Chapter 10
Linear Models for Functional Responses

In this second chapter on the functional linear model, the dependent or response variable is functional. We first consider a situation in which all of the independent variables are scalar and in particular look at two functional analyses of variance.

When one or more of the independent variables is also function, we have two possible classes of linear models. The simpler case is called `concurrent`, where the value of the response variable $y(t)$ is predicted solely by the values of one or more functional covariates at the same time t. The more general case where functional variables contribute to the prediction for all possible time values s is briefly reviewed.

10.1 Functional Responses and an Analysis of Variance Model

While we often find functional covariates associated with scalar responses, there are also cases where the interest lies in the prediction of a functional response. We begin this chapter with two examples of *functional analysis of variance* (fANOVA), where variation in a functional response is decomposed into functional effects through the use of a scalar design matrix $\mathbf{Z}$. That is, in both of these examples, the covariates are all scalar.

10.1.1 Climate Region Effects on Temperature

In the Canadian weather data, for example, we can divide the weather stations into four distinct groups: Atlantic, Pacific, Prairie and Arctic. It may be interesting to know the effect of geographic location on the shape of the temperature curves. That is, we have a model of the form

$$y_i(t) = \beta_0(t) + \sum_{j=1}^{4} x_{ij}\beta_j(t) + \varepsilon_i(t) \tag{10.1}$$

where $y_i(t)$ is a *functional response*. In this case, the values of x_{ij} are either 0 or 1. If the 35 by 5 matrix $\mathbf{Z}$ contains these values, then the first column has all entries equal to 1, which codes the contribution of the Canadian mean temperature; the remaining four columns contain 1 if that weather station is in the corresponding climate zone and 0 otherwise. In order to identify the specific effects of the four climate zones, we have to add the constraint

$$\sum_{j=1}^{4} \beta_j(t) = 0 \text{ for all } t. \tag{10.2}$$

There are a number of methods of imposing this constraint. In this example we will do this by adding the above equation as an additional 36^{th} "observation" for which $y_{36}(t) = 0$.

We first create a list containing five indicator variables for the intercept term and each of the regions. In this setup, the intercept term is effectively the Canadian mean temperature curve, and each of the remaining regression coefficients is the perturbation of the Canadian mean required to fit a region's mean temperature. These indicator variables are stored in the `List` object `regionList`.

```
regions         = unique(CanadianWeather$region)
p               = length(regions) + 1
regionList      = vector("list", p)
regionList[[1]] = c(rep(1,35),0)
for (j in 2:p) {
  xj = CanadianWeather$region == regions[j-1]
  regionList[[j]] = c(xj,1)
}
```

The next step is to augment the temperature functional data object by a 36th observation that takes only zero values as required by (10.2).

```
coef     = tempfd$coef
coef36   = cbind(coef,matrix(0,65,1))
temp36fd = fd(coef36,tempbasis,tempfd$fdnames)
```

We now create functional parameter objects for each of the coefficient functions, using 11 Fourier basis functions for each.

```
betabasis = create.fourier.basis(c(0, 365), 11)
betafdPar = fdPar(betabasis)
betaList  = vector("list",p)
for (j in 1:p) betaList[[j]] = betafdPar
```

Now call `fRegress`, extract the coefficients and plot them, along with the predicted curves for the regions.

```
fRegressList = fRegress(temp36fd, regionList,
                        betaList)
betaestList  = fRegressList$betaestlist
regionFit    = fRegressList$yhatfd
regions      = c("Canada", regions)
par(mfrow=c(2,3),cex=1)
for (j in 1:p) plot(betaestList[[j]]$fd, lwd=2,
       xlab="Day (July 1 to June 30)",
       ylab="", main=regions[j])
plot(regionFit, lwd=2, col=1, lty=1,
       xlab="Day", ylab="",
       main="Prediction")
```

The five regression coefficients are shown in Figure 10.1; the final panel shows the predicted mean temperature curves for each of the four regions.

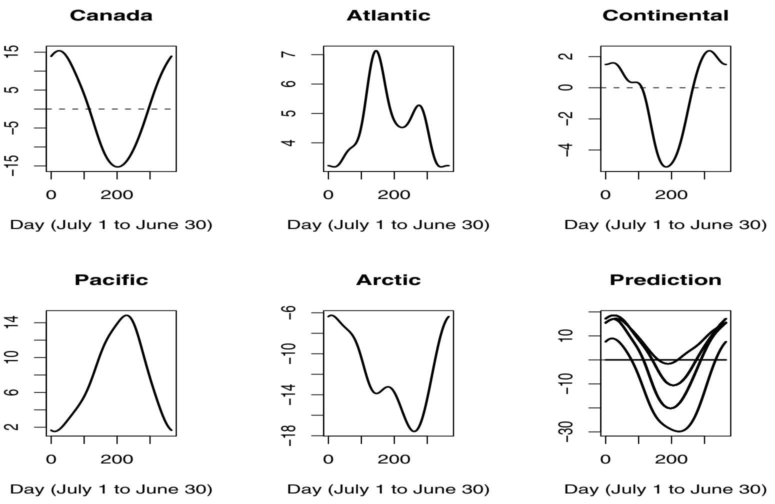

Fig. 10.1 The regression coefficients estimated for predicting temperature from climate region. The first panel is the intercept coefficient, corresponding to the Canadian mean temperature. The last panel contains the predicted mean temperatures for the four regions.

10.1.2 Trends in Seabird Populations on Kodiak Island

Winter abundance of 13 seabird species has been monitored annually in a number of bays on Kodiak Island, Alaska, since 1979 (Zwiefelhofer et al., 2008). Since 1986 these marine surveys have been conducted by the Kodiak National Wildlife Refuge using a standard protocol revisiting a fixed set of transects in each bay. The bird counts analyzed here are from two bays, Uyak and Uganik, both influenced by the waters of the Shelikof Strait. We focus on results from 1991 to 2005, less 1998 when the boat was in dry dock. We want to investigate potential differences in time trends of bird species that primarily eat fish compared to those that primarily eat shellfish and mollusks.

Figure 10.2 shows the base 10 logarithms of counts of the 13 species averaged over transects and sites, and separated according to diet. It is obvious that there are substantial differences in abundances over species that are consistent across the years of observation, and there is more variation in abundance among the fish-eating birds. The two mean functions suggest a slight tendency for the fish-eating birds to be increasing in abundance relative to what we see for shellfish and mollusk eaters, although this may be due to the sharply increasing trend for one fish-eating species. We will use `fRegress` to see how substantial this difference is.

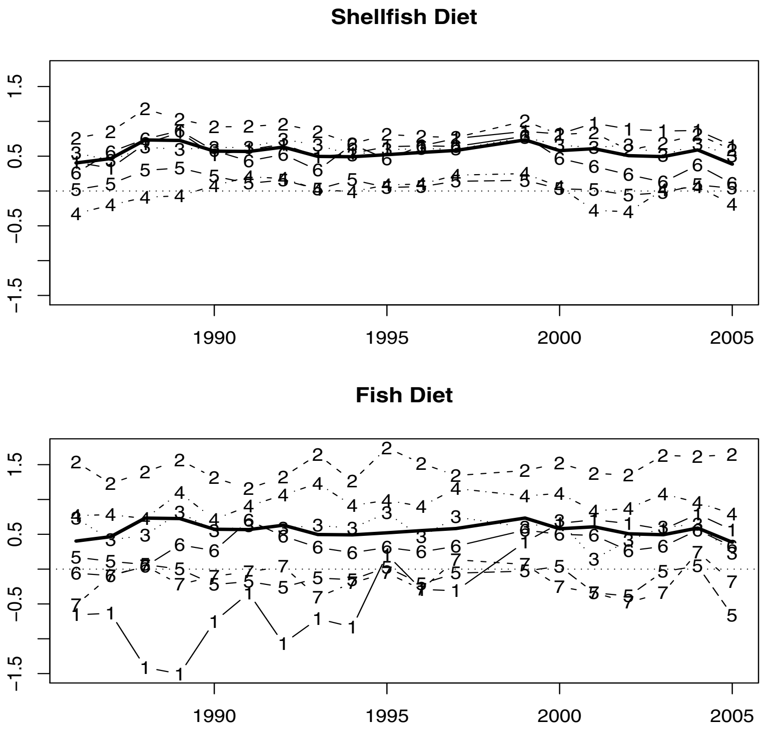

Fig. 10.2 The base 10 logarithms of seabird counts on Kodiak Island averaged over transects and sites. The top panel shows counts for birds who eat shellfish and mollusks, and the bottom shows counts for fish-eating birds. In each panel, the mean count taken across birds is shown as a heavy solid line.

