Chapter 14
Functional Regression

14.1 Principles of Functional Linear Regression Analyses

The general functional linear regression model with scalar response (Y) and one functional covariate ($x(\cdot)$) is defined by

$$Y = \mu + \int_0^T x(t)\beta(t)dt + \epsilon, \tag{14.1}$$

where $x(t)$ and $\beta(t)$ are a centered functional covariate and the functional coefficient regression, respectively, and ϵ is a random error often assumed to have a normal distribution with mean 0 and variance σ^2. Functional regression replaces the linear predictor of a linear regression model by integrating the product of a coefficient function $\beta(t)$ and centered covariate $x(t)$, which corresponds to a continuous non-delaying process.

Determining the infinite-dimensional beta coefficients $\beta(t)$ from a finite number of observations of the model (1) is a very difficult task. Indeed, it is almost always possible to find a function $\beta(t)$ satisfying the model with an error equal to 0, and there is an infinite number of these functions that give the same predictions (Ramsay et al. 2009). There are several procedures to solve this problem (Cardot and Sarda 2006); one of them is based on basis expansion (Fourier, B-splines, etc.) and will be adopted and described here.

A basis expansion solution is obtained by first representing the beta coefficient function $\beta(t)$ as

$$\beta(t) = \sum\nolimits_{l=1}^{L_1} \beta_l \phi_l(t), \tag{14.2}$$

where $\phi_l(\cdot)$, $l = 1, \ldots, L_1$, is a collection of functions corresponding to the first L_1 elements of basis for a function space and β_l are constants that depend on the

O. A. Montesinos López et al., *Multivariate Statistical Machine Learning Methods for Genomic Prediction*, https://doi.org/10.1007/978-3-030-89010-0_14

function to be represented (Ramsay et al. 2009). Then, by assuming this form for $\beta(t)$, model (14.1) can be expressed as

$$Y = \mu + \sum_{l=1}^{L_1} \beta_l \int_0^T x(t)\phi_l(t)dt + \epsilon = \mu + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\beta}_0 + \epsilon$$
$$= \boldsymbol{x}^{*\mathrm{T}}\boldsymbol{\beta} + \epsilon, \tag{14.3}$$

where $\boldsymbol{x}^* = [1, \boldsymbol{x}^{\mathrm{T}}]^{\mathrm{T}}$, $\boldsymbol{x} = [x_1, \ldots, x_{L_1}]^{\mathrm{T}}$, $x_l = \int_0^T x(t)\phi_l(t)dt$, $l = 1, \ldots, L_1$. So, if y_i, $i = 1, \ldots, n$, are independent observations of model (14.1), corresponding to covariate functions $x_i(\cdot)$, $i = 1, \ldots, n$, a basis expansion solution for the beta coefficient function is obtained by estimating the parameters involved in model (14.3), and then substituting $\widehat{\boldsymbol{\beta}} = \left[\mu, \widehat{\beta}_1, \ldots, \widehat{\beta}_{L_1}\right]^{\mathrm{T}}$ in (14.2) to obtain a basis-based estimation of $\beta(t)$:

$$\widehat{\beta}(t) = \sum_{l=1}^{L_1} \widehat{\beta}_l \phi_l(t).$$

When smoothing in the function coefficient is desired, one way to take more control of this is by using a roughness penalty, which combined with a high-dimensional basis could reduce the possibility of not considering some important features or taking into account some extraneous features (Ramsay et al. 2009). However, sometimes we can obtain good results without recurring to this, if the number of basis functions is smaller than the number of individuals in the sample (Ramsay et al. 2009).

Assuming that random errors are independently and identically distributed as a normal random variable with mean 0 and variance σ^2, $\epsilon_1, \ldots, \epsilon_n \sim iid\, N(0, \sigma^2)$, then $Y_i = \mu + \sum_{l=1}^{L_1} x_{il}\beta_l + \epsilon_i \sim N\left(\mu + \sum_{l=1}^{L_1} x_{il}\beta_l, \sigma^2\right)$, $x_{il} = \int_0^T x_i(t)\phi_l(t)dt$, $i = 1, \ldots, n$, $l = 1, \ldots, L_1$. So, the maximum likelihood estimation of parameters $\boldsymbol{\beta}$ and σ^2 is given by

$$\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{*\mathrm{T}}\boldsymbol{X}^*\right)^{-1}\boldsymbol{X}^{*\mathrm{T}}\boldsymbol{y} \tag{14.4}$$

$$\widehat{\sigma}^2 = \frac{1}{n}\left(\boldsymbol{y} - \boldsymbol{X}^*\widehat{\boldsymbol{\beta}}\right)^{\mathrm{T}}\left(\boldsymbol{y} - \boldsymbol{X}^*\widehat{\boldsymbol{\beta}}\right), \tag{14.5}$$

where $\boldsymbol{X}^* = [\mathbf{1}_n \ \boldsymbol{X}]$, $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^{\mathrm{T}}$ is assumed to be of full column rank ($n > L_1$), $\mathbf{1}_n$ is a vector of dimension $n \times 1$ with all its entries equal to 1 and $\boldsymbol{x}_i = [x_{i1}, \ldots, x_{iL_1}]^{\mathrm{T}}$, $i = 1, \ldots, n$.

