Chapter 5
Smoothing: Computing Curves from Noisy Data

The previous two chapters have introduced the Matlab and R code needed to specify basis function systems and then to define curves by combining these coefficient arrays. For example, we saw how to construct a basis object such as `heightbasis` to define growth curves and how to combine it with a matrix of coefficients such as `heightcoef` so as to define growth functional data objects such as were plotted in Figure 1.1.

We now turn to methods for computing these coefficients with more careful consideration of measurement error. For example, how do we compute these coefficients to obtain an optimal fit to data such as the height measurements for 54 girls in the Berkeley growth study stored in the 31 by 54 matrix that we name `heightmat`? Or how do we replace the rather noisy mean daily precipitation observations by smooth curves?

Two strategies are discussed. The simplest revisits the use of regression analysis that concluded Chapter 4, but now uses a special function for this purpose. The second and more elaborate strategy aims to miss nothing of importance in the data by using a powerful basis expansion, but avoids overfitting the data by imposing a penalty on the "roughness" of the function, where the meaning of "rough" can be adapted to special features of the application from which the data were obtained.

5.1 Regression Splines: Smoothing by Regression Analysis

We tend, perhaps rather too often, to default to defining data fitting as the minimization of the sum of squared errors or residuals,

$$\texttt{SSE}(x) = \sum_j^n [y_j - x(t_j)]^2. \tag{5.1}$$

When smoothing function x is defined as a basis function expansion (3.1), the least-squares estimation problem becomes

$$\mathtt{SSE}(\mathbf{c}) = \sum_j^n [y_j - \sum_k^K c_k \phi_k(t_j)]^2 = \sum_j^n [y_j - \phi(t_j)'\mathbf{c}]^2. \tag{5.2}$$

The approach is motivated by the error model

$$y_j = x(t_j) + \varepsilon_j = \mathbf{c}'\phi(t) + \varepsilon_j = \phi'(t_j)\mathbf{c} + \varepsilon_j \tag{5.3}$$

where the true errors or residuals ε_j are statistically independent and have a normal or Gaussian distribution with mean 0 and constant variance. Of course, if we look closely, we often see that this error model is too simple. Nevertheless, the least-squares estimation process can be defended on the grounds that it tends to give nearly optimal answers relative to "best" estimation methods so long as the true error distribution is fairly short-tailed and departures from the other assumptions are reasonably mild.

Readers will no doubt recognize (5.3) as the standard regression analysis model, along with its associated least-squares solution. Using matrix notation, let the n-vector $\mathbf{y}$ contain the n values to be fit, vector ε contain the corresponding true residual values, and n by k matrix Φ contain the basis function values $\phi_k(t_j)$. Then we have

$$\mathbf{y} = \Phi\mathbf{c} + \varepsilon$$

and the least-squares estimate of the coefficient vector $\mathbf{c}$ is

$$\hat{\mathbf{c}} = (\Phi'\Phi)^{-1}\Phi'\mathbf{y}. \tag{5.4}$$

R and Matlab already have the capacity to smooth data through their functions for regression analysis. Here is how we can combine these functions with the basis creation functions available in the `fda` package. Suppose that we want a basis system for the growth data with $K = 12$ basis functions using equally spaced knots. This can be accomplished in R with the following command:

```
heightbasis12 = create.bspline.basis(c(1,18), 12, 6)
```

If we evaluate the basis functions at the ages of measurement in vector object `age` by the command `basismat = eval.basis(age, heightbasis12)` (in R), then we have a 31 by 12 matrix of covariate or design values that we can use in a least-squares regression analysis defined by commands such as

```
heightcoef = lsfit(basismat, heightmat,
                   intercept=FALSE)$coef
heightcoef = basismat\heightmat
```

in R and Matlab, respectively. Spline curves fit by regression analysis are often referred to as *regression splines* in statistical literature.

However, the function `smooth.basis` (R) and `smooth_basis` (Matlab) are provided to produce the same results as well as much more without the need to explicitly evaluate the basis functions, through the R command

```
heightList = smooth.basis(age, heightmat,
                                    heightbasis12)
```

and the Matlab version

```
[fdobj, df, gcv, coef, SSE, penmat, y2cMap] = ...
    smooth_basis(age, heightmat, heightbasis12);
```

The R function `smooth.basis` returns an object `heightlist` of the `list` class, and the Matlab function `smooth_basis` returns all seven of its objects as an explicit sequence of variable names surrounded by square brackets. However, if we just wanted the first three returned objects as separate objects, in R we would have to extract them individually:

```
heightfd   = heightList$fd
height.df  = heightList$df
height.gcv = heightList$gcv
```

In Matlab, we would just request only the first three objects:

```
[fdobj, df, gcv] = ...
    smooth_basis(age, heightmat, heightbasis12);
```

In any case, the three most important returned objects are the following, where the names in bold type are used in each language to retrieve the objects:

fd An object of class `fd` containing the curves that fit the data.
df The degrees of freedom used to define the fitted curves.
gcv The value of the generalized cross-validation criterion: a measure of lack of fit discounted for degrees of freedom. If there are multiple curves, a vector is returned containing `gcv` values for each curve. (See Ramsay and Silverman (2005) for details.)

Notice that the coefficient estimate $\hat{\mathbf{c}}$ in (5.4) is obtained from the data in the vector $\mathbf{y}$ by multiplying this vector by a matrix, to which we give the text name `y2cMap`. We will use this matrix in many places in this book where we need to estimate the *variability* in quantities determined by $\hat{\mathbf{c}}$, so we here give it a name:

$$\texttt{y2cMap} = (\Phi'\Phi)^{-1}\Phi' \text{ so that } \hat{\mathbf{c}} = \texttt{y2cMap}\,\mathbf{y}. \tag{5.5}$$

Here is the corresponding R code for computing this matrix for the growth data:

```
age = growth$age
heightbasismat = eval.basis(age, heightbasis12)
y2cMap = solve(crossprod(heightbasismat),
                           t(heightbasismat))
```

In Matlab this last command would be

```
y2cMap = (heightbasismat'*heightbasismat) \ ...
          heightbasismat';
```

This code for the mapping matrix `y2cMap` only applies to regression-based smoothing. More general expressions for `y2cMap` include other term(s) that disappear with zero smoothing. This is important because as we change the smoothing, `y2cMap` changes, but $\hat{\mathbf{c}}$ is still the product of `y2cMap`, however changed, and the data.