We elected to fit the count data exactly using a polygonal basis, since we were less interested in estimating smooth trends for each species than we were in estimating the functional diet effect. By interpolating the data in this way, we were sure to retain all the information in the original data. The following two commands set up the polygonal basis and fit the data in the 19 by 26 matrix `logCounts2`, and `yearCode = c(1:12, 14:20)` (because no data were collected for 1986, year 13). In this matrix, the first 13 columns are for the Uganik site and the remaining 13 for the Uyak site, and each row corresponds to a year of observation.

```
birdbasis = create.polygonal.basis(yearCode)
birdList  = smooth.basis(yearCode,logCounts2,birdbasis)
birdfd    = birdList$fd
```

We analyze the time trend in the log mean counts as affected by the type of diet, with bird species nested within the diet factor. The model that we consider is

$$y_{ijk}(t) = \mu(t) + (-1)^i \alpha(t) + \beta_{ij}(t) + \varepsilon_{ijk}(t) \tag{10.3}$$

where $i = 1,2$ indexes food groups, $j = 1,\ldots,n_i$ indexes birds within a food group, and $k = 1,2$ indexes sites. The functional parameter $\mu(t)$ is the intercept or grand mean indicating the average trend for all birds. Parameter $\alpha(t)$ will indicate the time trend of the mean *difference* between the shellfish/mollusk-eating birds and the fish-eating birds, and multiplies a scalar covariate taking two values: 1 if an observation is for a shellfish/mollusk eater, and -1 otherwise. The 13 parameters $\beta_{ij}(t)$ are time trends for each bird, but represent deviations from $\mu(t)$ that are specific to a food group. This is achieved by imposing two constraint equations: $\sum_j \beta_{1j}(t) = 0$ and $\sum_j \beta_{2j}(t) = 0$ for all values of t. In each of these summations, index j takes values only within the corresponding food group. Finally, $\varepsilon_{ijk}(t)$ is the inevitable residual function required to balance each equation. The total number of equations is 28, being two blocks of 13 species plus one constraint for each food groups. There are two blocks, one for each bay or observation site.

To set up this model, we first define a variable for the diet effect, containing values 1 for a shellfish eater and -1 for a fish eater. This covariate effectively indicates the difference between being a shellfish eater and a fish eater.

```
foodindex = c(1,2,5,6,12,13)
foodvarbl = (2*rep(1:13 %in% foodindex, 2)-1)
```

Next we set up indicator variables for the effect of bird species; this is the identity matrix of order 13 stacked on top of a replicate of itself.

```
birdvarbl = diag(rep(1,13))
birdvarbl = rbind(birdvarbl, birdvarbl)
```

Now we set up the 26 by 15 design matrix **Z** for the regression analysis. Its first column is all 1's in order to code the intercept or grand mean effect. Its second column contains 1's in the first 13 rows corresponding to the shellfish diet, and -1's in the remaining rows. The remaining 13 columns code the bird effects.

```
Zmat0         = matrix(0,26,15)
Zmat0[,1]     = 1
Zmat0[,2]     = foodvarbl
Zmat0[,3:15] = birdvarbl
```

However, defining an effect for each bird in this way would make **Z** singular since the sum of these effects (columns 3:15) in each row is 1, and so is the intercept value. To correct for this, we need to force the sum of the bird effects within each diet group to add to zero. This requires two steps: we add two rows to the bottom of **Z** coding the sum of the bird effects for each diet group, and we add two corresponding functional observations to the 26 log count curves whose values are identically zero for all t.

```
Zmat       = rbind(Zmat0, matrix(0,2,15))
fishindex = (1:13)[-foodindex]
Zmat[27,foodindex+2] = 1
Zmat[28,fishindex+2] = 1
birdextfd = birdfd
birdextfd$coef =
    cbind(birdextfd$coefs, matrix(0,19,2))
```

Now we set up the arguments for `fRegress`. In these commands we insert each column in turn in matrix **Z** into the corresponding position in a list object `xfdlist`.

```
xfdlist = vector("list",15)
names(xfdlist) = c("const", "diet", birds)
for (j in 1:15) xfdlist[[j]] = Zmat[,j]
```

Now we define the corresponding list object `betalist`. We only want constant functions for the bird regression coefficient functions effects since there only the mean counts at the two sites available to estimate any bird's effect. However, for both the intercept and the food effect, we use a B-spline basis with knots at observation times. We determined the level of smoothing to be applied to the intercept and food regression functions by minimizing the cross-validated error sum of squares, as described in the next section, and the result was $\lambda = 10$.

```
betalist  = xfdlist
foodbasis = create.bspline.basis(rng,21,4,yearCode)
foodfdPar = fdPar(foodbasis, 2, 10)
betalist[[1]] = foodfdPar
betalist[[2]] = foodfdPar
conbasis  = create.constant.basis(rng)
for (j in 3:15) betalist[[j]] = fdPar(conbasis)
```

Next we fit the model using `fRegress`, which involves first defining lists for both the covariates (in this case all scalar) and a list of low-dimensional regression functions.

```
birdRegress = fRegress(birdextfd, xfdlist, betalist)
betaestlist = birdRegress$betaestlist
```

Figure 10.3 displays the regression functions for the intercept and food effects, along with 95% pointwise confidence intervals estimated by the methods described in Section 10.2.2. The trend in the intercept in the top panel models the mean trend over all species, and indicates a steady increase up to 1999 followed by some decline. The difference in the mean trends of the two food groups is shown in the bottom panel, and suggests a steady decline in the shellfish and mollusk eaters relative to the fish eaters starting in 1988. This is what we noticed in the log mean counts Figure 10.2.

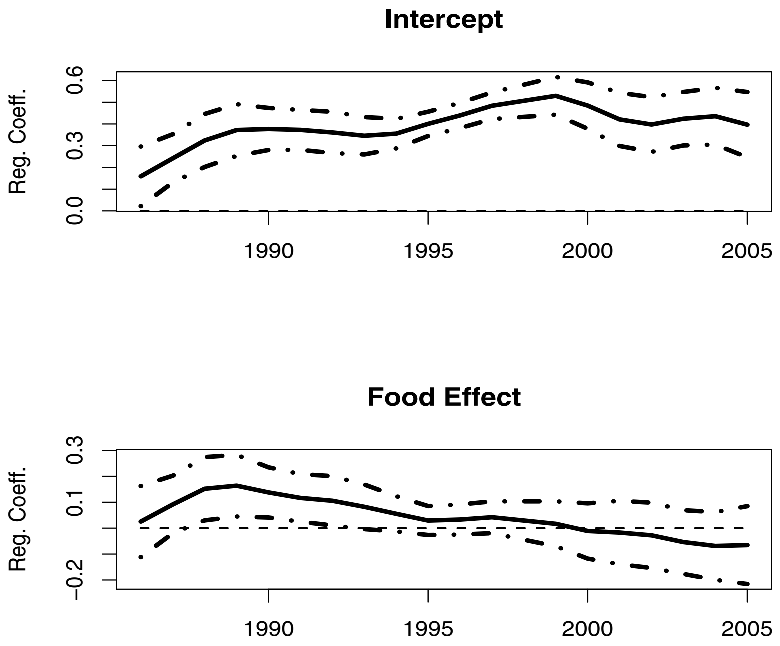

Fig. 10.3 The top panel shows the mean trend across all species, and the bottom panel shows the difference between being a shellfish/mollusk eater and a fish eater. The dashed lines indicate pointwise 95% confidence limits for these effects.

10.1.3 Choosing Smoothing Parameters

As for scalar response models, we would like to have a criterion for choosing any smoothing parameters that we use. Unfortunately, while ordinary cross-validation can be calculated for scalar response models without repeatedly re-estimating the model, this can no longer be done efficiently with functional response models. Here, we use the function `fRegress.CV` to compute the *cross-validated integrated squared error*:

$$\mathrm{CVISE}(\lambda) = \sum_{i=1}^{N} \int \left(y_i(t) - \hat{y}_i^{(-i)}(t) \right)^2 \mathrm{d}t$$

where $y^{(-i)}(t)$ is the predicted value for $y_i(t)$ when it is omitted from the estimation. In the following code, we search over a range of values for λ applied to both the intercept and the food effect:

```
loglam  = seq(-2,4,0.25)
SSE.CV1 = rep(NA,length(loglam))
betalisti = betaestlist
for (i in 1:length(loglam){
  for(j in 1:2)
    betalisti[[j]]$lambda = 10^loglam[i]
  CVi = fRegress.CV(birdRegress, xfdlist,
                        betalisti)
  SSE.CV1[i] = CVi$SSE.CV
}
```

This produces Figure 10.4, which indicates a unique minimum with λ approximately $\sqrt{10}$, although the discontinuities in the plot suggest that the cross-validated error sum of squares can be rather sensitive to non-smooth variation in the response functions as we defined them.