However, in practice, the functional covariate is often unknown and not continuously observed. Usually, it is only measured in a finite number of points $t_1 < t_2 < \ldots < t_m$ in time or another domain. So, to complete the solution described before, the usual approach is also to assume that the covariate function can be represented as a linear combination of a set of basis functions ($\psi_l(\cdot)$, $l = 1, \ldots, L_2$)

$$x_i(t) = \sum\nolimits_{o=1}^{L_2} c_{io}\psi_o(t), \tag{14.6}$$

where c_{io}, $o = 1, \ldots, L_2$, are constants to be determined for each observation, $i = 1, \ldots, n$. Usually, this is determined by least squares, in which case, by assuming that all curves were observed at the same time points, this can be computed as

$$\widehat{\boldsymbol{c}}_i = [\widehat{c}_{i1}, \ldots, \widehat{c}_{iL_2}]^{\mathrm{T}} = \left(\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{x}_i(\boldsymbol{t}), \tag{14.7}$$

where $\boldsymbol{\Psi}$ is a matrix of dimension $m \times L_2$ given by

$$\boldsymbol{\Psi} = \begin{bmatrix} \psi_1(t_1) & \cdots & \psi_{L_2}(t_1) \\ \psi_1(t_2) & \ddots & \psi_{L_2}(t_2) \\ \vdots & \vdots & \vdots \\ \psi_1(t_m) & \cdots & \psi_{L_2}(t_m) \end{bmatrix} \tag{14.8}$$

and $\boldsymbol{x_i(t)} = [x_i(t_1), \ldots, x_i(t_m)]^{\mathrm{T}}$ is the vector with the actual values where the covariate curve of individual i was observed. With this, the elements of $\boldsymbol{x}_i = [x_{i1}, \ldots, x_{iL_1}]^{\mathrm{T}}$, $x_{il} = \int_0^T x_i(t)\phi_l(t)dt$, can be re-expressed as $x_{il} = \int_0^T x_i(t)\phi_l(t)dt = \sum_{o=1}^{L_2}\widehat{c}_{io}\int_0^T \psi_o(t)\phi_l(t)dt = \boldsymbol{x}_l^{**\mathrm{T}}\widehat{\boldsymbol{c}}_i$, with $\boldsymbol{x}_l^{**} = \left[x_{l1}^*, \ldots, x_{lL_2}^*\right]^{\mathrm{T}}$ and $x_{lo}^* = \int_0^T \phi_l(t)\psi_o(t)dt$, $o = 1, \ldots, L_2$ and $l = 1, \ldots, L_1$. From here, $\boldsymbol{x}_i$ can be expressed as

$$\boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{x}_1^{**\mathrm{T}}\widehat{\boldsymbol{c}}_i \\ \vdots \\ \boldsymbol{x}_{L_1}^{**\mathrm{T}}\widehat{\boldsymbol{c}}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1^{**\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{L_1}^{**\mathrm{T}} \end{bmatrix}\widehat{\boldsymbol{c}}_i = \begin{bmatrix} \int_0^T \phi_1(t)\psi_1(t)dt & \cdots & \int_0^T \phi_1(t)\psi_{L_2}(t)dt \\ \vdots & \ddots & \vdots \\ \int_0^T \phi_{L_1}(t)\psi_m(t)dt & \cdots & \int_0^T \phi_{L_1}(t)\psi_{L_2}(t)dt \end{bmatrix}\widehat{\boldsymbol{c}}_i = \boldsymbol{Q}\widehat{\boldsymbol{c}}_i,$$

where

$$\boldsymbol{Q} = \begin{bmatrix} \int_0^T \phi_1(t)\psi_1(t)dt & \cdots & \int_0^T \phi_1(t)\psi_{L_2}(t)dt \\ \vdots & \ddots & \vdots \\ \int_0^T \phi_{L_1}(t)\psi_m(t)dt & \cdots & \int_0^T \phi_{L_1}(t)\psi_{L_2}(t)dt \end{bmatrix}.$$