While we are at it, we also will need what is often called the "hat" matrix, denoted by $\mathbf{H}$. This maps the data vector into the vector of fitted values

$$\mathbf{H} = \Phi(\Phi'\Phi)^{-1}\Phi' \text{ so that } \hat{\mathbf{y}} = \mathbf{H}\mathbf{y}. \tag{5.6}$$

The regression approach to smoothing data only works if the number K of basis functions is substantially smaller than the number n of sampling points. With the growth data, it seems that roughly $K = 12$ spline basis functions are needed to adequately smooth the growth data. Larger values of K will tend to undersmooth or overfit the data. Interestingly, after over a century of development of parametric growth curve models, the best of these also use about 12 parameters in this example.

Although regression splines are often adequate for simple jobs where only curve values are to be used, the instability of regression spline derivative estimates at the boundaries is especially acute. The next section describes a more sophisticated approach that can produce much better derivative results and also allows finer control over the amount of smoothing.

5.2 Data Smoothing with Roughness Penalties

The roughness penalty approach uses a large number of basis functions, possibly extending to one basis function per observation and even beyond, but at the same time imposing smoothness by penalizing some measure of function complexity. For example, we have already in the last chapter defined a basis system for the growth data called `heightbasis` that has 35 basis functions, even though we have only 31 observations per child. Would using such a basis system result in overfitting the data, as well as singularity problems on the computational side? That answer is, "Not if a positive penalty is applied to the degree to which the fit is not smooth."

5.2.1 Choosing a Roughness Penalty

We define a measure of the *roughness* of the fitted curve, and then minimize a fitting criterion that trades off curve roughness against lack of data fit.

Here is a popular way to quantify the notion "roughness" of a function. The square of the second derivative $[D^2x(t)]^2$ of a function x at argument value t is often called its *curvature* at t, since a straight line, which we all tend to agree has no curvature, has second derivative zero. Consequently, a measure of a function's roughness is the *integrated squared second derivative* or *total curvature*

$$\mathtt{PEN}_2(x) = \int [D^2x(t)]^2 \, \mathrm{d}t \; . \tag{5.7}$$

(Unless otherwise stated, all integrals in this book are definite integrals over the range of t.)

Penalty terms such as $\mathtt{PEN}_2(x)$ provide smoothing because wherever the function is highly variable, the square of the second derivative $[D^2x(t)]^2$ is large. We can apply this concept to derivative estimation as well. If we are interested in the second derivative D^2x of x, chances are that we want it to appear to be smooth. This suggests that we ought to penalize the curvature of the second derivative, that is, use the roughness measure

$$\mathtt{PEN}_4(x) = \int [D^4x(t)]^2 \, \mathrm{d}t \; . \tag{5.8}$$

But is "roughness" always related to the second derivative? Thinking a bit more broadly, we can define roughness as the extent to which a function departs from some baseline "smooth" behavior. For periodic functions of known period that can vary in level, such as mean temperature curves, the baseline behavior can be considered to be shifted sinusoidal variation,

$$x(t) = c_0 + a_1 \sin \omega t + b_1 \cos \omega t, \tag{5.9}$$

that is, represented by the first three terms in the Fourier series for some known $\omega = 2\pi/T$. If we compute $\omega^2 Dx + D^3x$ for such a simple function, we find that the result is exactly 0. We refer to the *differential operator* $L = \omega^2 D + D^3$ in Ramsay and Silverman (2005) as the *harmonic acceleration operator*. What happens when we apply this *harmonic acceleration operator* to higher-order terms in a Fourier series:

$$L[a_j \sin j\omega t + b_j \cos j\omega t] = \omega^2 j(1-j^2)[a_j \cos j\omega t - b_j \sin j\omega t]. \tag{5.10}$$

This expression is 0 for $j = 1$ and increases with the cube of j. This property suggests that the integral of the square of this *harmonic acceleration operator* may be a suitable measure of roughness for periodic data like the temperatures curves:

$$\mathtt{PEN}_L(x) = \int [Lx(s)]^2 \, ds \; . \tag{5.11}$$

When used on a finite Fourier series, this expression is proportional to $[j^2(1-j^2)^2]$. Thus, the term with $j = 1$ does not get penalized at all, and higher-order terms in the Fourier approximation receive substantially higher penalties.

Whatever roughness penalty we use, we add some multiple of it to the error sum of squares to define the compound fitting criterion. For example, using $\mathtt{PEN}_2(x)$ gives us the following:

$$F(\mathbf{c}) = \sum_j [y_j - x(t_j)]^2 + \lambda \int [D^2 x(t)]^2 \mathrm{d}t, \tag{5.12}$$

where $x(t) = \mathbf{c}'\phi(t)$. The *smoothing parameter* λ specifies the emphasis on the second term penalizing curvature relative to goodness of fit quantified in the sum of squared residuals in the first term. As λ moves from 0 upward, curvature becomes increasingly penalized. With λ sufficiently large, D^2x will be essentially 0. This in turn implies that x will be essentially a straight line = polynomial of degree one, order two, except possibly at a finite number of isolated points such as join points or knots of a B-spline. At the other extreme, $\lambda \rightarrow 0$ leaves the function x free to fit the data as closely as possible with the selected basis set, sometimes at the expense of some fairly wild variations in the approximating function.

It is usually convenient to plot and modify λ on a logarithmic scale. More generally, the use of a differential operator L to define roughness will result in $\lambda \rightarrow \infty$ forcing the fit to approach more and more closely a solution to the differential equation $Lx = 0$. If $L = D^m$, this solution will be a polynomial of order m (i.e., degree $m-1$). For the harmonic acceleration operator, this solution will be of the form (5.9). In this way, we can achieve an important new form of control over the smoothing process, namely by having the capacity to define the concept "smooth" in a way that is appropriate to the application.

5.2.2 *The Roughness Penalty Matrix* R

We can now provide an explicit form of the estimate of the coefficient vector $\hat{\mathbf{c}}$ for roughness penalty smoothing that is the counterpart of (5.4) for regression smoothing. The general version of the roughness penalized fitting criterion (5.12) is

$$F(\mathbf{c}) = \sum_j [y_j - x(t_j)]^2 + \lambda \int [Lx(t)]^2 \mathrm{d}t. \tag{5.13}$$

If we substitute the basis expansion $x(t) = \mathbf{c}'\phi(t) = \phi'(t)\mathbf{c}$ into this equation, we get

$$F(\mathbf{c}) = \sum_j [y_j - \phi'(t_j)\mathbf{c}]^2 + \lambda \mathbf{c}'[\int L\phi(t) L\phi'(t) \mathrm{d}t]\mathbf{c}. \tag{5.14}$$

Now we define the order K symmetric roughness penalty matrix as

$$\mathbf{R} = \int \phi(t)\phi'(t)\mathrm{d}t. \tag{5.15}$$

With this defined, it is a relatively easy exercise in matrix algebra to work out that

$$\hat{\mathbf{c}} = (\Phi'\Phi + \lambda \mathbf{R})^{-1}\Phi'\mathbf{y}. \tag{5.16}$$

From here we can define the matrix `y2cMap` that we will use in Chapter 6 for computing confidence regions about estimated curves:

$$\texttt{y2cMap} = (\Phi'\Phi + \lambda \mathbf{R})^{-1}\Phi'. \tag{5.17}$$

The corresponding hat-matrix is now

$$\mathbf{H} = \Phi(\Phi'\Phi + \lambda \mathbf{R})^{-1}\Phi'. \tag{5.18}$$

But how is one to compute matrix **R** in either language? This is taken care of in the function `eval.penalty` in R and `eval_penalty` in Matlab. These functions require two arguments:

`basisobj` A functional basis object of the `basisfd` class in R and `basis` class in Matlab.
`Lfdobj` A linear differential operator object of the `Lfd` class.