10.2 Functional Responses with Functional Predictors: The Concurrent Model

We can extend (10.1) to allow for functional covariates as follows:

$$y_i(t) = \beta_0(t) + \sum_{j=1}^{q-1} x_{ij}(t)\beta_j(t) + \varepsilon_i(t) \tag{10.4}$$

where $x_{ij}(t)$ may be a functional observation. Of course, x_{ij} may also be a scalar observation or a categorical indicator, in which case it can be simply interpreted as a function that is constant over time. Model (10.4) is called *concurrent* because it only relates the value of $y_i(t)$ to the value of $x_{ij}(t)$ at the same time points t. The intercept function $\beta_0(t)$ in effect multiplies a scalar covariate whose value is always one, and captures the variation in the response that does not depend on any of the covariate functions.

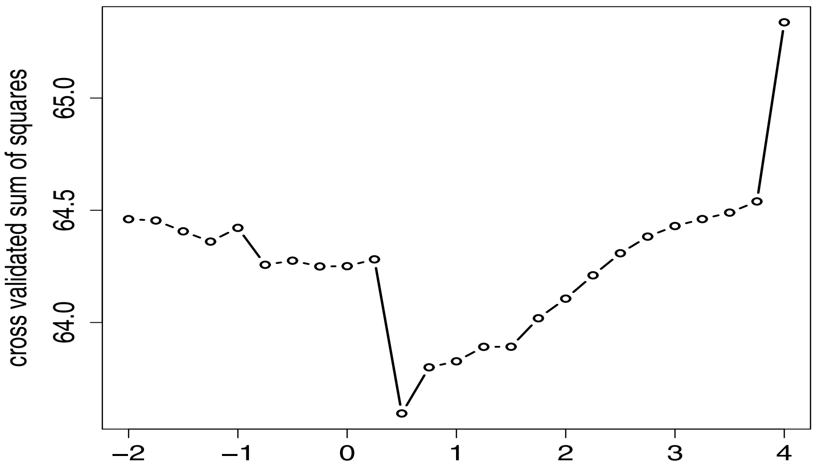

Fig. 10.4 The cross-validated integrated square errors for the bird data over a range of logarithms of λ.

10.2.1 Estimation for the Concurrent Model

As in ordinary regression, we must worry about redundancy or *multicollinearity* among the intercept and the functional (and scalar if present) covariates. Multicollinearity brings a host of problems, including imprecision in estimates due to rounding error, difficulty in discerning which covariates play an important role in predicting the dependent variable, and instability in regression coefficient estimates due to trade-offs between covariates in predicting variation in the dependent variable. When more than one functional covariate is involved, multicollinearity is often referred to as *concurvity*.

To better understand the multicollinearity problem, we look more closely at how the functional regression coefficients β_j are estimated by function `fRegress` by reducing the problem down to the solution of a set of linear equations. The coefficient matrix defining this linear system can then be analyzed to detect and diagnose problems with ill-conditioning and curvilinearity.

Let the N by q functional matrix $\mathbf{Z}$ contain these x_{ij} functions, and let the vector coefficient function β of length q contain each of the regression functions. The concurrent functional linear model in matrix notation is then

$$\mathbf{y}(t) = \mathbf{Z}(t)\beta(t) + \varepsilon(t) \;, \tag{10.5}$$

where $\mathbf{y}$ is a functional vector of length N containing the response functions. Let

$$\mathbf{r}(t) = \mathbf{y}(t) - \mathbf{Z}(t)\beta(t) \tag{10.6}$$

be the corresponding N-vector of residual functions. The weighted regularized fitting criterion is

$$\texttt{LMSSE}(\beta) = \int \mathbf{r}(t)'\mathbf{r}(t)\,\mathrm{d}t + \sum_j^p \lambda_j \int [L_j\beta_j(t)]^2\,\mathrm{d}t. \tag{10.7}$$

Let regression function β_j have the expansion

$$\beta_j(t) = \sum_k^{K_j} b_{kj}\theta_{kj}(t) = \theta_j(t)'\mathbf{b}_j$$

in terms of K_j basis functions θ_{kj}. In order to express (10.5) and (10.7) in matrix notation referring explicitly to these expansions, we need to construct some composite or supermatrices.

Defining $K_\beta = \sum_j^q K_j$, we first construct vector $\mathbf{b}$ of length K_β by stacking the vectors vertically, that is,

$$\mathbf{b} = (\mathbf{b}_1', \mathbf{b}_2', \ldots, \mathbf{b}_q')' \,.$$

Now assemble q by K_β matrix function $\Theta(t)$ as follows:

$$\Theta(t) = \begin{bmatrix} \theta_1(t)' & 0 & \cdots & 0 \\ 0 & \theta_2(t)' & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \theta_q(t)' \end{bmatrix} \,. \tag{10.8}$$

Then $\beta(t) = \Theta(t)\mathbf{b}$. With this, (10.6) can be rewritten as follows:

$$\mathbf{r}(t) = \mathbf{y}(t) - \mathbf{Z}(t)\Theta(t)\mathbf{b}$$

Next let $\mathbf{R}(\lambda)$ be the block diagonal matrix with jth block as follows:

$$\lambda_j \int [L_j\theta_j(t)]'[L_j\theta_j(t)]\mathrm{d}t.$$

Then (10.7) can be written as follows:

$$\texttt{LMSSE}(\beta) = \int [\mathbf{y}(t)'\mathbf{y}(t) - 2\mathbf{b}'\Theta(t)'\mathbf{Z}(t)'\mathbf{y}(t) + \mathbf{b}'\Theta(t)'\mathbf{Z}(t)'\mathbf{Z}(t)\Theta(t)\mathbf{b}]\mathrm{d}t$$

$$+\mathbf{b}'\mathbf{R}(\lambda)\mathbf{b}$$

If we differentiate this with respect to the coefficient vector $\mathbf{b}$ and set it to zero, we get the normal equations penalized least squares solution for the composite coefficient vector $\hat{\mathbf{b}}$:

$$[\int \Theta'(t)\mathbf{Z}'(t)\mathbf{Z}(t)\Theta(t)\,\mathrm{d}t + \mathbf{R}(\lambda)]\hat{\mathbf{b}} = [\int \Theta'(t)\mathbf{Z}'(t)\mathbf{y}(t)\,\mathrm{d}t] \,. \tag{10.9}$$

This is a linear matrix equation defining the scalar coefficients in vector $\hat{\mathbf{b}}$, $\mathbf{A}\hat{\mathbf{b}} = \mathbf{d}$, where the normal equation matrix is

$$\mathbf{A} = \int \Theta'(t)\mathbf{Z}'(t)\mathbf{Z}(t)\Theta(t)\,\mathrm{d}t + \mathbf{R}(\lambda) , \tag{10.10}$$

and the right-hand side vector of the system is

$$\mathbf{d} = \int \Theta'(t)\mathbf{Z}'(t)\mathbf{y}(t)\,\mathrm{d}t . \tag{10.11}$$

These equations are all given in terms of integrals of basis functions with functional data objects. In some cases, it is possible to evaluate them explicitly, but we will otherwise revert to numerical integration. In practice, numerical integration is both feasible and accurate (with reasonable choices for basis sets, etc.).

Concurrent linear models make up an important subset of all possible linear functional response models, especially for examining dynamics (see Chapter 11). However, they can be particularly restrictive; we discuss the general class of linear functional response models in Section 10.3.

10.2.2 Confidence Intervals for Regression Functions

Confidence intervals for regression coefficients are produced by first estimating an error covariance and observing that our estimates are linear functions of our data. In doing this, we account for both the variation of the smoothed $\mathbf{y}(t)$ about their predicted values and the residuals of the original smoothing process. The residual for the ith observation of the jth curve is

$$r_{ij} = y_{ij} - \mathbf{Z}_j(t_i)\beta(t_i).$$

When the y_{ij} all occur at a grid of times, we can produce an estimate

$$\Sigma_e^* = \frac{1}{N}\mathbf{r}\mathbf{r}'. \tag{10.12}$$

Where $\mathbf{r}$ is the matrix of residuals.

In the Seabird data, the error variance is calculated from

```
yhatmat = eval.fd(year, yhatfdobj)
rmatb   = logCounts2 - yhatmat
SigmaEb = var(t(rmatb))
```

With this estimate of Σ_e^*, we must consider the smoothing done to take the observations of the response $\mathbf{y}$ onto the space spanned by the response basis functions $\phi(t)$. Let $\mathbf{C}$ denote the matrix of regression coefficients for this representation, so $\mathbf{y}(t) = \mathbf{C}\phi(t)$. Substituting this into (10.9), we get

$$\begin{aligned}\hat{\mathbf{b}} &= \mathbf{A}^{-1}\Big[\int \Theta(t)'\mathbf{Z}(t)'\mathbf{C}\phi(t)\mathrm{d}t\Big]\\ &= \mathbf{A}^{-1}\Big[\int \phi(t)'\otimes(\Theta(t)'\mathbf{Z}(t)')\mathrm{d}t\Big]\ \mathrm{vec}(\mathbf{C})\\ &= \texttt{c2bMap}\ \mathrm{vec}(\mathbf{C}) \end{aligned} \tag{10.13}$$

where $\otimes$ is used to represent the Kronecker product. The explicit use of a basis expansion for $\mathbf{y}(t)$ allows the flexibility of modeling variation in $\mathbf{y}$ by itself or of including the original measurements of each response curve into the variance calculation.