Now, matrix $\boldsymbol{X}^*$ can be computed as

$$\boldsymbol{X}^* = [\,\mathbf{1}_n \quad \boldsymbol{X}\,], \tag{14.9}$$

where

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_n^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \widehat{\boldsymbol{c}}_1^{\mathrm{T}} \boldsymbol{Q}^{\mathrm{T}} \\ \vdots \\ \widehat{\boldsymbol{c}}_n^{\mathrm{T}} \boldsymbol{Q}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} \widehat{\boldsymbol{c}}_1^{\mathrm{T}} \\ \vdots \\ \widehat{\boldsymbol{c}}_n^{\mathrm{T}} \end{bmatrix} \boldsymbol{Q}^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{x}_1(\boldsymbol{t})^{\mathrm{T}} \boldsymbol{\Psi} \left(\boldsymbol{\Psi}^{\mathrm{T}} \boldsymbol{\Psi} \right)^{-1} \\ \vdots \\ \boldsymbol{x}_n(\boldsymbol{t})^{\mathrm{T}} \boldsymbol{\Psi} \left(\boldsymbol{\Psi}^{\mathrm{T}} \boldsymbol{\Psi} \right)^{-1} \end{bmatrix} \boldsymbol{Q}^{\mathrm{T}}$$
$$= \boldsymbol{X}^{**} \boldsymbol{\Psi} \left(\boldsymbol{\Psi}^{\mathrm{T}} \boldsymbol{\Psi} \right)^{-1} \boldsymbol{Q}^{\mathrm{T}}$$

with $\boldsymbol{X}^{**} = [\boldsymbol{x}_1(\boldsymbol{t}) \quad \cdots \quad \boldsymbol{x}_n(\boldsymbol{t})]^{\mathrm{T}}$. Finally, the complete practical solution of the parameter estimates is obtained with (14.4) and (14.5) but replacing $\boldsymbol{X}^*$ as computed in (14.9).

There are several proposals to choose the "optimal" number of bases (L_1) to represent the $\beta(\cdot)$ function coefficient. One way is by means of the Bayesian information criterion (Górecki et al. 2018), which is defined as follows:

$$\text{BIC} = -2\ell\left(\widehat{\boldsymbol{\beta}}, \widehat{\sigma}^2; \boldsymbol{y}\right) + (L_1 + 1)\log(n),$$

where $\ell\left(\widehat{\boldsymbol{\beta}}, \widehat{\sigma}^2; \boldsymbol{y}\right)$ is the log-likelihood evaluated in the maximum likelihood estimation of parameters $\boldsymbol{\beta}$ and σ^2. This is a compromise between the fit of the model (first term) and its complexity (second term, the number of parameters in the model). In general, the model with the lowest BIC is preferred. In particular, with this criterion, the "optimal" number of basis functions corresponds to the lowest BIC.

When smoothing is required in the curve to be estimated, one way to control it is through the introduction of a penalty term, as will be described later. However, sometimes good results can be obtained without the need for this, as long as the number of basis functions relative to the amount of data is kept small (Ramsay et al. 2009).

To choose the "optimal" number of basis functions (L_2) to represent the functional covariate data, we can also use the BIC criteria. To do this, consider that each curve is observed with error under the following model:

$$x_i(t_j) = \sum\nolimits_{o=1}^{L_2} c_{io}\psi_o(t_j) + \epsilon_{ij},$$

where for each $i = 1, \ldots, n$, $\epsilon_{ij}, j = 1, \ldots, m$, are independent random variables with distribution $N\left(0, \sigma_x^2\right)$. Then the likelihood of the parameters to be estimated ($\boldsymbol{c}_i = (c_{i1}, \ldots, c_{iL_2})^{\mathrm{T}}$ and σ_x^2) is given by

$$L\left(\boldsymbol{c}_i, \sigma_x^2; \boldsymbol{x}_i(\boldsymbol{t})\right) = \prod_{j=1}^{m} f_{nx_i(t_j)}\left(x_i(t_j)\right)$$
$$= \left(\frac{1}{2\pi\sigma_x^2}\right)^{m/2} \exp\left[-\frac{1}{2\sigma_x^2}\sum_{j=1}^{m}\left(x_i(t_j) - \sum_{o=1}^{L_2} c_{io}\psi_o(t_j)\right)^2\right].$$

From this, the maximum likelihood of the parameters of $\boldsymbol{c}_i$ and σ_x^2 are $\widehat{\boldsymbol{c}}_i = [\widehat{c}_{i1}, \ldots, \widehat{c}_{iL_2}]^{\mathrm{T}} = \left(\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{\Psi}\right)^{-1}\boldsymbol{\Psi}^{\mathrm{T}}\boldsymbol{x}_i(\boldsymbol{t})$ and $\widehat{\sigma}_x^2 = \frac{1}{m}\sum_{j=1}^{m}\left(x_i(t_j) - \sum_{o=1}^{L_2}\widehat{c}_{io}\psi_o(t_j)\right)^2$, respectively. So, before fitting the regression model, the "optimal" value of L_2 that will represent each curve in the sample can be chosen with the smallest value of BIC in the corresponding model:

$$\mathrm{BIC} = -2\log\left[L\left(\widehat{\boldsymbol{c}}_i, \widehat{\sigma}_x^2; \boldsymbol{x}_i(\boldsymbol{t})\right)\right] + (L_2 + 1)\log(m).$$

A global value of L_2 can be adopted as suggested by Górecki et al. (2018), and the mode of the "optimal" values obtained across all the represented curves. Note that the maximum likelihood estimate of $\boldsymbol{c}_i$ is the same as the least square estimate mentioned above.

