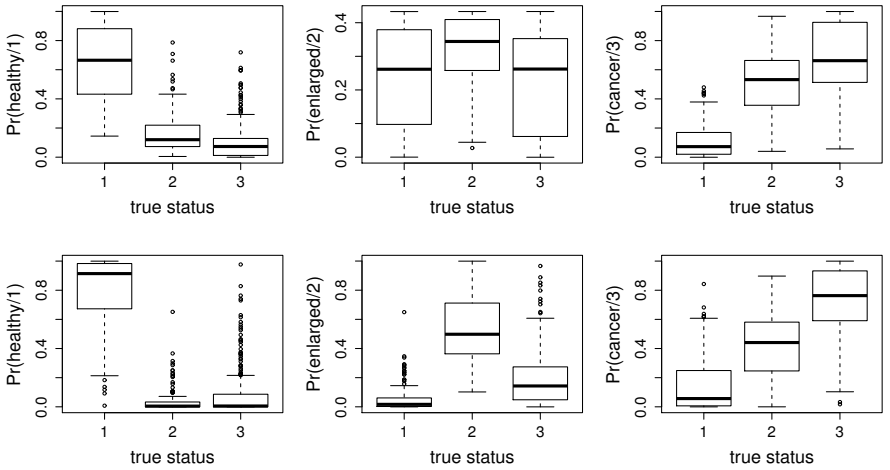

Figure 7.47 *Top row: Box plots of ordered categorical model predicted probabilities for each health category (1-healthy, 2-enlarged, 3-cancer) by actual category. Bottom row: the same for the multinomial model, which appears to be substantially better.*

takes the value $j \neq 0$ is $\exp(\eta^{[j]})/\{1 + \sum_k \exp(\eta_i^{[k]})\}$. In this case we would like both linear predictors to have the same structure used in the ordered categorical case and fitting is therefore as follows.

```
prostate$type1 <- prostate$type - 1 ## recode for multinom
b1 <- gam(list(type1 ~ s(MZ,by=intensity,k=100),
                  ~ s(MZ,by=intensity,k=100)),
            family=multinom(K=2),data=prostate)
```

Figure 7.46 shows the estimated smooths. Notice that while both are somewhat similar to the single coefficient function in the ordered categorical model, there are now some differences between them. `predict(b1,type="response")` will produce the three column matrix of class probabilities for this model. Figure 7.47 gives an impression of the predictive performance of the multinomial model (bottom row) and the ordered categorical model (top row). The multinomial model clearly has much better discriminatory power. For example, if we classify patients according to the category for which their probability is highest, then 89% classified as healthy are really healthy (up from 83%), while 76% of those categorised as having cancer, really do (up from 72%). However, the real gain is in the enlarged category with 76% correctly classified, up from 37%.

Of course, for a real diagnostic procedure we would not classify in this crude way, but would use the probabilities as inputs to a decision rule adjusted to have the best achievable balance of false positive to false negative rate in the cancer classification.

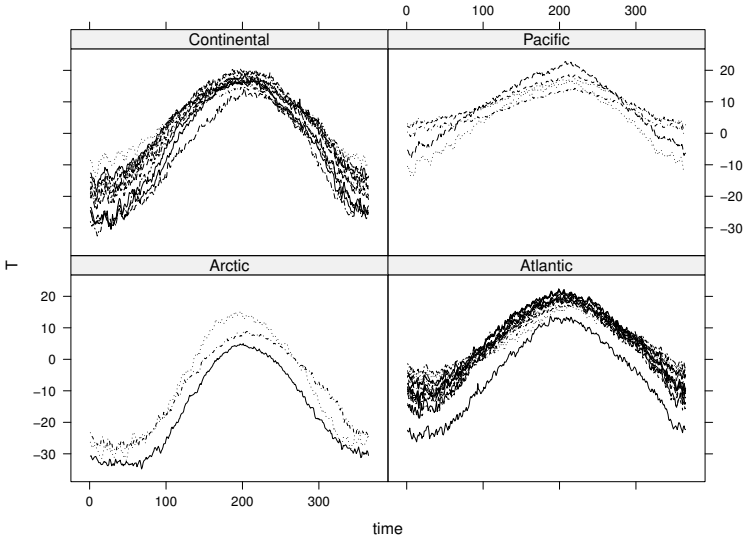

Figure 7.48 *Daily temperature data (degrees centigrade) for one year from 35 locations in Canada, classified by climatic zone.*