We now require the matrix `y2cMap` that is used compute the regression coefficient matrix $\mathbf{C}$ from the original observations, $\mathbf{y}$. This this can be obtained from functions like `smooth.basis` or `smooth.basisPar`. The map (`c2bMap` `y2cMap`) now maps original observations directly to $\hat{\mathbf{b}}$. Therefore:

$$\mathrm{Var}\left[\hat{\mathbf{b}}\right] = \texttt{c2bMap y2cMap}\ \Sigma_e^*\ \texttt{y2cMap}'\ \texttt{c2bMap}'. \tag{10.14}$$

In the `fda` package, these intervals are created using `fRegress.stderr`. This requires the result of a call to `fRegress` along with the matrices `y2cMap` and Σ_e^*. The standard errors for the regression coefficients used to create Figure 10.3 are computed using the following code.

```
y2cMap = birdList2$y2cMap
stderrList = fRegress.stderr(birdRegress, y2cMap,
                                      SigmaEb)
betastderrlist = stderrList$betastderrlist
```

Finally we plot the results using the special purpose plotting function `plotbeta`.

When the original curves are not the result of smoothing data that have common observation times over curves, we can at least estimate confidence intervals based on the variation of the smoothed curves about the model predictions. To do this, we simply create pseudo data by evaluating the residual functions at a fine grid of points and calculating variance matrix from this. When we do this, the use of `y2cMap` above is no longer valid. Instead we replace it with a projection matrix that takes the pseudo data to the coefficients $\mathbf{C}$. This is simply $[\Phi(\mathbf{t})'\Phi(\mathbf{t})]^{-1}$, but we will not pursue this here.

10.2.3 Knee Angle Predicted from Hip Angle

The gait data displayed in Figure 1.6 are measurements of angle at the hip and knee of 39 children as they walk through a single gait cycle. The cycle begins at the point where the child's heel under the leg being observed strikes the ground. For plotting simplicity we run time here over the interval [0,20], since there are 20 times at which the two angles are observed. This analysis is inspired by the question, "How much control does the hip angle have over the knee angle?"

Figure 10.5 plots the mean knee angle along with its angular velocity and acceleration, and Figure 10.6 plots knee angle acceleration against velocity. We can see three distinct phases in knee angle of roughly equal durations:

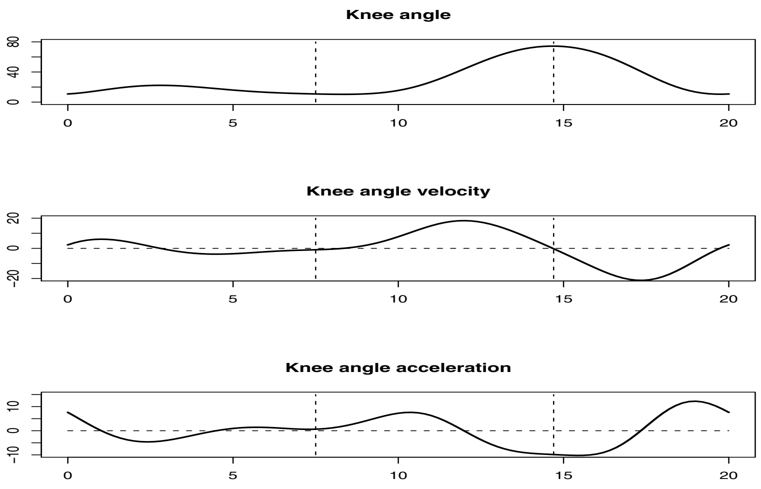

Fig. 10.5 Knee angle and its velocity and acceleration over a single gait cycle, which begins when the heel strikes the ground. The vertical dashed lines separate distinct phases in the cycle.

1. From time 0 to 7.5, the leg is bearing the weight of the child by itself, and the knee is close to being straight. This corresponds to the small loop in the cycle plot starting just before the marker "1" and up to the cusp.
2. From time 7.5 to time 14.7, the knee flexes in order to lift the foot off the ground, reaching a maximum mean angle of about 70 degrees.
3. From time 14.7 to time 20, the knee is extended to receive the load at the next heel-strike.

Together the second and third phases look like straightforward harmonic motion. A similar analysis of the hip motion reveals only a single harmonic phase. We wonder how the hip motion is coupled to knee motion.

Starting with functional data objects `kneefd` and `hipfd` for knee and hip angle, respectively, these commands execute a concurrent functional regression analysis where knee angle is fit by intercept and hip angle coefficient functions:

```
xfdlist      = list(rep(1,39), hipfd)
betafdPar    = fdPar(gaitbasis, harmaccelLfd)
betalist     = list(betafdPar,betafdPar)
fRegressList = fRegress(kneefd, xfdlist, betalist)
kneehatfd    = fRegressList$yhatfd
```

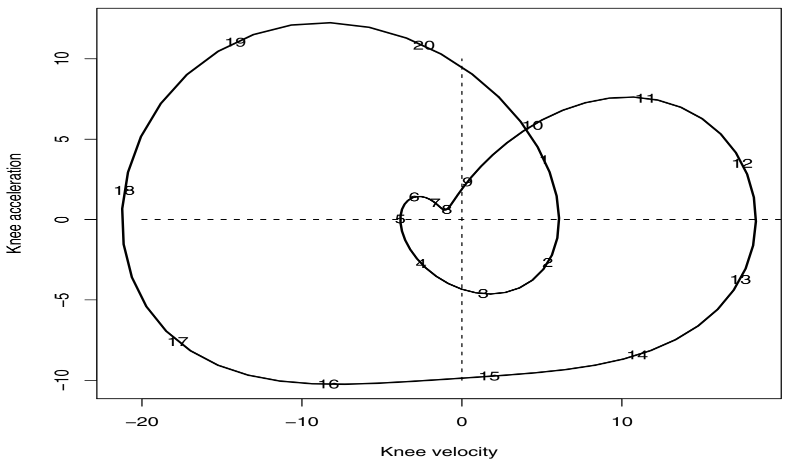

Fig. 10.6 A phase-plane plot of knee angle over a gait cycle. Numbers indicate indices of times of observation of knee angle.

```
betaestlist  = fRegressList$betaestlist
```

The intercept and hip regression coefficient functions are plotted as solid lines in Figure 10.7.

These commands compute the residual variance-covariance matrix estimate, which we leave as is rather than converting it to a diagonal matrix.

```
%kneemat    = eval.fd(gaittime, kneefd)
kneehatmat = eval.fd(gaittime, kneehatfd)
resmat.    = gait - kneehatmat
SigmaE     = cov(t(resmat.))
```

We also set up error sums of square functions for variation about both the model fit and mean knee angle. Then we compare the two via a squared multiple correlation function.

```
kneefinemat    = eval.fd(gaitfine, kneefd)
kneemeanvec    = eval.fd(gaitfine, kneemeanfd)
kneehatfinemat = eval.fd(gaitfine, kneehatfd)
resmat  = kneefinemat - kneehatfinemat
resmat0 = kneefinemat -
          kneemeanvec %*% matrix(1,1,ncurve)
SSE0 = apply((resmat0)^2, 1, sum)
SSE1 = apply(resmat^2, 1, sum)
Rsqr = (SSE0-SSE1)/SSE0
```

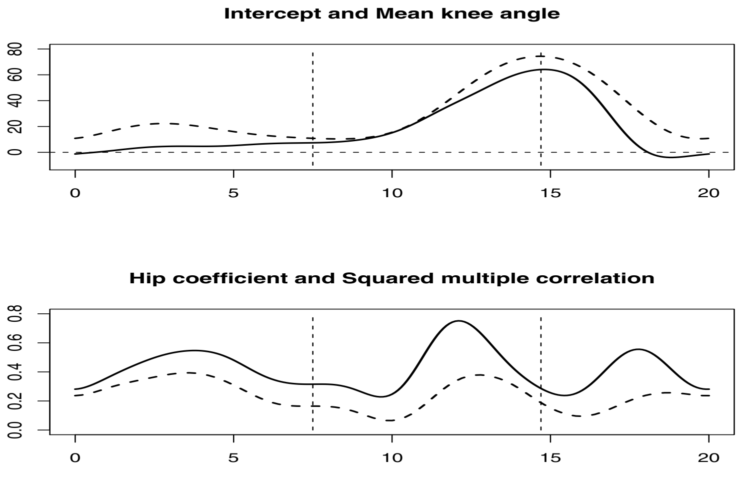

Fig. 10.7 The top panel shows as a solid line the intercept term in the prediction of knee angle from hip angle; the dashed line indicates the mean knee angle assuming no hip angle effect. The bottom panel shows as a solid line the functional regression coefficient multiplying hip angle in the functional concurrent linear model, and the dashed line shows the squared multiple correlation coefficient function associated with this model. Vertical dashed lines indicated boundaries between the three phases of the gait cycle.