Another alternative to the BIC approach is to choose the "optimal" number of basis functions by estimating the predictive ability obtained by using different values of L_2 and selecting the value with the best predictive performance (Ruppert et al. 2003). One way to do this is by using the leave-one-out cross-validation (LOOCV) with mean squared error of prediction as the criterion to measure the predictive performance:

$$\mathrm{CV}_1(L_2) = \sum_{j=1}^{m}\left(x(t_j) - \widehat{x}_{-j}(t_j)\right)^2,$$

where $\widehat{x}_{-j}(t_j)$ is the predicted value of point j, $x(t_j)$, obtained by doing the representation of the function with L_2 bases but without this point, that is,

$$\widehat{x}_{-j}(t_j) = \sum_{o=1}^{L_2}\widehat{c}_{-j,o}\psi_o(t_j),$$

where $\widehat{\boldsymbol{c}}_{-j} = \left[\widehat{c}_1^{-j}, \ldots, \widehat{c}_{L_2}^{-j}\right]^{\mathrm{T}} = \left(\boldsymbol{\Psi}_{-j}^{\mathrm{T}}\boldsymbol{\Psi}_{-j}\right)^{-1}\boldsymbol{\Psi}_{-j}^{\mathrm{T}}\boldsymbol{x}_{-j}$ and $\boldsymbol{\Psi}_{-j}$ is a matrix of dimension $(m-1) \times L_2$, like the matrix design basis defined in (14.8) over L_2 basis functions, but without row j, and $\boldsymbol{x}_{-j}$ is the same as the vector that contains the observed values of the latent function, $\boldsymbol{x}(\boldsymbol{t})$, but removing the value of its position j. For a specific basis, the optimal number of basis is the one with the lowest value of $\mathrm{CV}_1(L_2)$.

14.2 Basis Functions

A "base" is a set of basis functions (ϕ_l, $l = 1, 2, 3, \ldots$) such that "any" function ($x(t)$) can be approximated as well as required, by means of a linear combination of L_2 of these functions:

$$x(t) = \sum_{l=1}^{L_2} c_l \phi_l(t),$$

where c_l are values that will determine the function.

In general, to represent data in functions by means of basis functions, you need to

(a) Choose suitable basis functions (polynomial basis, Fourier basis, B-spline basis, etc.).
(b) Determine the number of basis functions to consider (L_2).
(c) Estimate the coefficients c_l, $l = 1, \ldots, L_2$.

The degree of smoothness of function $x(t)$ depends on the value of L_2 that is chosen (a small value of L_2 causes more smoothing of the curves) and the optimum value for L_2 selected using the Bayesian information criterion (BIC) (Górecki et al. 2018) or with cross-validation, as described before.

14.2.1 Fourier Basis

The Fourier basis is often used for periodic or near-periodic data and is often useful for expanding functions with weak local characteristics and with an approximately constant curvature. It is not appropriate for data with discontinuities in the function or in low order derivatives (Ramsay et al. 2009).

The Fourier basis is created by the following functions:

$$\phi_1(t) = \frac{1}{\sqrt{P}}, \phi_2(t) = \frac{1}{\sqrt{\frac{P}{2}}} \sin(\omega t), \phi_3 = \frac{1}{\sqrt{\frac{P}{2}}} \sin(\omega t), \phi_4(t) = \frac{1}{\sqrt{\frac{P}{2}}} \cos(2\omega t),$$

$$\phi_5 = \frac{1}{\sqrt{\frac{P}{2}}} \cos(2\omega t), \phi_6(t) = \frac{1}{\sqrt{\frac{P}{2}}} \cos(3\omega t), \quad \phi_7 = \frac{1}{\sqrt{P/2}} \cos(3\omega t), \ldots,$$

where ω is related to period P by $\omega = 2\pi/P$, and in practical applications, this is often taken as the range of t values where the data are observed (Ramsay et al. 2009).

The graph on interval (0,8) of the first five of these functions $(0, 8)$ with period 4 is given in Fig. 14.1. The vertical dotted lines are the end of the subinterval that corresponds to the period of each function. This figure can be reproduced by the following R code:

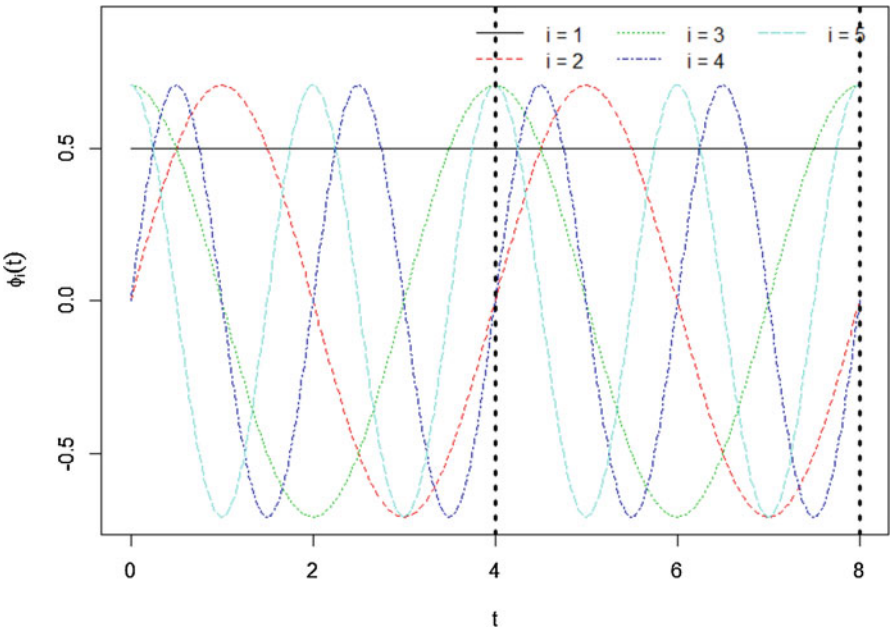


Fig. 14.1 Graph of the first five elements of the Fourier basis with period 4 on interval (0,8)

```
library(fda)
BF = create.fourier.basis(rangeval=c(0,8),nbasis=5,period=4)
plot(BF,xlab='t',ylab=expression(phi[i](t)),ylim=c(-.7,.9), lty=1:
5,lwd=1)
abline(v=seq(4,8,4),lty=3,lwd=3)
legend('topright',paste('i = ',1:5,sep=''),lty=1:5,col=1:5,
bty='n',lwd=1,ncol=3)
```

14.2.2 B-Spline Basis

A B-spline basis is typically used for nonperiodic data where the underlying function is locally smooth. The coefficients of a B-spline basis can be calculated quickly; they form a very flexible system because very good approximations of functions can be obtained even with a relatively small number of basis functions.

A B-spline is a type of spline that is a piecewise-polynomial continuous function and has a specific number of continuous derivatives on an interval. Specifically, a q + 1-order spline with interior knots T_j, $j = 1, \ldots, K$ (usually placed to take into account the data change points) on the observation interval $[0, T] = (0, T_1] \cup (T_1, T_2] \cup \ldots \cup (T_K, T]$ is a continuous function s_q such that in each subinterval $(T_{j-1}, T_j]$ there is a polynomial of degree q+1 that has continuous derivatives of order $q-1$ in

each knot, that is, the dth derivate of s_d, $s_q^{(d)}\left(T_j\right)$, is a continuous function in each knot, for each $d = 1, \ldots, q - 1$ (Quarteroni et al. 2000; Hastie et al. 2009).

Indeed, a q + 1-order B-spline basis is a basis for the q + 1-order spline function space on a given sequence of knots, that is, any spline function of order q + 1 can be represented as a linear combination of B-splines, and unlike other bases, the truncated power basis, for example, is very attractive numerically (Quarteroni et al. 2000; Hastie et al. 2009).

Once chosen, the order (q + 1) and the interior knots T_j, $j = 1, \ldots, K$,of a spline, because we need K + 1 polynomials (one for each of the K + 1 intervals) and Kq constraints (continuity of the B-spline in its interior knots + continuity of derivatives of order $q - 1$ in each knot), the number of basis functions is given by

$$L = (q + 1)(K + 1) - Kq = q + K + 1 = \text{order} + \text{number of interior knots}.$$

Practically, the position of the knots can be chosen according to the data change points or by allowing the observation time to determine their positions at appropriate percentiles (Fig. 14.2). For example, in R, if we want a B-spline of order 4 on the interval (0, 12) with three specific interior knots ($T_1 = 3$, $T_2 = 6$, $T_3 = 9$), this can be defined, plotted, and evaluated by applying the following code:

```
Order = 4 ; breaks = seq(0,12,3)
BS = create.bspline.basis(rangeval=c(0,12),norder=Order, breaks=breaks)
#Number of basis functions: 4 + 3
BS$nbasis
#Graphic of the seven basis functions
plot(BS,xlab='t')

#Evaluation of these seven basis functions in
tv= seq(0,12, length=100)
EBS = eval.basis(tv, basisobj=BS)
head(EBS)
matplot(tv,EBS,add=TRUE,type='o',pch='+')
```

Alternatively, if we want a B-spline basis of some order (q + 1) with a specific number of basis (L_2), in R it can be obtained similarly as before (Fig. 14.3). For example, a B-spline of order 4 with six bases will contain two (6–4) interior knots equally spaced and can be obtained with the following code:

```
Order = 4; nbasis = 6
BS = create.bspline.basis(rangeval=c(0,12),norder = Order,nbasis =
nbasis)
#Graphic of the 6 basis functions
plot(BS,xlab='t')
#Evaluation of this 6 basis functions in
tv= seq(0,12,length=100)
EBS = eval.basis(tv, basisobj=BS)
head(EBS)
matplot(tv,EBS,add=TRUE,type='o',pch='+')
```