7.11.2 Function on scalar regression: Canadian weather

'The Canadian Weather Data' are a much overused example in functional data analysis, but provide a nice illustration of function on scalar regression. They are in the `CanWeather` dataset in `gamair`, and are shown in figure 7.48, which results from

```
require(gamair);require(lattice);data(canWeather)
xyplot(T~time|region,data=CanWeather,type="l",groups=place)
```

Treating each location's temperature profile as a function of time, we would like to model the temperature profiles using the scalar covariates `region` (one of four climate zones) and `latitude`. A possible model is

$$\mathtt{T}_i(\mathtt{time}) = f_{\mathtt{region}(i)}(\mathtt{time}) + f(\mathtt{time})\mathtt{latitude}_i + e_i(\mathtt{time})$$

i.e., there are five smooth functions of time to estimate, an intercept function for each climate zone, and a function giving the slope with latitude. It is assumed that the residual function $e_i(\mathtt{time})$ is a process that leads to AR1 Gaussian residuals when observed daily.

If we assume independent daily residuals then this model can be estimated using `gam`; however, the AR1 assumption requires either that we use `gamm` or `bam`. `bam` uses the simple computationally efficient AR residual handling of section 1.8.5 (p. 52): an AR1 correlation structure is assumed between consecutive observations in the fitting data, with correlation parameter given by the `rho` argument to `bam`. Breaks between autocorrelated sequences of residuals are indicated by the logical vector argument `AR.start` which is `TRUE` at the start of a new sequence of correlated

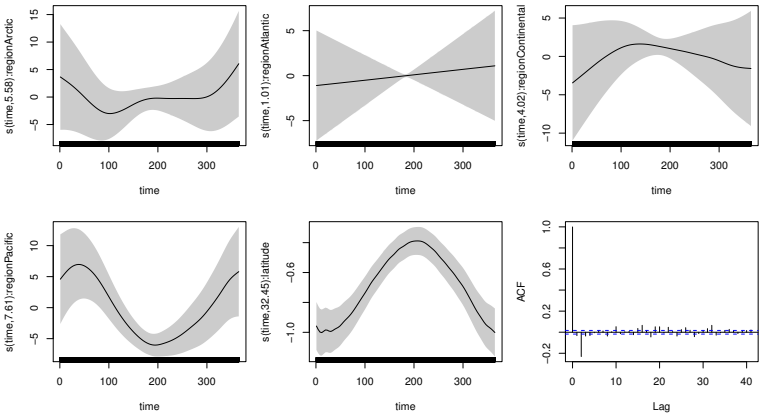

Figure 7.49 *The fitted intercept functions and the* `latitude` *slope function estimates for the function on scalar model of the Canadian Weather data. The lower right panel shows the ACF of the standardized residuals for the fitted model: the negative correlation at lag 2 shows that the AR1 model is certainly not perfect.*

residuals, and `FALSE` otherwise. To choose `rho` a sequence of values can be tried, and the value chosen that minimizes the AIC or negative REML score. Examination of the ACF for a fit with `rho=0` is usually sufficient to find a likely range for `rho`.

The following code fits the model for 10 ρ values evenly spaced between 0.9 and 0.99.** Notice that the intercept function smooths are centred, so the main effect of `region` is also needed.

```
aic <- reml <- rho <- seq(0.9,0.99,by=.01)
for (i in 1:length(rho)) {
  b <- bam(T ~ region + s(time,k=20,bs="cr",by=region) +
           s(time,k=40,bs="cr",by=latitude),
           data=CanWeather,AR.start=time==1,rho=rho[i])
  aic[i] <- AIC(b); reml[i] <- b$gcv.ubre
}
```

Both the AIC and negative REML score have a single minimum at 0.97 here, so this is the value chosen. Figure 7.49 shows the estimated smooth effects. It appears that, as expected, the annual cycle becomes more pronounced with increasing latitude. In practice the Atlantic zone is acting as the reference climate in the model fit, with the Pacific region generally showing warmer winter temperatures. The continental region shows some evidence for colder winters, while the Arctic appears to have slightly warmer winters than the linear latitude model would predict. The lower right plot is the ACF for the standardized residuals returned in `b$std.rsd`: if the AR1 model is perfect then the standardized residuals should appear uncorrelated. The correlation at lag 2 clearly indicates that the model is not perfect, and that something slightly more

**For much larger datasets or models, the option `discrete=TRUE` would be worthwhile – here it is only around twice as quick.

sophisticated would ideally be used. However the AR1 model is certainly much better than an un-correlated model, which gives a very much worse ACF.