The R^2 function is included in the second panel of Figure 10.7 as a dashed line. We see that it tracks pretty closely the variation in the hip regression coefficient.

The commands plot the intercept and hip regression coefficients with 95% confidence intervals, shown in Figure 10.8:

```
y2cMap = kneefdPar$y2cMap
fRegressList1 = fRegress(kneefd, xfdlist, betalist,
                                y2cMap, SigmaE)
fRegressList2 = fRegress.stderr(fRegressList1,
                                y2cMap, SigmaE)
betastderrlist = fRegressList2$betastderrlist
titlelist = list("Intercept", "Hip coefficient")
plotbeta(betaestlist, betastderrlist, gaitfine,
         titlelist)
```

We see that hip angle variation is coupled to knee angle variation in the middle of each of these three episodes, and the relation is especially strong during the middle flexing phase. It seems logical that a strongly flexed knee is associated with a sharper hip angle.

We can repeat these analyses to explore the relationship between knee and hip acceleration. This can be interesting because neural activation of these two muscle

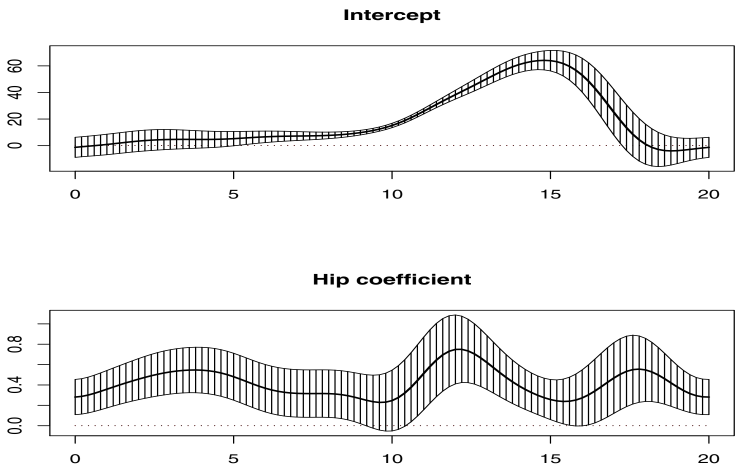

Fig. 10.8 The intercept and hip regression coefficient function for the gait cycle with 95% pointwise confidence intervals.

groups produces contraction, and contraction impacts acceleration directly by Newton's Second Law. Figure 10.9 shows the results of doing this. Now it is apparent that these two angles are coupled at a nearly evenly spaced series of seven points in the gait cycle. Given that a gait duration can be of the order of a second, it is striking to compare these plots with those of the handwriting data in Figure 8.6.

10.3 Beyond the Concurrent Model

The concurrent linear model only relates the value of a functional response to the current value of functional covariate(s). A more general version for a single functional covariate and an intercept is

$$y_i(t) = \beta_0(t) + \int_{\Omega_t} \beta_1(t,s)x_i(s)\mathrm{d}s + \varepsilon_i(t). \tag{10.15}$$

The bivariate regression coefficient function $\beta_1(s,t)$ defines the dependence of $y_i(t)$ on covariate $x_i(s)$ at each time t. In this case $x_i(s)$ need not be defined over the same range, or even the same continuum, as $y_i(t)$.

Set Ω_t in (10.15) contains the range of values of argument s over which x_i is considered to influence response y_i at time t, and the subscript t on this set indicates that this set can change from one value of t to another. For example, when both s and t are time, using $x_i(s)$ to predict $y_i(t)$ when $s > t$ may imply backwards causation.

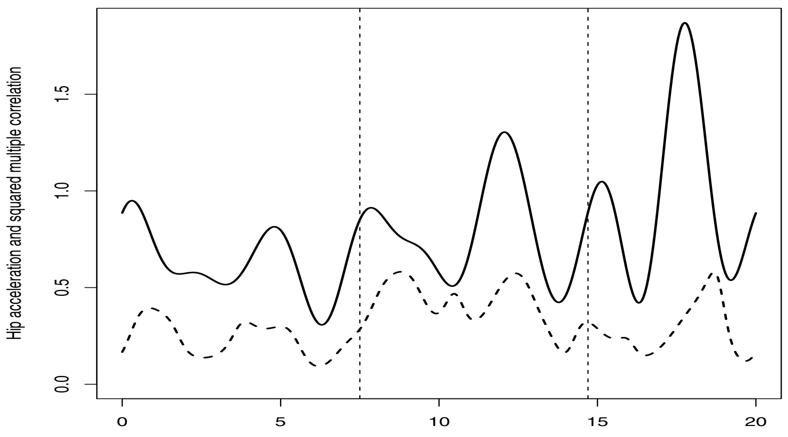

Fig. 10.9 The solid line shows the regression function multiplying hip angle acceleration in the prediction of knee angle acceleration, and the dashed line indicates the corresponding squared multiple coerrelation function.

To avoid this nonsense, we consider only values of x_i before time t. We may also add a restriction on how far back in time the influence of x_i on y_i can happen. This leads us to restrict the integral to

$$\Omega_t = \{s | t - \delta \leq s \leq t\} ,$$

where $\delta > 0$ specifies how much history is relevant to the prediction. Malfait and Ramsay (2003) described this as the *historical linear model.*

10.4 A Functional Linear Model for Swedish Mortality

We illustrate the estimation of (10.15) using Swedish life table data taken from census records in Sweden. The data are the number of deaths at each age for women born in each year from 1751 to 1914 and for ages 0 to 80. We will use the data up until 1884 in order to allow the extrapolation problem to be considered (see Section 10.10). Figure 10.10 displays the *log hazard rates* for four selected years. The log hazard rate is the natural logarithm of the ratio of the number of females who die at a specific age to the number of females alive with that age. These data were obtained from `http://mortality.org`. See also Chiou and Müller (2009) for another approach to modeling these data.

The hazard rate is greatest for infants and for the very elderly, and in most years attains its minimum in the early teens. The four curves indicate that the hazard rate decreases substantially as the health of the population improves over this period. However, there are localized features in each curve that reflect various historical events such as outbreaks of disease, war, and so on.

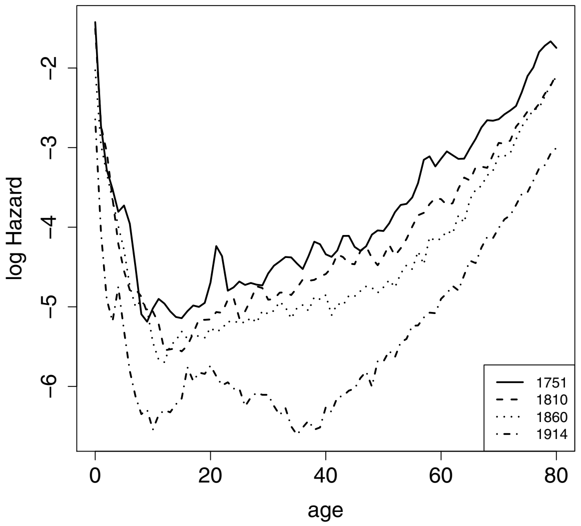

Fig. 10.10 Log hazard rates as a function of age for Swedish women born in the years 1751, 1810, 1860 and 1914. These data are derived from mortality tables at `http://mortality.org`.

Let $x_i(t)$ represent the log hazard rate at age t for birth year i. We propose the model

$$x_{i+1}(t) = \beta_0(t) + \int \beta_1(s,t)x_i(t)\mathrm{d}s + \varepsilon_i(t)\ . \tag{10.16}$$

That is, for any year from 1752 to 1894, we model the log hazard function for that year using as the functional covariate the log hazard curve for the preceding year. Assume that the response curves have been smoothed and represented as functional data object `NextYear`, and that the covariate curves are in functional data object `ThisYear`.

The regression function β_1 has the basis function expansion

$$\begin{aligned}\beta_1(s,t) &= \sum_{k=1}^{K_1}\sum_{\ell=1}^{K_2} b_{k\ell}\phi_k(s)\psi_\ell(t)\\ &= \phi'(s)\mathbf{B}\psi(t),\end{aligned} \tag{10.17}$$

where the coefficients for the expansion are in the K_1 by K_2 matrix $\mathbf{B}$. We therefore need to define two bases for β_1, as well as a basis for the intercept function β_0.

For a bivariate function such as $\beta_1(t,s)$ smoothness can be imposed by penalizing the s and t directions separately:

$$\texttt{PEN}_{\lambda_t,\lambda_s}(\beta_1(t,s)) = \lambda_1 \int [L_t\beta_1(t,s)]^2\, \mathrm{d}s\,\mathrm{d}t + \lambda_2 \int [L_s\beta_1(t,s)]^2\, \mathrm{d}s\,\mathrm{d}t\ , \qquad (10.18)$$

where linear differential operator L_s only involves derivatives with respect to s and L_t only involves derivatives with respect to t. We can also apply a penalty to the roughness of the intercept β_0.