In the case of the harmonic accelerator operator, we can calculate the roughness penalty matrix `Rmat` in R by

```
    Rmat = eval.penalty(tempbasis, harmaccelLfd)
```

We hasten to add that most routine functional data analyses will not actually need to calculate roughness penalty matrices, since this happens inside functions such as `smooth.basis`. Computing **R** can involve numerical approximations to the integrals involved in (5.15). However, for a spline basis, if L is a power of D, then the integrals are analytically available and evaluated to within machine precision.

5.2.3 The Smoothing or "Hat" Matrix and Degrees of Freedom

The values $x(t_j), j = 1, \ldots, n$ defined by minimizing criterion (5.14) are critical for a detailed analysis of how well alternative choices λ work for fitting the data values y_j. Let us denote these by the vector $\hat{\mathbf{x}}$ and the corresponding data values by $\mathbf{y}$. It turns out (see Ramsay and Silverman (2005) for details) that $\hat{\mathbf{x}}$ has the following *linear* relationship to $\mathbf{y}$:

$$\hat{\mathbf{x}} = \mathbf{H}(\lambda)\mathbf{y}. \tag{5.19}$$

The *smoothing matrix* $\mathbf{H}(\lambda)$ is square, symmetric and of order n and, needless to say, a function of λ. It has many uses, among which is that a measure of the effective degrees of freedom of the fit defined by λ is defined by

$$\texttt{df}(\lambda) = \text{trace}[\mathbf{H}(\lambda)], \tag{5.20}$$

and the associated degrees of freedom for error is $n - \texttt{df}(\lambda)$.

As $\lambda \to 0, \texttt{df}(\lambda) \to \min(n, K)$, where n = the number of observations and K = the number of basis functions. Similarly, as $\lambda \to \infty, \texttt{df}(\lambda) \to m$, where m is the order of the highest derivative used to define the roughness penalty.

5.2.4 Defining Smoothing by Functional Parameter Objects

Going beyond the smoothing problem, we need the general capacity in functional data analysis to impose smoothness on estimated *functional parameters*, of which the smoothing curve is only one example. We now explain how this is made possible in the two programming languages.

A roughness penalty is defined by constructing a *functional parameter object* consisting of:

- a basis object,
- a derivative order m or a differential operator L to be penalized and
- a smoothing parameter λ.

We put these elements together by using the `fdPar` class in either language and the function `fdPar` to construct an object of that class.

The following R commands do two things: First they set up an order six B-spline basis for smoothing the growth data using a knot at each age. Then they define a functional parameter object that penalizes the roughness of growth acceleration by using the fourth derivative in the roughness penalty. The smoothing parameter value that we have found works well here is $\lambda = 0.01$.

```
norder = 6
nbasis = length(age) + norder - 2
heightbasis = create.bspline.basis(c(1,18),
                             nbasis, norder, age)
heightfdPar = fdPar(heightbasis, 4, 0.01)
```

The data are in array `heightmat`. In Chapter 4, these data were passed to `smooth.basis` with a basis object as the third argument. Here, we will use the functional parameter object `heightfdPar` as the third argument:

```
heightfd = smooth.basis(age, heightmat,
                              heightfdPar)$fd
```

Notice that we set up a functional data object `heightfd` directly by using the suffix `$fd`. In Matlab, we would use

```
heightfd = smooth_basis(age, heightmat, heightfdPar)
```

5.2.5 Choosing Smoothing Parameter λ

The *generalized cross-validation* measure GCV developed by Craven and Wahba (1979) is designed to locate a best value for smoothing parameter λ. The criterion is

$$\mathtt{GCV}(\lambda) = \left(\frac{n}{n - df(\lambda)}\right)\left(\frac{\mathtt{SSE}}{n - df(\lambda)}\right) . \tag{5.21}$$

Notice that this is a twice-discounted mean square error measure. The right factor is the unbiased estimate of error variance σ^2 familiar in regression analysis, and thus represents some discounting by subtracting $df(\lambda)$ from n. The left factor further discounts this estimate by multiplying by $n/(n - df(\lambda))$.

Figure 5.1 shows how the generalized cross-validation (GCV) criterion varies as a function of $\log_{10}(\lambda)$ for the entire female Berkeley growth data. Matlab code for generating the plotted values is

```
loglam   = -6:0.25:0;
gcvsave = zeros(length(loglam),1);
dfsave  = gcvsave;
for i=1:length(loglam)
  lambdai   = 10^loglam(i);
  hgtfdPari = fdPar(heightbasis, 4, lambdai);
  [hgtfdi, dfi, gcvi] =
      smooth_basis(age, hgtfmat, hgtfdPari);
  gcvsave(i) = sum(gcvi);
  dfsave(i)  = dfi;
end
```

The minimizing value of λ is about 10^{-4}, and at that value `df`$(\lambda) = 20.2$. In fact, the value $\lambda = 10^{-4}$ is rather smaller than the value of 10^{-2} that we chose to work with in our definition of the `fdPar` object in Section 5.2.4, for which `df`$(\lambda) = 12.7$. We explain our decision in Section 5.3, and recommend a cautious and considered approach to choosing the smoothing parameter rather than relying solely on automatic methods such as GCV minimization.

GCV values often change slowly with $\log_{10}\lambda$ near the minimizing value, so that a fairly wide range of λ values may give roughly the same GCV value. This is a sign that the data are not especially informative about the "true" value of λ. If so, it is not worth investing a great deal of effort in precisely locating the minimizing value, and simply plotting GCV over a mesh of $\log_{10}\lambda$ might be sufficient. Plotting the function `GCV`(λ) in any case will inform us about the curvature of near its minimum. If the data are not telling us all that much about λ, then it is surely reasonable to use your judgment in working with values which seem to provide more useful results than the minimizing value does. Indeed, Chaudhuri and Marron (1999) argue persuasively for inspecting data smooths over a range of λ values in order to see what is revealed at each level of smoothing. However, if a more precise value seems important, the function `lambda2gcv` can be used as an argument in an optimization function that will return the minimizing value.