7.12 Other packages

There are now far too many packages for smooth modelling in R to hope to present meaningful illustrations of their use without massively lengthening this book, so instead this section will just offer some pointers to some of what is available.

- The original back-fitting approach to GAMs (e.g., Hastie and Tibshirani, 1986, 1990, 1993) is available in Trevor Hastie's `gam` package: a port to R of the original `gam` in S-PLUS. The `SemiPar` package on the other hand focuses on the semi-parametric approach of the book by Ruppert et al. (2003).
- For the full spline smoothing approach (e.g., Wahba, 1990; Wang, 1998b,a; Gu, 2013; Gu and Kim, 2002; Gu and Wahba, 1991) packages `gss` and `assist` are available. `gss` focuses especially on smoothing spline ANOVA models and efficient computation. More recently the `bigspline` package offers yet more efficiency (Helwig and Ma, 2016).
- In a fully Bayesian context `R2BayesX` offers an R interface to the BayesX smooth modelling software (e.g., Fahrmeir et al., 2004; Kneib and Fahrmeir, 2006; Lang et al., 2014; Klein et al., 2015). The `INLA` package is available from `http://www.r-inla.org` and implements the approach of Rue et al. (2009) including the very flexible spatial smoothing methods opened up by Lindgren et al. (2011).
- Smooth distributional regression and more is covered by package `VGAM` (Yee and Wild, 1996) and the `gamlss` packages of Rigby and Stasinopoulos (2005).
- In addition there are many more packages offering related models or some smoothing: for example `scam` (Pya and Wood, 2015) provides shape constrained additive models, `refund` offers functional data analysis, `mboost` provides GAMS via boosting, the `survival` package allows spline effects, and so it goes on.

As one simple example, here is the code for fitting a version of the Chicago air-pollution model using the `gam` package (the covariates were summed over the 4 days preceding death, and degrees of freedom have to be chosen by the modeller).

```
> library(gam)
> bfm <- gam(death~s(time,df=140)+lo(o3,tmp,span=.1),
             family=poisson,control=gam.control(bf.maxit=150))
```

The results are shown in figure 7.50.

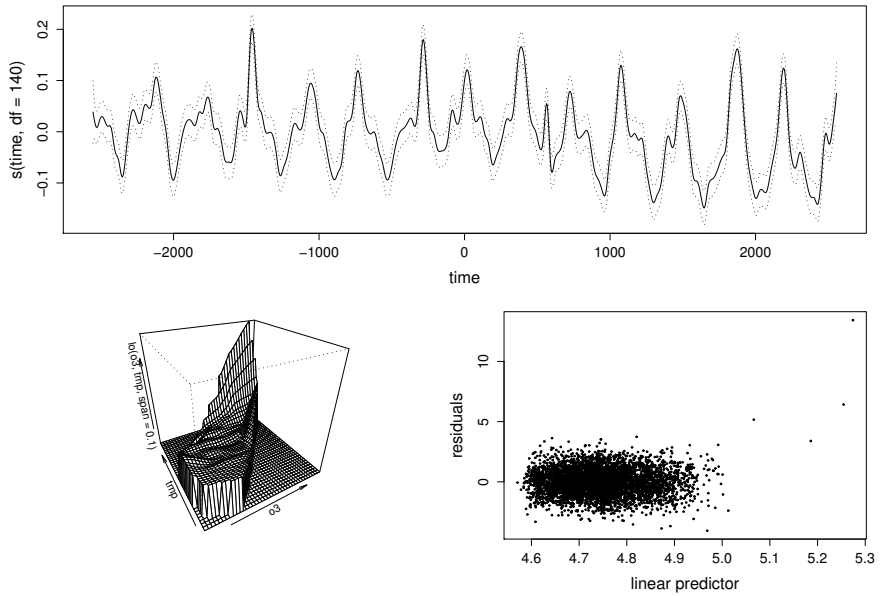

Figure 7.50 *A GAM fitted to the Chicago air pollution data, using* `gam` *from package* `gam`*. The upper panel is the estimated smooth of time while the lower left panel is a perspective plot of the estimated ozone – temperature interaction. The lower right plot is a residual plot.*