The following code sets up a B-spline basis of order four with 23 basis functions. This is used to define functional parameter objects for β_0, $\beta_1(\cdot,t)$ and $\beta_1(s,\cdot)$. The second derivative is penalized in each case, but the smoothing parameter values vary as shown. The final statement assembles these three functional parameter objects into a `list` object to be supplied to function `linmod` as an argument.

```
betabasis = create.bspline.basis(c(0,80),23)
beta0Par  = fdPar(betabasis, 2, 1e-5)
beta1sPar = fdPar(betabasis, 2, 1e3)
beta1tPar = fdPar(betabasis, 2, 1e3)
betaList  = list(beta0Par, beta1sPar, beta1tPar)
```

Function `linmod` is invoked in R for these data by the command

```
linmodSmooth = linmod(NextYear, LastYear, betaList)
```

Figure 10.11 displays the estimated regression surface $\beta_1(s,t)$. The estimated intercept function β_0 ranged over values four orders of magnitude smaller than the response functions, and can therefore be considered to be essentially zero. The strong ridge one year off the diagonal, namely $\beta_1(s-1,s)$, indicates that mortality at any age is most strongly related to mortality at the previous year for that age less one. In other words, mortality is most strongly determined by age-specific factors like infectious diseases in infancy, accidents and violent death in early adulthood, and aging late in life. The height of the surface declines to near zero for large differences between s and t for this reason as well.

10.5 Permutation Tests of Functional Hypotheses

As was the case for scalar response models, we have so far focused on exploratory analyses. In the context of functional response models, it would again be useful to gauge the extent to which the estimated relationship can be distinguished from zero.

The type of questions that we are interested in are generalizations of common statistical tests and of common statistical models:

- Are two or more groups of functions statistically distinguishable?

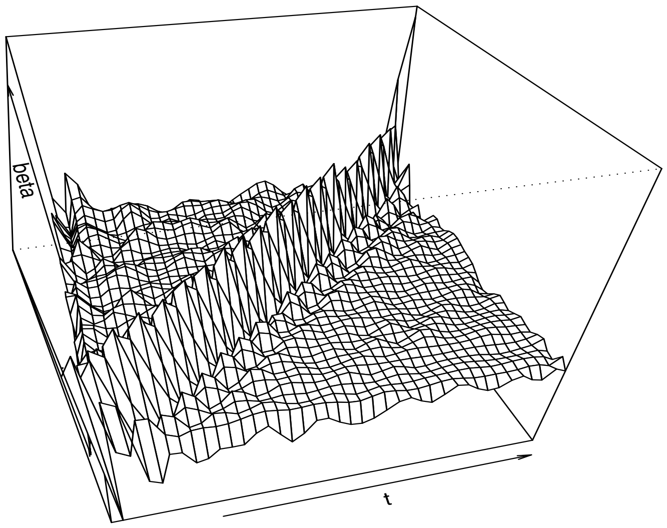

Fig. 10.11 The bivariate regression coefficient function $\beta_1(s,t)$ for the model (10.16) estimated from the 143 log hazard rate functions for the Swedish life table data. The ridge in $\beta_1(s,t)$ is one year off the diagonal.

- Are there statistically significant relationships among functional random variables?

Since functional data are inherently high dimensional, we again use permutation testing.

10.5.1 Functional *t*-Tests

Consider the Berkeley growth study data for both boys and girls in Figure 10.12. This plot suggests that boys generally become taller than girls. However, is this difference statistically significant? To evaluate this, we consider the absolute value of a t-statistic at each point:

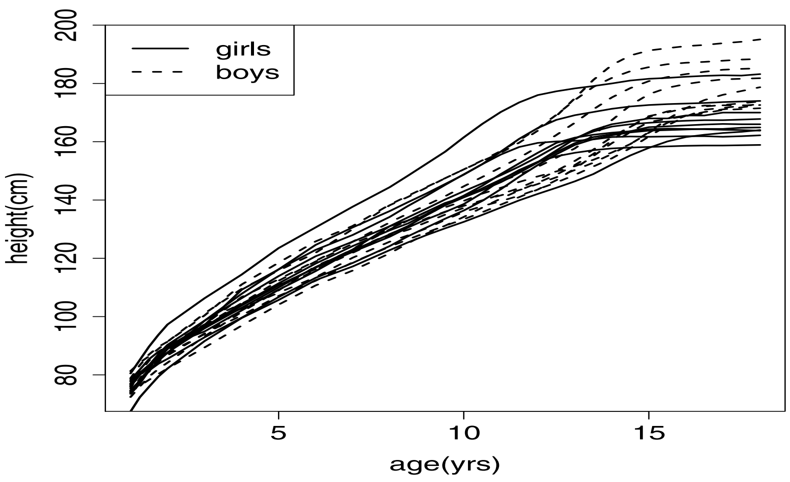

Fig. 10.12 The heights of the first ten boys and the first ten girls in the Berkeley growth study. We use a permutation test to evaluate whether the the growth curves for the two groups are different.

$$T(t) = \frac{|\bar{x}_1(t) - \bar{x}_2(t)|}{\sqrt{\frac{1}{n_1}\mathrm{Var}[x_1(t)] + \frac{1}{n_2}\mathrm{Var}[x_2(t)]}}. \tag{10.19}$$

This is plotted in the solid line in Figure 10.13. By itself, this provides a sense of the relative separation of the two groups of functions. However, a formal hypothesis test requires a value or statistic to test and a probability value indicating the result of the test. The test statistic that we use is the maximum value of the multivariate T-test, $T(t)$. To find a critical value of this statistic, we use a permutation test. We perform the following procedure:

1. Randomly shuffle the labels of the curves.
2. Recalculate the maximum of $T(t)$ with the new labels.

Repeating this many times allows a null distribution to be constructed. This provides a reference for evaluating the maximum value of the observed $T(t)$.

The following code executes a permutation test and generates the graphic in Figure 10.13. It uses a default value of 200 random shuffles, which is more than adequate for such a large difference as is shown, but might not suffice for more delicate effects.

```
tperm.fd(hgtmfd,hgtffd)
```

Here `hgtmfd` and `hgtffd` are functional data objects for the males and females in the study. It is apparent that there is little evidence for difference up to the age of around 12, about the middle of the female growth spurt, at which point the boys rapidly become taller. We can conclude that the main reason why boys end up taller than girls is that they get an extra couple of years of growth on the average.

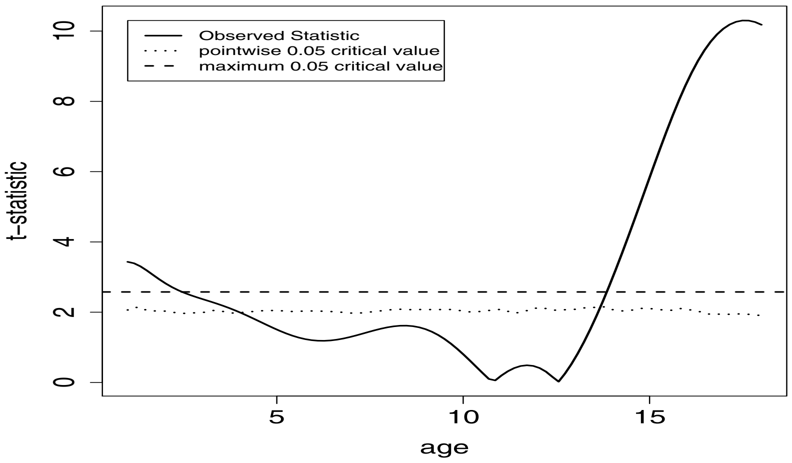

Fig. 10.13 A permutation test for the difference between girls and boys in the Berkeley growth study. The dashed line gives the permutation 0.05 critical value for the maximum of the t-statistic and the dotted the permutation critical value for the pointwise statistic.