5.3 Case Study: The Log Precipitation Data

The `fda` package for R includes `CanadianWeather` data, which includes the base 10 logarithms of the average annual precipitation in millimeters (after replacing

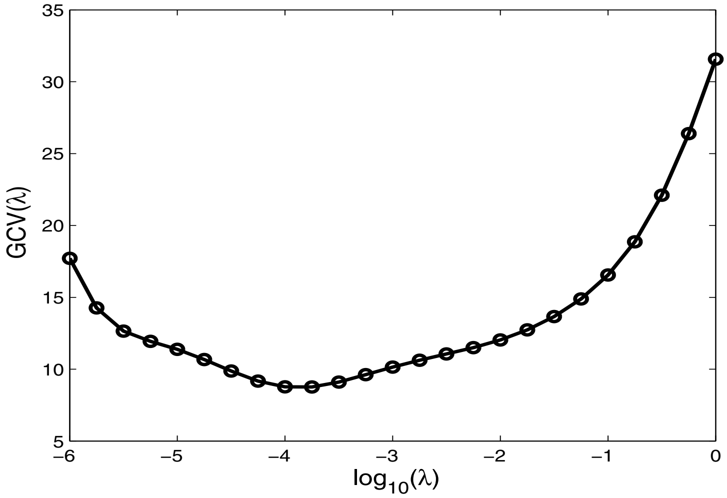

Fig. 5.1 The values of the generalized cross-validation or GCV criterion for choosing the smoothing parameter λ for fitting the female growth curves.

zeros with 0.05) for each day of the year at 35 different weather stations. We put these data in `logprecav`, shifted to put winter in the middle, so the year begins with July 1 and ends with June 30:

```
logprecav = CanadianWeather$dailyAv[
          dayOfYearShifted, , 'log10precip']
```

Next we set up a saturated Fourier basis for the data:

```
dayrange  = c(0,365)
daybasis  = create.fourier.basis(dayrange, 365)
```

We will smooth the data using a harmonic acceleration roughness penalty that penalizes departures from a shifted sine, $x(t) = c_1 + c_2 \sin(2\pi t/365) + c_3 \cos(2\pi t/365)$. Here we define this penalty. The first command sets up a vector containing the three coefficients required for the linear differential operator, and the second uses function `vec2Lfd` to convert this vector to the linear differential operator object `harmaccelLfd`.

```
Lcoef        = c(0,(2*pi/diff(dayrange))^2,0)
harmaccelLfd = vec2Lfd(Lcoef, dayrange)
```

Now that we are set up to do some smoothing, we will want to try a range of smoothing parameter λ values and examine the degrees of freedom and values of the generalized cross–validation coefficient GCV associated with each value of λ. First we set up a range of values (identified, of course, by some preliminary trial-

and-error experiments). We also set up two vectors to contain the degrees of freedom and GCV values.

```
loglam  = seq(4,9,0.25)
nlam    = length(loglam)
dfsave  = rep(NA,nlam)
gcvsave = rep(NA,nlam)
```

Here are commands that loop through the smoothing values, storing degrees of freedom and GCV along the way:

```
for (ilam in 1:nlam) {
  cat(paste('log10 lambda =',loglam[ilam],'\n'))
  lambda   = 10^loglam[ilam]
  fdParobj = fdPar(daybasis, harmaccelLfd, lambda)
  smoothlist = smooth.basis(day.5, logprecav,
                              fdParobj)
  dfsave[ilam]  = smoothlist$df
  gcvsave[ilam] = sum(smoothlist$gcv)
}
```

The GCV values have to be summed, since function `smooth.basis` returns a vector of GCV values, one for each replicate.

Figure 5.2 plots the GCV values. This shows a minimum at $\log_{10}(\lambda) = 6$. Next we smooth at this level and add labels to the resulting functional data object. Then we plot all the log precipitation curves in a single plot, followed by a curve–by–curve plot of the raw data and the fitted curve.

```
lambda   = 1e6
fdParobj = fdPar(daybasis, harmaccelLfd, lambda)
logprec.fit = smooth.basis(day.5,logprecav,fdParobj)
logprec.fd = logprec.fit$fd
fdnames = list("Day (July 1 to June 30)",
           "Weather Station" = CanadianWeather$place,
           "Log 10 Precipitation (mm)")
logprec.fd$fdnames = fdnames
plot(logprec.fd)
plotfit.fd(logprecav, day.5, logprec.fd)
```

This example will be revisited in Chapter 7. There, we will see that the λ = 1e6 leaves some interesting structure in the residuals for a few weather stations. Moreover, the curvature in the `GCV` function is rather weak, suggesting we will not lose much by using other values of λ in the range of 1e5 to 1e8. Our advice at the end of Section 5.2.5 seems appropriate here, and perhaps we should have worked with a lower value of λ.

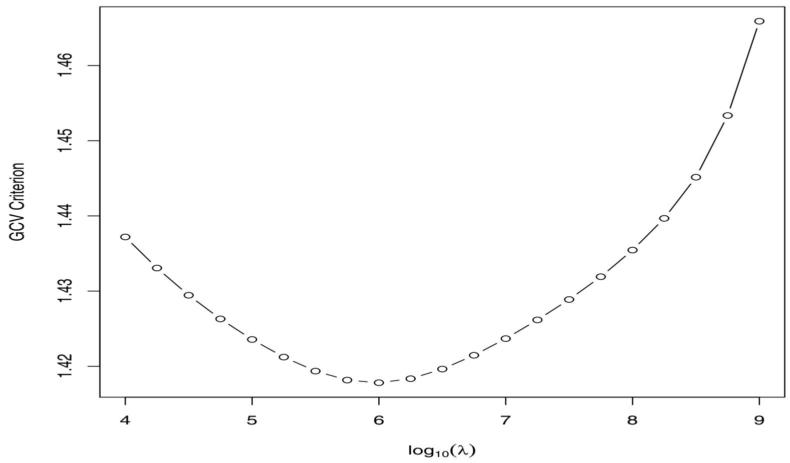

Fig. 5.2 The values of the generalized cross-validation or GCV criterion for the log precipitation data. The roughness penalty was defined by harmonic acceleration.

5.4 Positive, Monotone, Density and Other Constrained Functions

Often estimated curves must satisfy one or more side constraints. If the data are counts or other values that cannot be negative, then we do not want negative curve values, even over regions where values are at or close to zero. If we are estimating growth curves, it is probably the case that negative slopes are implausible, even if the noisy measurements do go down here and there. If the data are proportions, it would not make sense to have curve values outside the interval [0,1].