7.13 Exercises

1. This question re-examines the `hubble` data from Chapter 1.
   (a) Use `gam` to fit the model, $\mathtt{V}_i = f(\mathtt{D}_i) + \epsilon_i$, to the `hubble` data, where f is a smooth function and the ϵ_i are i.i.d. $N(0, \sigma^2)$. Does a straight line model appear to be most appropriate? How would you interpret the best fit model?
   (b) Examine appropriate residual plots and refit the model with more appropriate distributional assumptions. How are your conclusions from part (a) modified?
2. This question is about using `gam` for univariate smoothing, the advantages of penalized regression and weighting a smooth model fit. The `mcycle` data in the `MASS` package are a classic dataset in univariate smoothing, introduced in Silverman (1985). The data measure the acceleration of the rider's head, against time, in a simulated motorcycle crash.
   (a) Plot the acceleration against time, and use `gam` to fit a univariate smooth to the data, selecting the smoothing parameter by GCV (`k` of 30 to 40 is plenty for this example). Plot the resulting smooth, with partial residuals, but without standard errors.
   (b) Use `lm` and `poly` to fit a polynomial to the data, with approximately the same degrees of freedom as was estimated by `gam`. Use `termplot` to plot

the estimated polynomial and partial residuals. Note the substantially worse fit achieved by the polynomial, relative to the penalized regression spline fit.

(c) It's possible to overstate the importance of penalization in explaining the improvement of the penalized regression spline, relative to the polynomial. Use `gam` to refit an un-penalized thin plate regression spline to the data, with basis dimension the same as that used for the polynomial, and again produce a plot for comparison with the previous two results.

(d) Redo part (c) using an un-penalized cubic regression spline. You should find a fairly clear ordering of the acceptability of the results for the four models tried — what is it?

(e) Now plot the model residuals against time, and comment.

(f) To try and address the problems evident from the residual plot, try giving the first 20 observations the same higher weight, α, while leaving the remaining observations with weight one. Adjust α so that the variance of the first 20 residuals matches that of the remaining residuals. Recheck the residual plots.

(g) Experiment with the order of penalty used in the smooth. Does increasing it affect the model fit?

3. This question uses the `mcycle` data again, in order to more fully explore the influence matrix of a smoother.

(a) Consider a model for response data, $\mathbf{y}$, which has the influence matrix, $\mathbf{A}$, mapping the response data to the fitted values, given a smoothing parameter, λ, i.e., $\hat{\boldsymbol{\mu}} = \mathbf{A}\mathbf{y}$. Show that if we fit the same model, with the same λ, to the response data $\mathbf{I}_j$ (the j^{th} column of the identity matrix), then the resulting model fitted values are the j^{th} column of $\mathbf{A}$.

(b) Using the result from part (a) evaluate the influence matrix, $\mathbf{A}$, for the model fitted in question 2(a).

(c) What value do all the rows of $\mathbf{A}$ sum to? Why?

(d) Any smoothing model that can be represented as $\hat{\boldsymbol{\mu}} = \mathbf{A}\mathbf{y}$, simply replaces each value y_j by a weighted sum of neighbouring y_i values. For example, $\hat{\mu}_j = \sum_i A_{ji} y_i$, where the A_{ji} are the weights in the summation. It is instructive to examine the weights in the summation, so plot the weights used to form $\hat{\mu}_{65}$ against `mcycle$time`. What you have plotted is the 'equivalent kernel' of the fitted spline (at the 65^{th} datum).

(e) Plot all the equivalent kernels, on the same plot. Why do their peak heights vary?

(f) Now vary the smoothing parameter around the GCV selected value. What happens to the equivalent kernel for the 65^{th} datum?

4. This question follows on from questions 2 and 3, and examines the autocorrelation of residuals that results from smoothing.

(a) Based on the best model from question 2, produce a plot of the residual autocorrelation at lag 1 (average correlation between each residual and the previous residual, see `?acf`) against the model effective degrees of freedom. Vary the degrees of freedom between 2 and 40 by varying the `sp` argument to `gam`.

(b) Why do you see positive auto-correlation at very low EDF, and what causes this to reduce as the EDF increases?