10.5.2 Functional F-Tests

In the more general case of functional linear regression, the same approach can be applied. In this case, we define a functional version of the univariate F-statistic:

$$F(t) = \frac{\mathrm{Var}[\hat{\mathbf{y}}(t)]}{\frac{1}{n}\sum(y_i(t) - \hat{y}(t))^2} \tag{10.20}$$

where $\hat{\mathbf{y}}$ are the predicted values from a call to `fRegress`. Apart from a scale factor, this is the functional equivalent of the scalar F-statistic in multiple linear regression. It reduces to that for scalar-response models, as discussed in Section 9.5 above. As before, we reduce this to a single number by calculating $\max(F(t))$ and conducting a permutation test. In this case, we permute the response curves (or values), leaving the design unchanged. A test for no-effect of geographic region on temperature profile is conducted below. Figure 10.14 reports pointwise and maximal F-statistics and their corresponding permutation critical values for the temperature data.

```
F.res = Fperm.fd(temp36fd, regionList, betaList)
```

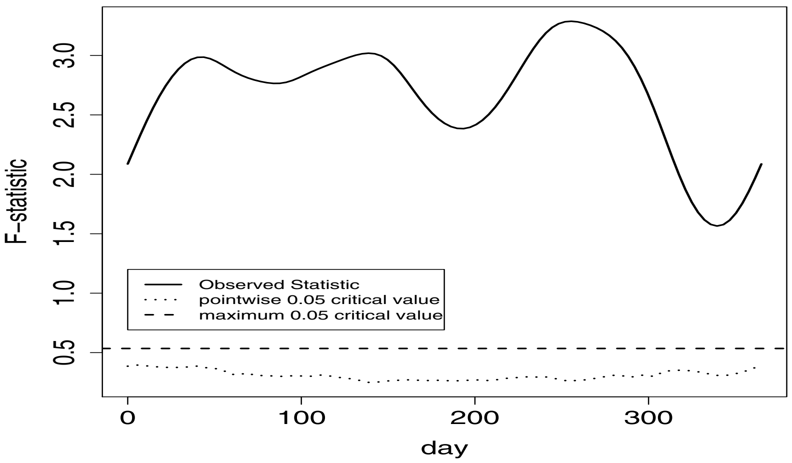

Fig. 10.14 A permutation test for a predictive relationship between geographic region and temperature profile for the Canadian weather data.

10.6 Details for R Functions `fRegress`, `fRegress.CV` and `fRegress.stderr`

10.6.1 Function `fRegress`

Because of its importance, we have set up a number of calling sequences for `fRegress`. These depend on the class of `y`, the first argument of the function. The first three cases concern the response being a vector of numbers, a functional data object or a functional parameter object. The last two allow the user to specify a model using a formula as in the R core function `lm`. The Matlab version can handle the first three cases, but not the last two:

numeric:
```
fRegress(y, xfdlist, betalist, wt=NULL,
    y2cMap=NULL, SigmaE=NULL, ...)
```

fd:
```
fRegress(y, xfdlist, betalist, wt=NULL,
    y2cMap=NULL, SigmaE=NULL, ...)
```

fdPar:
```
fRegress(y, xfdlist, betalist, wt=NULL,
     y2cMap=NULL, SigmaE=NULL, ...)
```

character:
```
fRegress(y, data=NULL, betalist=NULL,
   wt=NULL, y2cMap=NULL, SigmaE=NULL,
   method=c('fRegress', 'model'),
   sep='.', ...)
```

formula:
```
fRegress(y, data=NULL, betalist=NULL,
   wt=NULL, y2cMap=NULL, SigmaE=NULL,
   method=c('fRegress', 'model'),
   sep='.', ...)
```

The arguments are described as follows:

`y` The dependent variable object. It may be an object of five possible classes:

`scalar` A vector if the dependent variable is scalar.
`fd` A functional data object if the dependent variable is functional. A `y` of this class is replaced by `fdPar(y, ...)` and passed to `fRegress.fdPar`.
`fdPar` A functional parameter object if the dependent variable is functional, and if it is desired to smooth the prediction of the dependent variable.
`character or formula` A `formula` object or a `character` object that can be coerced into a `formula` providing a symbolic description of the model to be fitted satisfying the following rules: The left-hand side, `formula y`, must be either a numeric vector or a univariate object of class `fd` or `fdPar`. If the former, it is replaced by `fdPar(y, ...)`.

All objects named on the right-hand side must be either `numeric` or `fd` (functional data) or `fdPar`. The number of replications of `fd` or `fdPar` object(s) must match each other and the number of observations of `numeric` objects named, as well as the number of replications of the dependent variable object. The right-hand side of this `formula` is translated into `xfdlist`, then passed to another method for fitting (unless `method = 'model'`). Multivariate independent variables are allowed in a `formula` and are split into univariate independent variables in the resulting `xfdlist`. Similarly, categorical independent variables with `k` levels are translated into `k-1` contrasts in `xfdlist`. Any smoothing information is passed to the corresponding component of `betalist`.

`data` An optional `list` or `data.frame` containing names of objects identified in the `formula` or `character y`.
`xfdlist` A list of length equal to the number of independent variables (including any intercept). Members of this list are the independent variables. They can be objects of either of these two classes:

`scalar` A numeric vector if the independent variable is scalar.
`fd` A (univariate) functional data object.

In either case, the object must have the same number of replications as the dependent variable object. That is, if it is a scalar, it must be of the same length as the dependent variable, and if it is functional, it must have the same number of replications as the dependent variable. (Only univariate independent variables are currently allowed in xfdlist.)

betalist For the fd, fdPar, and numeric methods, betalist must be a list of length equal to length(xfdlist). Members of this list are functional parameter objects (class fdPar) defining the regression functions to be estimated. Even if a corresponding independent variable is scalar, its regression coefficient must be functional if the dependent variable is functional. (If the dependent variable is a scalar, the coefficients of scalar independent variables, including the intercept, must be constants, but the coefficients of functional independent variables must be functional.) Each of these functional parameter objects defines a single functional data object, that is, with only one replication.

For the formula and character methods, betalist can be either a list, as for the other methods, or NULL, in which case a list is created. If betalist is created, it will use the bases from the corresponding component of xfdlist if it is function or from the response variable. Smoothing information (arguments Lfdobj, lambda, estimate, and penmat of function fdPar) will come from the corresponding component of xfdlist if it is of class fdPar (or for scalar independent variables from the response variable if it is of class fdPar) or from optional ... arguments if the reference variable is not of class fdPar.

wt Weights for weighted least squares.

y2cMap The matrix mapping from the vector of observed values to the coefficients for the dependent variable. This is output by function smooth.basis. If this is supplied, confidence limits are computed, otherwise not.

SigmaE Estimate of the covariances among the residuals. This can only be estimated after a preliminary analysis with fRegress.

method A character string matching either fRegress for functional regression estimation or mode to create the argument lists for functional regression estimation without running it.

sep Separator for creating names for multiple variables for fRegress.fdPar or fRegress.numeric created from single variables on the right-hand side of the formula y. This happens with multidimensional fd objects as well as with categorical variables.

... Optional arguments.

These functions return either a standard fRegress fit object or a model specification:

fRegress fit A list of class fRegress with the following components:

y The first argument in the call to fRegress (coerced to class fdPar).
xfdlist The second argument in the call to fRegress.
betalist The third argument in the call to fRegress.

`betaestlist` A list of length equal to the number of independent variables and with members having the same functional parameter structure as the corresponding members of `betalist`. These are the estimated regression coefficient functions.

`yhatfdobj` A functional parameter object (class `fdPar`) if the dependent variable is functional or a vector if the dependent variable is scalar. This is the set of predicted by the functional regression model for the dependent variable.

`Cmatinv` A matrix containing the inverse of the coefficient matrix for the linear equations that define the solution to the regression problem. This matrix is required for function `fRegress.stderr` that estimates confidence regions for the regression coefficient function estimates.

`wt` The vector of weights input or inferred.

`df`, `OCV`, `gcv` Equivalent degrees of freedom for the fit, leave-one-out cross validation score for the model, and generalized cross validation score; only present if `class(y)` is `numeric`.

If `class(y)` is either `fd` or `fdPar`, the `fRegress` object returned also includes 5 other components:

`y2cMap` An input `y2cMap`.

`SigmaE` An input `SigmaE`.

`betastderrlist` An `fd` object estimating the standard errors of `betaestlist`.

`bvar` The covariance matrix defined in (10.14).

`c2bMap` A matrix converting the smoothing coefficients of the response variable into the regression coeffient, as defined in 10.13.

`model specification` The `fRegress.formula` and `fRegress. character` functions translate the `formula` into the argument list required by `fRegress. fdPar` or `fRegress.numeric` and then call the appropriate other `fRegress` function.

Alternatively, to see how the `formula` is translated, use the alternative `"model"` value for the argument `method`. In that case, the function returns a list with the arguments otherwise passed to other `fRegress` methods (functions) plus the following additional components:

`xfdlist0` A list of the objects named on the right hand side of `formula`. This will differ from `xfdlist` for any categorical or multivariate right-hand side object.

`type` The `type` component of any `fd` object on the right-hand side of `formula`.

`nbasis` A vector containing the `nbasis` components of variables named in `formula` having such components.

`xVars` An integer vector with all the variable names on the right-hand side of `formula` containing the corresponding number of variables in `xfdlist`. This can exceed 1 for any multivariate object on the right-hand side of class either `numeric` or `fd` as well as any categorical variable.