Unfortunately, linear combinations of basis functions such as those we have been using up to this point are difficult to constrain in these ways. The solution to the problem is simple: We transform the problem to one where the curve being estimated is unconstrained. We lose simple closed form expressions for the smoothing curve and therefore must resort to iterative methods for calculating the transformed curve, but the price is well worth paying.

5.4.1 Positive Smoothing

This transformation strategy is easiest to see in the case of positive (or negative) curves. We express the smoothing problem (5.3) as the transformed problem

$$y_j = \exp[w(t_j)] + \varepsilon_j = \exp[\phi(t_j)'\mathbf{c}] + \varepsilon. \tag{5.22}$$

That is, function $w(t)$ is now the logarithm of the data-fitting function $x(t) = \exp[w(t)]$, and consequently is unconstrained as to its sign, while at the same time the fitting function is guaranteed to be positive. It can go as close to zero as we like by permitting the values of $w(t)$ to be arbitrarily large negative numbers.

For example, we can smooth Vancouver's mean daily precipitation data, which can have zero but not negative values, using these commands using the function `smooth.pos` in R or `smooth_pos` in Matlab:

```
lambda    = 1e3
WfdParobj = fdPar(daybasis, harmaccelLfd, lambda)
VanPrec   = CanadianWeather$dailyAv[
  dayOfYearShifted, 'Vancouver', 'Precipitation.mm']
VanPrecPos = smooth.pos(day.5, VanPrec, WfdParobj)
Wfd = VanPrecPos$Wfdobj
```

These commands plot `Wfd`, the estimated log precipitation.

```
Wfd$fdnames = list("Day (July 1 to June 30)",
     "Weather Station" = CanadianWeather$place,
                  "Log 10 Precipitation (mm)")
plot(Wfd)
```

The fit to the data, shown in Figure 5.3, is displayed by

```
precfit = exp(eval.fd(day.5, Wfd))
plot(day.5, VanPrec, type="p", cex=1.2,
     xlab="Day (July 1 to June 30)",
     ylab="Millimeters",
     main="Vancouver's Precipitation")
lines(day.5, precfit,lwd=2)
```

5.4.2 Monotone Smoothing

Some applications require a fitting function $x(t)$ that is either monotonically increasing or decreasing, even though the observations may not exhibit perfect monotonicity:

$$y_j = \beta_0 + \beta_1 x(t_j) + \varepsilon_j \tag{5.23}$$

We can get this easily by letting

$$x(t) = \int_{t_0}^{t} \exp[w(u)]\,du. \tag{5.24}$$

Here t_0 is the fixed origin for the range of t-values for which the data are being fit. The intercept term β_0 in (5.23) is the value of the approximating function at t_0.

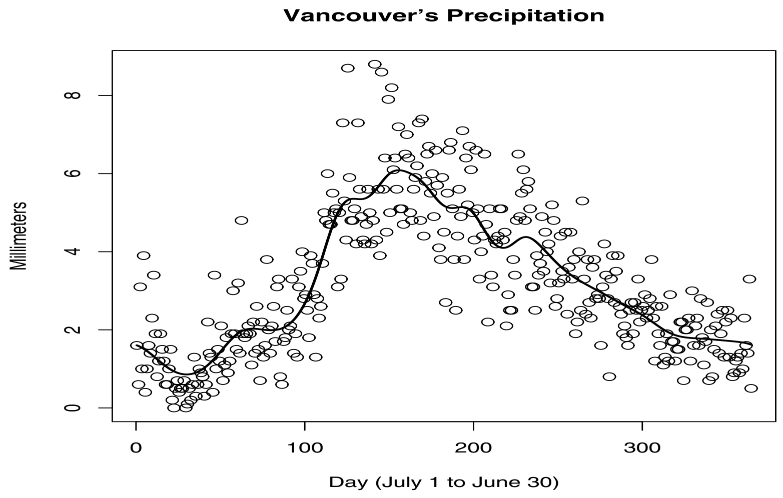

Fig. 5.3 Vancouver's precipitation data, along with a fit estimated by positive smoothing.

For monotonically increasing functions, β_1 could be absorbed into $w(u)$. However, to allow for monotonically decreasing functions, we keep β_1 separate and select normalize $w(u)$ for numerical stability.

Substituting (5.24) into (5.23) produces the following:

$$y_j = \beta_0 + \beta_1 \int_{t_0}^{t_j} \exp[w(u)]\, du + \varepsilon_j = \beta_0 + \beta_1 \int_{t_0}^{t_j} \exp[\phi(u)'\mathbf{c}]\, du + \varepsilon_j. \quad (5.25)$$

The function `smooth.monotone` estimates β_0, β_1, and $w(u)$.

5.4.2.1 Smoothing the Length of a Newborn Baby's Tibia

Figure 5.4 shows the length of the tibia of a newborn infant, measured by Dr. Michael Hermanussen with an error of the order of 0.1 millimeters, over its first 40 days. The staircase nature of growth in this early period and need to estimate the velocity of change in bone length, also shown in the figure, makes monotone smoothing essential. It seems astonishing that this small bone in the baby's lower leg has the capacity to grow as much as two millimeters in a single day.

Variables `day` and `tib` in the following code contain the numbers of the days and the measurements, respectively. A basis for function w and a smoothing profile are set up, the data are smoothed, the values of the functional data object for w and the coefficients β_0 and β_1 are returned. Then the values of the smoothing and velocity curves are computed.