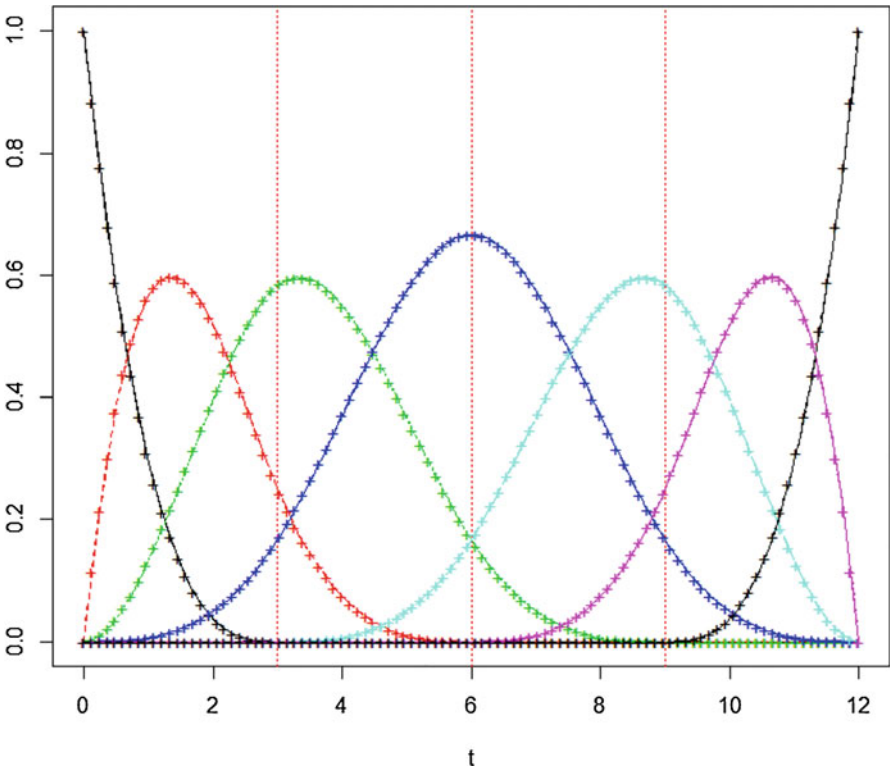


Fig. 14.2 Graphic of the seven B-spline basis functions of order 4 on interval (0,12) and interior knots $T_1 = 3$, $T_2 = 6$, and $T_3 = 9$

A B-spline basis of the same order and the same number of basis functions (Fig. 14.4), but with specific positions of the interior knots ($T_1 = 2$ and $T_2 = 7$), can be obtained by adding the argument breaks = c(0,2,7,12) to the function create.bspline.basis:

```
breaks = c(0,2,7,12)
Order = 4; nbasis = 6
BS = create.bspline.basis(rangeval=c(0,12),norder = Order,nbasis =
nbasis,
              breaks= breaks)
#Graphic of the six basis functions
plot(BS,xlab='t')
```

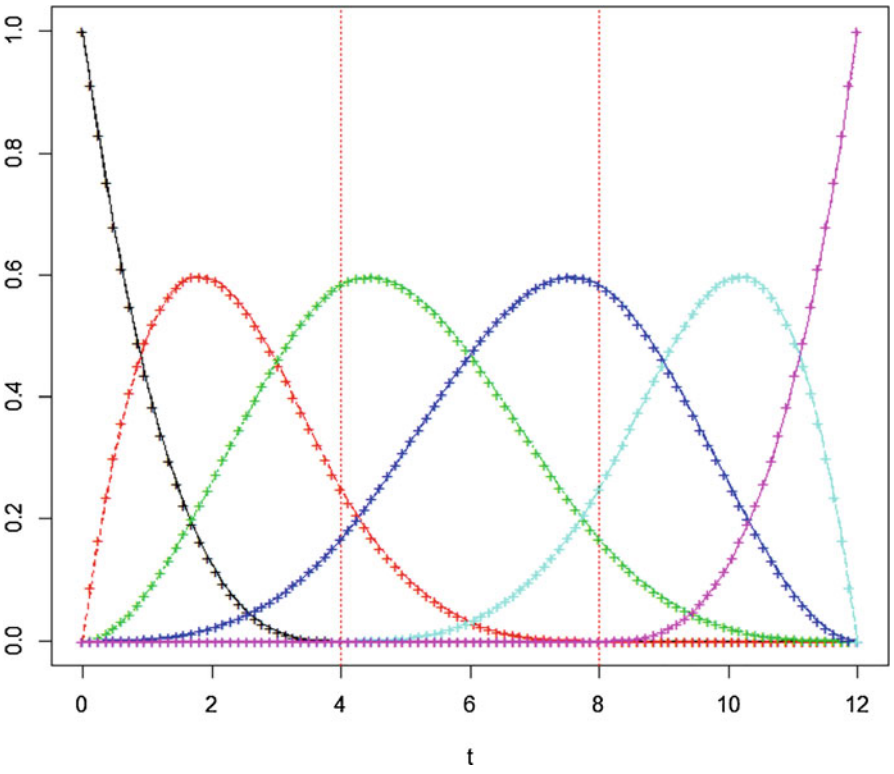
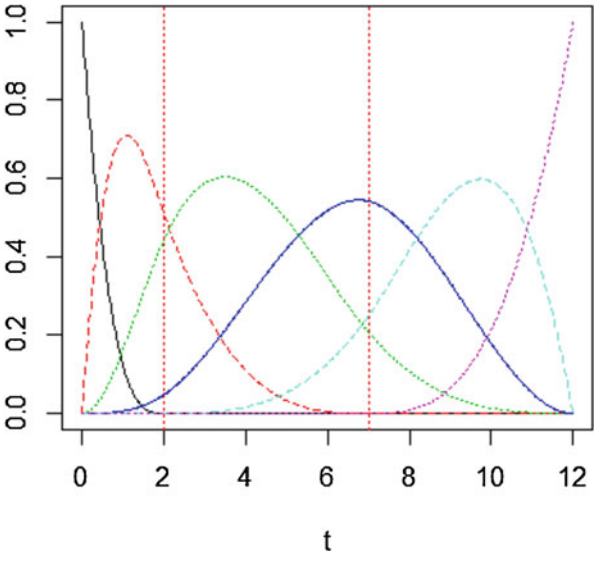