10.6.2 Function `fRegress.CV`

`fRegress.CV` performs leave-one-out cross-validation for a model computed with `fRegress`. In the case of a scalar response, ordinary and generalized cross-validation scores can be computed analytically without having to leave each observation out one at a time. These scores are already returned by `fRegress`. For functional response models, we must calculate the cross-validated scores by brute force. This can take some time.

The function call is

```
fRegress.CV(y, xfdlist, betalist, CVobs,...)
```

The arguments are as follows:

`y` The dependent variable object; either a vector, a functional data object or a functional parameter object.
`xfdlist` A list whose members are functional parameter objects specifying functional independent variables. Some of these may also be vectors specifying scalar independent variables.
`betalist` A list containing functional parameter objects specifying the regression functions and their level of smoothing.
`CVobs` A vector giving the indexes of the observations to leave out, one at a time, in computing the cross-validation scores. This defaults to all observations, but may be used to leave out artificial zero observations, as in the functional ANOVA models described in this chapter.
`...` Optional arguments not used by `fRegress.CV` but needed for superficial compatibability with `fRegress` methods.

The function returns a list object with components:

`SSE.CV` The sum of squared errors, or integrated squared errors.
`errfd.cv` Either a vector or a functional data object giving the cross-validated errors.

10.6.3 Function `fRegress.stderr`

The calling sequence is

```
fRegress.stderr(y, y2cMap, SigmaE, ...)
```

The arguments are as follows:

`y` The named list of length six that is returned from a call to function `fRegress`.
`y2cMap` A matrix that contains the linear transformation that takes the raw data values into the coefficients defining a smooth functional data object. Typically, this matrix is returned from a call to function `smooth.basis` that generates the dependent variable objects. If the dependent variable is scalar, this matrix is an identity matrix of order equal to the length of the vector.

`SigmaE` Either a matrix or a bivariate functional data object according to whether the dependent variable is scalar or functional, respectively. This object has a number of replications equal to the length of the dependent variable object. It contains an estimate of the variance-covariance matrix or function for the residuals.
`...` Optional arguments not used by `fRegress.stderr` but needed for superficial compatibability with `fRegress` methods.

The function returns a list object with these three components:

`betastderrlist` A list object of length equal to the number of independent variables. Each member contains a functional parameter object for the standard error of a regression function.
`bvar` A symmetric matrix containing sampling variances and covariances for the matrix of basis coefficients for the regression functions. These are stored column-wise in defining BVARIANCE.
`c2bMap` A matrix containing the mapping from response variable coefficients to coefficients for regression coefficients.

10.7 Details for Function `plotbeta`

The calling sequence is

```
plotbeta(betaestlist, betastderrlist, argvals=NULL,
         xlab="", ...)
```

The arguments are as follows:

`betaestlist` A list containing one or more functional parameter objects (class = `fdPar`) or functional data objects (class = `fd`).
`betastderrlist` A list containing functional data objects for the standard errors of the objects in `betaestlist`.
`argvals` A sequence of values at which to evaluate `betaestlist` and `betastderrlist`.
`xlab` x axis label.
`...` Additional plotting parameters passed to `plot`.

There is no return value.

10.8 Details for Function `linmod`

The calling sequence is

```
linmod(yfdobj, xfdobj, betaList)
```

The arguments are as follows:

`yfdobj` A functional data object for the response or dependent variable functions.

`xfdobj` A functional data object for the covariate or independent variable functions.

`betaList` A list object containing three functional parameter objects. The first is for the intercept term β_0 in (10.15), the second is for the bivariate regression function β_1 in (10.15) as a function of the first argument s, and the third is for β_1 as a function of the second argument t.

The function returns a list of length three with components as follows:

`beta0estfd` A functional data object for the estimated intercept.

`beta1estbifd` A bivariate functional data object for the bivariate regression function.

`yhatfdobj` A functional data object for the predicted response function.

10.9 Details for Functions `Fperm.fd` and `tperm.fd`

10.9.1 Function `Fperm.fd`

This function can be called with exactly the same calling sequence as `fRegress`, it has additional arguments which all have default values:

`nperm` Number of permutations to use in creating the null distribution.
`argvals` If `yfdPar` is a `fd` object, the points at which to evaluate the pointwise F-statistic.
`q` Critical upper-tail quantile of the null distribution to compare to the observed F-statistic.
`plotres` Argument to plot a visual display of the null distribution displaying the `q`th quantile and observed F-statistic.
... Additional plotting arguments that can be used with `plot`.

If `yfdPar` is a `fd` object, the maximal value of the pointwise F-statistic is calculated. The pointwise F-statistics are also returned. The default of setting `q = 0.95` is, by now, fairly standard. The default `nperm = 200` may be small, depending on the amount of computing time available. If `argvals` is not specified and `yfdPar` is a `fd` object, it defaults to 101 equally spaced points on the range of `yfdPar`.

If `plotres = TRUE` and `yfdPar` is a functional data object, a plot is produced giving the functional F-statistic along with 95th quantiles of the null distribution at each point and the 95th quantile of the null distribution of maximal F-values. If `yfdPar` is scalar, a histogram is plotted with the 95th quantile marked along with the observed statistic. The function returns a list with the following elements which may be used to reconstruct the plot.

`pval` The observed p-value of the permutation test.

`qval` The `q`th quantile of the null distribution.
`Fobs` The observed maximal F-statistic.
`Fnull` A vector of length `nperm` giving the observed values of the permutation distribution.
`Fvals` The pointwise values of the observed F-statistic.
`Fnullvals` The pointwise values of of the permutation observations.
`pvals.pts` Pointwise p-values of the F-statistic.
`qvals.pts` Pointwise `q`th quantiles of the null distribution.
`fRegressList` The result of `fRegress` on the observed data.
`argvals` Argument values for evaluating the F-statistic if `yfdPar` is a functional data object.

10.9.2 Function **`tperm.fd`**

This function carries out a permutation t-test for the difference between two groups of functional data objects. Its arguments are

`x1fd` and `x2fd` Functional data objects giving the two groups of functional observations.
`nperm` The number of permutations to use in creating the null distribution.
`q` Critical upper-tail quantile of the null distribution to compare to the observed t-statistic.
`argvals` If `yfdPar` is a `fd` object, the points at which to evaluate the pointwise t-statistic.
`plotres` Argument to plot a visual display of the null distribution displaying the `1-q`th quantile and observed t-statistic.

If `plotres=TRUE`, a plot is given showing the functional t-statistic, along with the critical values of the permutation distribution at each point and the permutation critical value of the maximal t-statistic. It returns a list with the objects necessary to recreate the plot:

`pval` The observed p-value of the permutation test.
`qval` The `q`th quantile of the null distribution.
`Tobs` The observed maximal t-statistic.
`Tnull` A vector of length `nperm` giving the observed values of the permutation distribution.
`Tvals` The pointwise values of the observed t-statistic.
`Tnullvals` The pointwise values of of the permutation observations.
`pvals.pts` Pointwise p-values of the t-statistic.
`qvals.pts` Pointwise `q`th quantiles of the null distribution.
`argvals` Argument values for evaluating the F-statistic if `yfdPar`is a functional data object.

10.10 Some Things to Try

The Swedish life table data consist of the log hazard rates (instantaneous risk of death) at ages 0 to 80 for Swedish women by birth year from 1751 to 1914. We want to develop a model for the way in which these have evolved over the years to 1894, and consider how well we can use this to forecast the hazard rate for women born in the year 1914.

1. Smooth the data appropriately. Explore these smooths – are there clearly evident features in how they change over time?
2. Create a functional linear model to predict the hazard curves from birth year for years 1751 to 1894. Choose smoothing parameters by cross-validation. Provide a plot of the error covariance. Plot the coefficient functions along with confidence intervals.
3. Examine the residuals from your model above. Are there any indications of lack of fit? If there are, construct an appropriately modified model. Plot the R-squared for both the linear model and the new one. Does there appear to be evidence for the effect of time on hazard curves?
4. Extrapolate your models to predict the hazard rate at 1914. How well does each do? Do they give better predictions than just the mean hazard curve?
5. Because the hazard curves are ordered in time, it is also possible to consider a *functional time series* model. Specifically, fit a model with the autoregressive structure:

   $$y_{i+1}(t) = \beta_0(t) + \beta_1(t)y_i(t) + \varepsilon_i(t).$$

   Report the results of your model. You should provide confidence intervals for $\beta_0(t)$ and $\beta_1(t)$. If you think the model will benefit from smoothing, do so. Does the model appear to fit well? Do you prefer this model or the model only using time as a predictor? Why?

10.11 More to Read

The concurrent linear model is closely related to the *varying coefficients model*. See Hastie and Tibshirani (1993), plus a large recent literature associated in Ramsay and Silverman (2005). A theoretical coverage of more general functional response models is given in Cuevas et al. (2002) as well as earlier papers by Cardot et al. (1999) and Ferraty and Vieu (2001). An elegant discussion of the ways in which the functional ANOVA can be treated is given in Brumback and Rice (1998) and associated discussion.