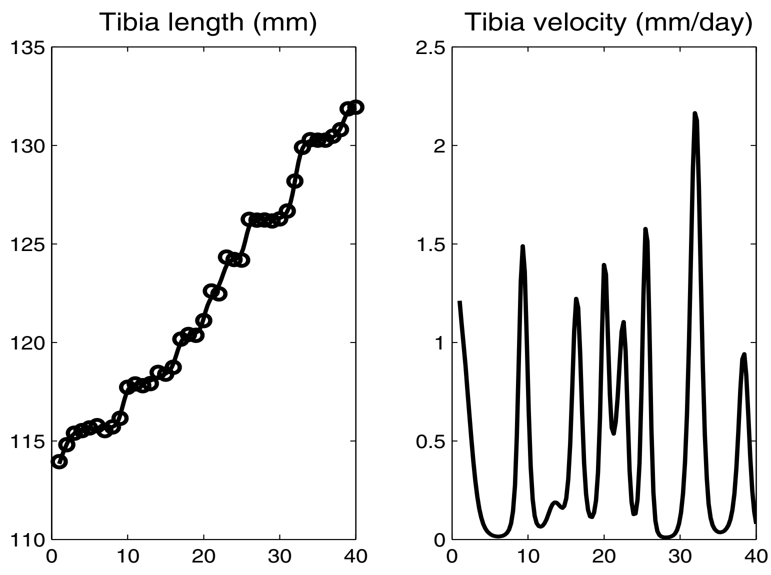

Fig. 5.4 The left panel shows measurements of the length of the tibia of a newborn infant over its first 40 days, along with a monotone smooth of these day. The right panel shows the velocity or first derivative of the smoothing function.

```
Wbasis = create.bspline.basis(c(1,n), nbasis)
Wfd0   = fd(matrix(0,nbasis,1), Wbasis)
WfdPar = fdPar(Wfd0, 2, 1e-4)
result = smooth.monotone(day, tib, WfdPar)
Wfd    = result$Wfd
beta   = result$beta
dayfine  = seq(1,n,len=151)
tibhat   = beta[1] + beta[2]*eval.monfd(dayfine ,Wfd)
Dtibhat  = beta[2]*eval.monfd(dayfine, Wfd, 1)
D2tibhat = beta[2]*eval.monfd(dayfine, Wfd, 2)
```

5.4.2.2 Smoothing the Berkeley Female Growth Data

In Chapter 8 we will need our best estimates of the growth acceleration functions for the Berkeley girls, and smoothing their data monotonically substantially improves these estimates over direct smoothing, and especially in the neighborhood of the pubertal growth spurts.

We set up an order 6 spline basis with knots at ages of observations for their functions w, along with a roughness penalty on their third derivatives and a smoothing parameter of $1/\sqrt{10}$, in these commands:

```
wbasis = create.bspline.basis(c(1,18), 35, 6, age)
growfdPar = fdPar(wbasis, 3, 10^(-0.5))
```

The monotone smoothing of the data in the 31 by 54 matrix `hgtf`, and the extraction of the the functional data object `Wfd` for the w_i functions, the coefficients β_{0i}, β_{1i} and the functional data object `hgtfhatfd` for the functions fitting the data are achieved by

```
growthMon = smooth.monotone(age, hgtf, growfdPar)
Wfd       = growthMon$Wfd
betaf     = growthMon$beta
hgtfhatfd = growthMon$yhatfd
```

5.4.3 Probability Density Functions

A probability density function $p(z)$ is used to indicate the probability of observing a scalar observation at or near a value z, and is one of the core functions in statistics. From our perspective, p is a special case of a positive function in the sense of having a unit integral over the range of z. That is, we can express p as

$$p(z) = C\exp[w(z)], \tag{5.26}$$

where the positive *normalizing constant* C satisfies the constraint $\int p(z)\mathrm{d}z = 1$. Estimating a free-form nonparametric version of p is not a smoothing problem as we have so far defined it, since we would not use an error sum of squares measure of lack of fit. Rather, the usual practice would be to minimize a *penalized negative log likelihood*,

$$-\ln L(\mathbf{c}) = -\sum_i^N \ln p(z_i) + \lambda \int [Lw(z)]^2 \mathrm{d}z = -\sum_i^N w(z_i) - N\ln C + \lambda \int [Lw(z)]^2 \mathrm{d}z, \tag{5.27}$$

where $w(z) = \mathbf{c}'\phi(z)$. Notice that the first two terms replace the error sum of squares in (5.13).

The linear differential operator L can be chosen so as to force p to approach specific parametric density functions as $\lambda \to \infty$. For example, $L = D^3$ will do this for the Gaussian density (Silverman, 1986).

Function `density.fd` is used to estimate a nonparametric probability density function from a sample of data. We will illustrate its use for a rather challenging problem: describing the variation in daily precipitation for the Canadian prairie city of Regina over the month of June and over the 34 years from 1960 through 1993. June is the critical month for wheat growers because the crop enters its most rapid growing phase, and an adequate supply of moisture in the soil is essential for a good crop.

Precipitation is a difficult quantity to model for several reasons. First of all, on about 65% of the days in this region, no rain is even possible, so that zero really means a "nonprecipitation day" rather than "no rain." Since there can be a small amount of precipitation from dew, we used only days when the measured precipitation exceeded two millimetres. Also, precipitation can come down in two main ways: as a gentle drizzle and, more often, as a sudden and sometimes violent thunderstorm. Consequently, the distribution of precipitation is extremely skewed, and Regina experienced three days in this period with more than 40 mm of rain. We deleted these days, too, in order to improve the graphical displays, leaving $N = 212$ rainfall values.

Figure 5.5 plots the *ordered* rainfalls for the 1,006 days when precipitation was recorded against their rank orders, a version of a *quantile plot*. We can see just how extreme precipitation can be; the highest rainfall of 132.6 mm on June 25, 1975, is said to have flooded 20,000 basements.

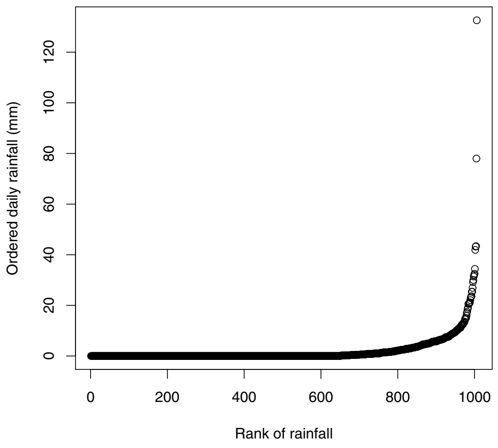

Fig. 5.5 The empirical quantile function for daily rainfall at Regina in the month of June over 34 years.

We set up the break points for a cubic B-spline basis to be the rainfalls at 11 equally spaced ranks, beginning at the first and ending at N. In this code variable `RegPrec` contains the 212 sorted rainfall amounts between 2 and 45 mm.

```
Wknots  = RegPrec[round(N*seq(1/N,1,len=11),0)]
Wnbasis = length(Wknots) + 2
Wbasis  = create.bspline.basis(range(RegPrec),13,4,
                                  Wknots)
```

Now we estimate the density, applying a light amount of smoothing, and extract the functional data object `Wfd` and the normalizing constant `C` from the list that `density.fd` returns.

```
  Wlambda     = 1e-1
  WfdPar      = fdPar(Wbasis, 2, Wlambda)
  densityList = density.fd(RegPrec, WfdPar)
  Wfd         = densityList$Wfdobj
  C           = densityList$C
```

These commands set up the density function values over a fine mesh of values.

```
  Zfine = seq(RegPrec[1],RegPrec[N],len=201)
  Wfine = eval.fd(Zfine, Wfd)
  Pfine = exp(Wfine)/C
```

The estimated density is shown in Figure 5.6. The multiphase nature of precipitation is clear here. The first phase is due to heavy dew or a few drops of rain, followed by a peak related to light rain from low-pressure ridges that arrive in this area from time to time, and then thunderstorm rain that can vary from about 7 mm to catastrophic levels.