(c) The explanation of why auto-correlation becomes negative is not quite so straightforward. Given the insight from question 3, that smoothers operate by forming weighted averages of neighbouring data, the cause of the negative autocorrelation can be understood by examining the simplest weighted average smoother: the k-point running mean. The fitted values from such a smoother are a simple average of the k nearest neighbours of the point in question (including the point itself). k is an odd integer.

i. Write out the form of a typical row, j, of the influence matrix, $\mathbf{A}$, of the simple running mean smoother. Assume that row j is not near the beginning or end of $\mathbf{A}$, and is therefore unaffected by edge effects.

ii. It is easy to show that the residuals are given by $\hat{\boldsymbol{\epsilon}} = (\mathbf{I}-\mathbf{A})\mathbf{y}$ and hence that their covariance matrix is $\mathbf{V}_{\hat{\epsilon}} = (\mathbf{I}-\mathbf{A})(\mathbf{I}-\mathbf{A})\sigma^2$. Find expressions for the elements of $\mathbf{V}_{\hat{\epsilon}}$ on its leading diagonal and on the sub- and super-diagonal, in terms of k. Again, only consider rows/columns that are unaffected by being near the edge of the data.

iii. What do your expressions suggest about residual auto-correlation as the amount of smoothing is reduced?

5. This question is about modelling data with seasonality, and the need to be very careful if trying to extrapolate with GAMs (or any statistical model). The data frame `co2s` contains monthly measurements of CO_2 at the south pole from January 1957 onwards. The columns are `co2`, the month of the year, `month`, and the cumulative number of months since January 1957, `c.month`. There are missing `co2` observations in some months.

(a) Plot the CO_2 observations against cumulative months.

(b) Fit the model $\texttt{co2}_i = f(\texttt{c.month}_i) + \epsilon_i$ where f is a smooth function and the ϵ_i are i.i.d. with constant variance, using the `gam` function. Use the `cr` basis, and a basis dimension of 300.

(c) Obtain the predicted CO_2 for each month of the data, plus 36 months after the end of the data, as well as associated standard errors. Produce a plot of the predictions with twice standard error bands. Are the predictions in the last 36 months credible?

(d) Fit the model $\texttt{co2}_i = f_1(\texttt{c.month}_i) + f_2(\texttt{month}_i) + \epsilon_i$ where f_1 and f_2 are smooth functions, but f_2 is cyclic (you will need to use the `knots` argument of `gam` to ensure that f_2 wraps appropriately: it's important to make sure that January is the same as January, not that December and January are the same!).

(e) Repeat the prediction and plotting in part (c) for the new model. Are the predictions more credible now? Explain the differences between the new results and those from part (c).

6. The data frame `ipo` contains data from Lowry and Schwert (2002) on the number of 'Intitial Public Offerings' (IPOs) per month in the US financial markets between 1960 and 2002. IPOs are the process by which companies go public:

ownership of the company is sold, in the form of shares, as a means of raising capital. Interest focuses on exploring the effect that several variables may have on numbers of IPOs per month, `n.ipo`. Of particular interest are the variables:

`ir` the average initial return from investing in an IPO. This is measured as percentage difference between the offer price of shares, and the share price after the first day of trading in the shares: this reflects by how much the offer price undervalues the company. One might expect companies to pay careful attention to this when deciding on IPO timing.

`dp` is the average percentage difference between the middle of the share price range proposed when the IPO is first filed, and the final offer price. This might be expected to carry information about the direction of changes in market sentiment.

`reg.t` the average time (in days) it takes from filing to offer. Obviously fluctuations in the length of time it takes to register have a direct impact on the number of companies making it through to offering in any given month.

Find a suitable possible model for explaining the number of IPOs in terms of these variables (as well as time, and time of year). Note that it is probably appropriate to look at lagged versions of `ir` and `dp`, since the length of the registration process precludes the number of IPOs in a month being driven by the initial returns in that same month. In the interests of interpretability it is probably worth following the advice of Kim and Gu (2004) and setting `gamma=1.4` in the `gam` call. Look at appropriate model checking plots, and interpret the results of your model fitting.

7. The data frame `wine` contains data on prices and growing characteristics of 25 high quality Bordeaux wines from 1952 to 1998 as reported in:
`http://schwert.ssb.rochester.edu/a425/a425.htm`.
`price` gives the average price as a percentage of 1961; `s.temp` is the average temperature (Celsius) over the summer preceding harvest, while `h.temp` is the average temperature at harvest; `w.rain` is the mm of rain in the preceding winter, while `h.rain` give the mm of rain in the harvest month; `year` is obvious. Create a GAM to model `price` in terms of the given predictors. Interpret the effects and use the model to predict the missing prices.