Fig. 14.3 Graphic of the six B-spline basis functions of order 4 on interval (0,12)

Fig. 14.4 Graphic of the six B-spline basis functions of order 4 on interval (0,12) with specific interior knots: $T_1 = 2$ and $T_2 = 7$



14.3 Illustrative Examples

Example 14.1
To illustrate how to use and get a better picture of the performance of Fourier and B-spline basis to represent functions, suppose that there is information on only 30 (tv) equispaced evaluations (xv) of an unknown function in interval (0,12):

tv = seq(0,12,length = 30)

xv = c(0.9, 0.924, 0.9461, 0.9658, 0.983, 0.9971, 1.008, 1.0152, 1.0187, 1.0181, 1.0133, 1.0042, 0.9906, 0.9727, 0.9504, 0.9237, 0.8928, 0.8579, 0.8192, 0.777, 0.7314, 0.683, 0.632, 0.5788, 0.5238, 0.4675, 0.4103, 0.3526, 0.2949, 0.2376).

The graphical representation of this information is given in Fig. 14.5, together with three representations of this using Fourier basis (5, 21, and 29 basis functions) with period 30 (range of the observation domain). From this we can see that a poor representation was obtained with five basis functions, while with the other numbers of basis used (21 and 29), almost equal and reasonable representations were obtained, except in the boundaries of the interval, which is related to the Gibbs

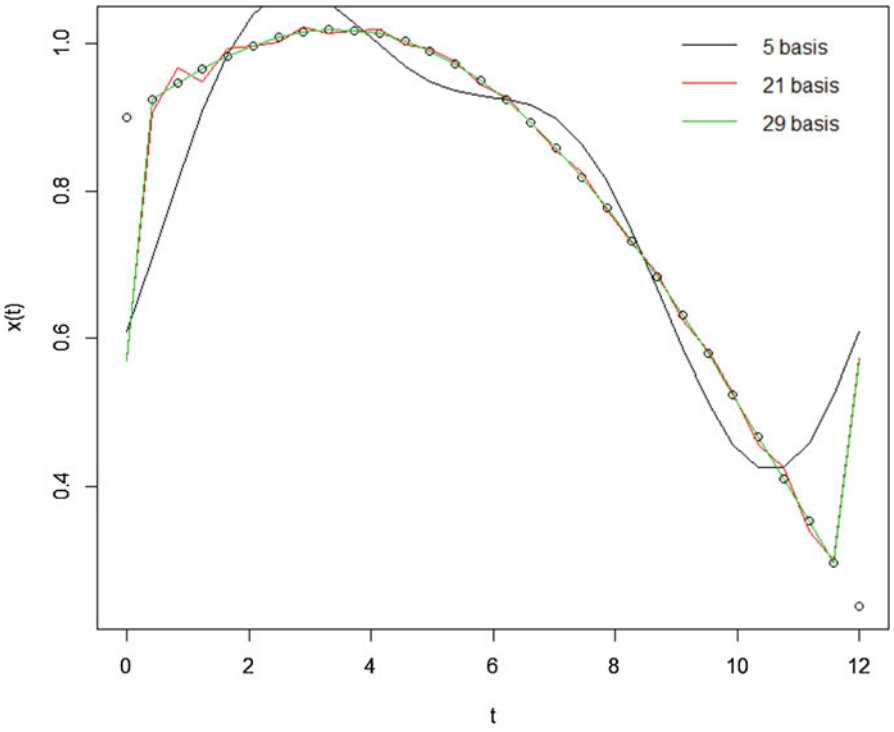


Fig. 14.5 Graphical representation of 30 evaluations (points) of a function in 30 equispaced time points in interval (0, 12), and representation of this using 5 (in black), 21 (in red), and 29 (in green) Fourier basis functions