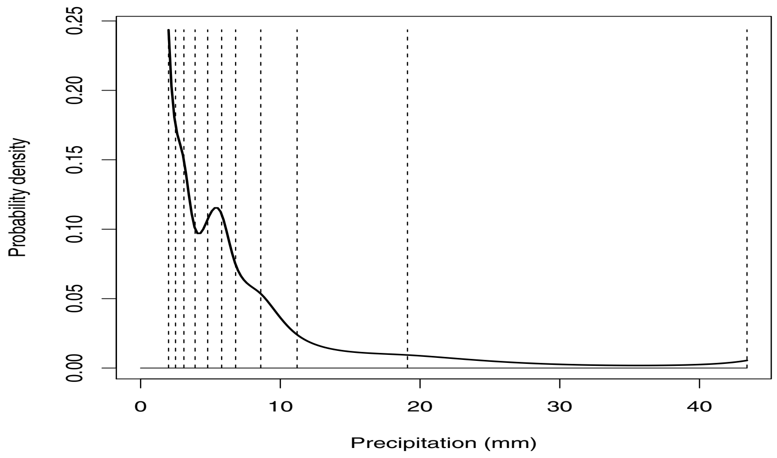

Fig. 5.6 The solid line indicates the probability density function $p(z)$ for rainfall in Regina of 2 mm or greater, but stopping at about 45 mm. The vertical dashed lines indicate the knot values used to define the cubic B-spline expansion for $w = \ln p$.

5.5 Assessing the Fit to the Data

Having smoothed the data, there are many questions to ask, and these direct us to do some further analyses of the residuals, $r_{ij} = y_{ij} - x_i(t_j)$. These analyses can be functional since there is some reason to suppose that at least part of the variation in these residuals across t is smooth.

Did we miss some important features in the data by oversmoothing? Perhaps, for example, there may have been something unusual in one or two curves that we missed because the GCV criterion selected a level of smoothing that worked best for all samples simultaneously. Put another way, could there be an indication that we might have done better to smooth each weather station's log precipitation data separately? We will defer looking at this question until the end of the next chapter, since principal components analysis can be helpful here.

A closely related question concerns whether the variation in the residuals conforms to the assumptions implicit in the type of smoothing that we performed. The use of the unweighted least-squares criterion is only optimal if the residuals for all time points are normally distributed and if the variance of these residuals is constant across both years and weather stations (curves).

We now return to the log precipitation data considered in Section 5.3 and create a 365 by 35 matrix of residuals from the fit discussed there. We then use this to create variance vectors across

- stations, of length 365, dividing by 35 since the residuals need not sum to zero on any day,
- time, of length 35, dividing by 365-12; the number "12" here is essentially the equivalent degrees of freedom in the fit (logprec.fit$df).

```
 logprecmat = eval.fd(day.5, logprec.fd)
 logprecres = logprecav - logprecmat
 #  across stations
 logprecvar1 = apply(logprecres^2, 1, sum)/35
 #  across time
 logprecvar2 = apply(logprecres^2, 2, sum)/(365-12)
```

Let us look at how residual variation changes over stations; Figure 5.7 displays their standard deviations. With labels on a few well-known stations and recalling that we number the stations from east to west to north, we see that there tends to be more variation for prairie and northerly stations in the center of the country, and less for marine stations. This is interesting but perhaps not dramatic enough to make us want to pursue the matter further.

Figure 5.8 shows how standard deviations taken over stations and within days vary. The smooth line in the plot was computed by smoothing the log of the standard deviations and exponentiating the result by these two commands:

```
logstddev.fd = smooth.basis(day.5,
                        log(logprecvar1)/2, fdParobj)$fd
logprecvar1fit = exp(eval.fd(day.5, logstddev.fd))
```

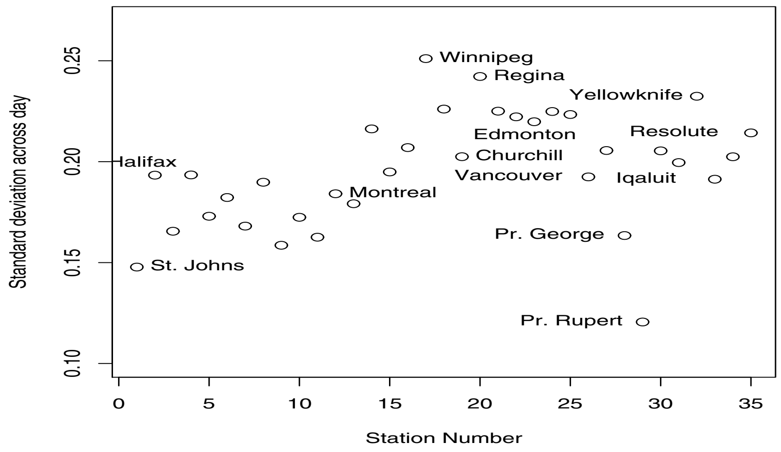

Fig. 5.7 Standard deviations of the residuals from the smooth of the log precipitation taken across days and within stations.

We could also have used `smooth.pos` to do the job. We see now that there is a seasonal variation in the size of the residuals, with more variation in summer months than in winter. Nevertheless, this form of variation is not strong enough to justify returning to do a weighted least-squares analysis using `smooth.basis`; we would need much larger variations in the variability for it to create a substantive difference between weighted and unweighted solutions.

Also implicit in our smoothing technology is the assumption that residuals are uncorrelated. This is a rather unlikely situation; departures from smooth variation tend also to be smooth, implying a strong positive autocorrelation between neighboring residuals. If observation times are equally spaced, we can use standard time series techniques to explore this autocorrelation structure.

5.6 Details for the `fdPar` Class and `smooth.basis` Function

5.6.1 The `fdPar` class

We give here the arguments of the constructor function `fdPar` that constructs an object of the functional parameter `fdPar` class. The complete calling sequence is

```
fdPar(fdobj=NULL, Lfdobj=NULL, lambda=0,
      estimate=TRUE, penmat=NULL)
```

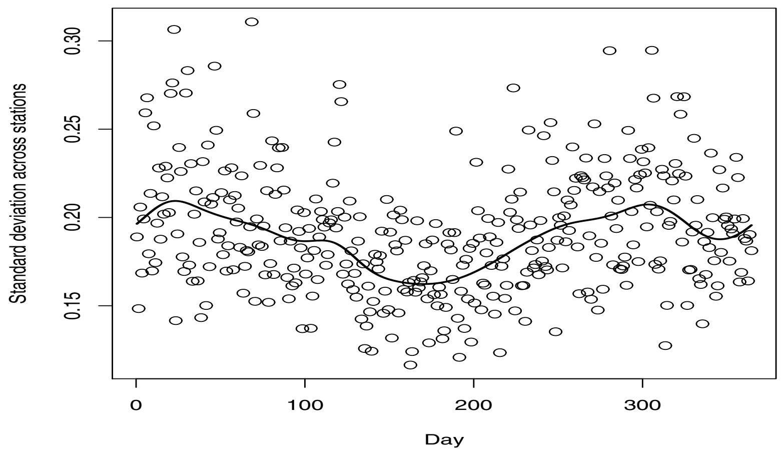

Fig. 5.8 Standard deviations of the residuals from the smooth of the log precipitation taken across stations and within days. The solid line is an exponentiated smooth of the log of the variances.