8. Sometimes rather unusual models can be expressed as GAMs. For example the data frame `blowfly` contains `counts` (not samples!) of adults in a laboratory population of blowflies, as reported in the classic ecological papers of Nicholson (1954a,b). One possible model for these data (basically Ellner et al., 1997) is that they are governed by the equation

$$\Delta n_{t+1} = f_b(n_{t-k})n_{t-k} - f_d(n_t)n_t + \epsilon_t,$$

where n_t is the population at time t, $\Delta n_{t+1} = n_{t+1} - n_t$, f_b and f_d are smooth functions and the ϵ_t are i.i.d. errors with constant variance. The idea is that the change in population is given by the difference between the birth and death rates plus a random term. *per capita* birth rates and death rates are smooth functions of populations, and the delayed effect of births on the adult population is because it takes around 12 days to go from egg to adult.

(a) Plot the blowfly data against time.

(b) Fit the proposed model, using `gam`. You will need to use `by` variables to do this. Comment on the estimates of f_b and f_d. It is worth making f_b and f_d depend on log populations, rather than the populations themselves, to avoid leverage problems.

(c) Using the beginning of the real data as a starting sequence, iterate your estimated model forward in time, to the end of the data, and plot it. First do this with the noise terms set to zero, and then try again with an error standard deviation of up to 500 (much larger and the population tends to explode). Comment on the results. You will need to artificially restrict the population to the range seen in the real data, to avoid problems caused by extrapolating the model.

(d) Why is the approach used for these data unlikely to be widely applicable?

9. This question is about creating models for calibration of satellite remote sensed data. The data frame `chl` contains direct ship based measurements of chlorophyll concentrations in the top 5 metres of ocean water, `chl`, as well as the corresponding satellite estimate `chl.sw` (actually a multi-year average measurement for the location and time of year), along with ocean depth, `bath`, day of year, `jul.day` and location, `lon`, `lat`. The data are from the world ocean database (see `http://seawifs.gsfc.nasa.gov/SEAWIFS/` for information on SeaWifs). `chl` and `chl.sw` do not correlate all that well with each other, probably because the reflective characteristics of water tend to change with time of year and whether the water is near the coast (and hence full of particulate matter) or open ocean. One way of improving the predictive power of the satellite observations might be to model the relationship between directly measured chlorophyll and remote sensed chlorophyll, viewing the relationship as a smooth one that is modified by other factors. Using ocean depth as an indicator for water type (near shore vs. open), a model something like

$$\mathbb{E}(\texttt{chl}_i) = f_1(\texttt{chl.sw}_i) f_2(\texttt{bath}_i) f_3(\texttt{jul.day}_i)$$

might be a reasonable starting point.

(a) Plot the response data and predictors against each other using `pairs`. Notice that some predictors have very skewed distributions. It is worth trying some simple power transformations in order to get a more even spread of predictors, and reduce the chance of a few points being overly influential: find appropriate transformations.

(b) Using `gam`, try modelling the data using a model of the sort suggested (but with predictors transformed as in part (a)). Make sure that you use an appropriate family. It will probably help to increase the default basis dimensions used for smoothing, somewhat (especially for `jul.day`). Use the `"cr"` basis to avoid excessive computational cost.

(c) In this sort of modelling the aim is to improve on simply using the satellite measurements as predictions of the direct measurements. Given the large number of data, it is easy to end up using rather complex models after quite a lot

of examination of the data. It is therefore important to check that the model is not over-fitting. A sensible way of doing this is to randomly select, say, 90% of the data to be fitted by the model, and then to see how well the model predicts the other 10% of data. Do this, using proportion deviance explained as the measure of fit/prediction. Note that the family you used will contain a function `dev.resids`, which you can use to find the deviance of any set of predictions.

10. This question follows on from question 11 Chapter 3, on soybean modelling.
    (a) Using `gamm`, replace the cubic function of time in the GLMM of question 10, with a smooth function of time. You should allow for the possibility that the varieties depend differently on time, and examine appropriate model checking plots.
    (b) Evaluate whether a model with or without a variety effect is more appropriate, and what kind of variety effect is most appropriate.
    (c) Explain why a model with separate smooths for the two different varieties is different to a model with a smooth for one variety and a smooth correction for the other variety.
11. The `med` dataset in `gamair` contains the equivalent of the mackerel survey data from section 7.5, but from 2010. Find a suitable model for these data, and investigate whether the Tweedie or negative binomial distributions offer a better model in this case.

Taylor & Francis
Taylor & Francis Group
http://taylorandfrancis.com