The arguments are as follows:

`fdobj` A functional data object, functional basis object, a functional parameter object or a matrix. If it a matrix, it is replaced by `fd(fdobj)`. If `class(fdobj) == 'basisfd'`, it is converted to an object of class `fd` with a coefficient matrix consisting of a single column of zeros.

`Lfdobj` Either a nonnegative integer or a linear differential operator object. If `NULL`, `Lfdobj` depends on `fdobj[['basis']][['type']]`:

- `bspline` `Lfdobj = int2Lfd(max(0, norder-2))`, where `norder = norder(fdobj)`.
- `fourier` `Lfdobj` is a harmonic acceleration operator set up for the period used to define the basis.
- anything else `Lfdobj <- int2Lfd(0)`

`lambda` A nonnegative real number specifying the amount of smoothing to be applied to the estimated functional parameter.

`estimate` Not currently used.

`penmat` A roughness penalty matrix. Including this can eliminate the need to compute this matrix over and over again in some types of calculations.

5.6.2 The `smooth.basis` Function

The calling sequence for `smooth.basis` is

```
smooth.basis(argvals, y, fdParobj,
             wtvec=rep(1, length(argvals)),
             fdnames=NULL)
```

The arguments are as follows:

`argvals` A vector of argument values correspond to the observations in array `y`.
`y` An array containing values of curves at a finite number of sampling points or argument values. If the array is a matrix, the rows must correspond to argument values and columns to replications, and it will be assumed that there is only one variable per observation. If `y` is a three-dimensional array, the first dimension corresponds to argument values, the second to replications, and the third to variables within replications. If `y` is a vector, only one replicate and variable are assumed.
`fdParobj` A functional parameter object, a functional data object or a functional basis object. If the object is a functional parameter object, then the linear differential operator object and the smoothing parameter in this object define the roughness penalty. If the object is a functional data object, the basis within this object is used without a roughness penalty, and this is also the case if the object is a functional basis object.
`wtvec` A vector of the same length as `argvals` containing weights for the values to be smoothed.
`fdnames` A list of length three containing character vectors of names for the following:

`args` Names for each observation or point in time at which data are collected.
`reps` Names for each `rep`, unit or subject.
`fun` Names for each function or (response) variable measured repeatedly (per `args`) for each `rep`.

Function `smooth.basis` returns an argument of the `fdSmooth` class, which is a named list of length eight with the following components:

`fd` A functional data object containing a smooth of the data.
`df` A degrees of freedom measure of the smooth.
`gcv` The value of the generalized cross-validation or GCV criterion. If there are multiple curves, this is a vector of values, one per curve. If the smooth is multivariate, the result is a matrix of GCV values, with columns corresponding to variables.
`SSE` The error sums of squares; `SSE` is a vector or a matrix of the same size as `gcv`.
`penmat` The penalty matrix.
`y2cMap` The matrix mapping the data to the coefficients.
`argvals, y` Input arguments

5.7 Some Things to Try

1. In order to understand the implications of choice of smoothing parameter λ, it is best to work with simulated data.

   a. Choose a function with some interesting variation, such as
      - $\sin(4\pi t)$ over [0,1]
      - $\exp(-t^2/2)$ over [-5,5]
   b. Specifying some sampling points $t_j, j = 1,\ldots,n$, evaluate your function at these points, and add some mean 0 normal random error to these values, where you specify the standard deviation.
   c. Specify a basis system, an order of derivative to penalize, and a value smoothing parameter λ, and bundle these together in an `fdPar` object.
   d. Smooth the points using function `smooth.basis` (R) or `smooth_basis` (Matlab).
   e. Compute the square root of the mean square error, and compare this to the standard deviation that you used to generate the error. You might want to subtract the equivalent degrees of freedom associated with the λ value that you used from n before dividing the sum of squared errors.
   f. Experiment with different values of λ, and find one that gives nearly the right value for root mean square error.
   g. Experiment with different values of λ, and save the GCV values for each. Plot these GCV values against $\log_{10}(\lambda)$, and estimate by eye the value of λ minimizing GCV.

2. Now try your hand at smoothing some real data, such as what you used in the exercises in Chapter 1.
3. Calculate the derivative of the function you used to generate the data in 1a. How does the derivative of your smooth compare with this? Does some value for λ other than that given by GCV improve the agreement between estimated and true derivatives?
4. **Melanoma Data** The `melanoma` data set in the `fda` package for R contains age-adjusted incidences of melanoma from the Connecticut Tumor Registry for the years 1935 to 1972.

   a. Fit these data with a Fourier basis, choosing the number of basis functions by minimizing the `gcv` value returned by `smooth.basis`.
   b. Try removing a linear trend for these data first, either directly or by looking at the residuals after a call to `lm`. Repeat the steps above; does the optimal number of basis functions change?
   c. Refit the data using a B-spline basis and a harmonic acceleration penalty. Try some values of λ to optimize `gcv`. You will need to guess at the period to use; how does doubling and halving the period change the degrees of freedom at the optimal value of λ?
   d. Set up the linear differential operator $\omega^2 D^2 + D^4$, which annihilates sinusoidal combined with linear trend. Smooth the melanoma data using this operator.

e. Plot the velocity versus acceleration curves for the fit using a Fourier basis and using the B-spline basis with a harmonic acceleration penalty. Are they substantially different? Do they provide evidence of subcycles?

5.8 More to Read

There is a large literature on smoothing methods, and Ramsay and Silverman (2005) devote a number of chapters to the problem. Recent book-length references are Eubank (1999), Rupert et al. (2003), and Simonoff (1996). Moreover, there are smoothing methods that do not define x explicitly in terms of basis functions that may serve as well, such as *local polynomial smoothing*. However, the well-known method of *kernel smoothing*, made all too available in software packages, should now be viewed as obsolete because its poor performance near the end points of the interval (Fan and Gijbels, 1996).