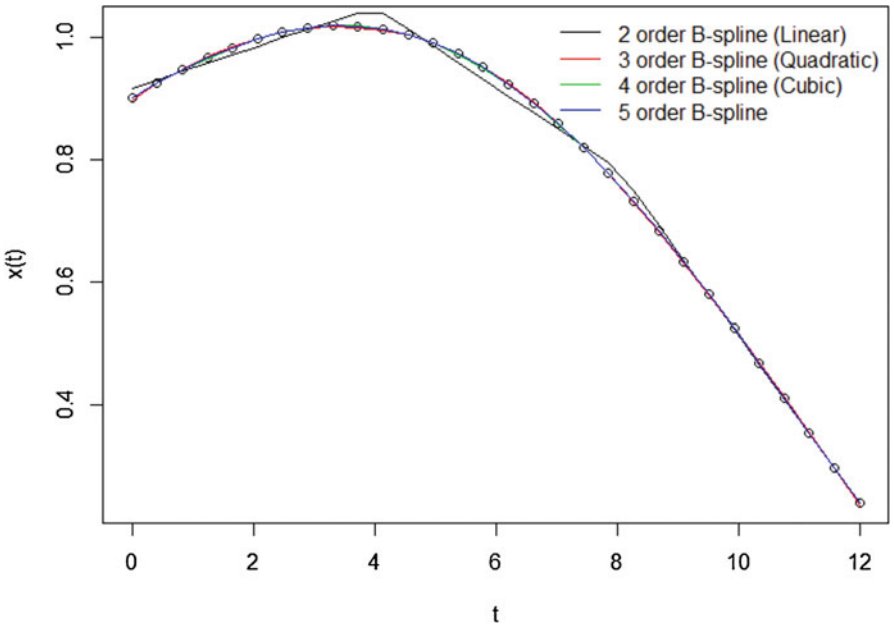


Fig. 14.6 Graphical representation of 30 evaluations (points) of a function in 30 equispaced time points in interval $(0, 12)$, and representation of this using B-splines of order 2 (shown in black), 3 (in red), 4 (in green), and 5 (in blue), each with two interior knots at $T_1 = 4$ and $T_2 = 8$

phenomenon, a spurious oscillation at the interval boundaries of an expansion in a Fourier series of a nonperiodic function (Shizgal and Jung 2003).

Figure 14.6 shows the same data but now together with representations of the latent function obtained by using B-splines of order 2, 3, 4, and 5, all with two interior knots ($T_1 = 4$ and $T_2 = 8$), to which correspond $2 + 2 = 4$, $2 + 3 = 5$, $2 + 4 = 6$, and $2 + 5 = 7$ basis functions, respectively. From this figure we can observe that the representation of order 3 is satisfactory, and this almost coincides with the representations obtained with orders 4 and 5.

```
#R code for B-spline representation
plot(tv,xv,type='p',xlab='t',ylab='x(t)')
breaks = seq(0,12,length=4)
Orderv = 2:5
for(i in 1:4)
{
    #A linear B-spline with two interior knots
    Order = Orderv[i]
    Degree = Order-1
    #No of basis functions= Order + length(breaks)-2
    nB = Order + length(breaks)-2
    BBS = create.bspline.basis(rangeval=c(0,12),norder=Order,
    breaks=breaks)
```

```
    EBBS = eval.basis(tv, basisobj=BBS)
    cv = solve(t(EBBS)%*%EBBS)%*%t(EBBS)%*%xv
    xv_p = EBBS%*%cv
    lines(tv,xv_p,col=i)
}
legend('topright',c('2 order B-spline (Linear)', '3 order B-spline
(Quadratic)',
          '4 order B-spline (Cubic)','5 order B-spline'),
   lty=c(1,1,1,1),col=1:4,bty='n')
```

Now, by using 3, 5, 7, and 10 B-spline basis of orders 2, 3, 4, and 5, respectively, the resulting representations are shown in Fig. 14.7. From this we can see that with five B-spline basis of order 3 (only two interior knots equally spaced were required), the representation started to be satisfactory, indicating flexibility in the B-spline basis.

R code to reproduce Fig. 14.7

```
plot(tv,xv,type='p',xlab='t',ylab='x(t)')
Orderv = 2:5
#No of basis functions = Order + length(breaks)-2
nBv = c(3,5,7,10)
#Interior points
K = nBv - Orderv
for(i in 1:4)
{
    BBS = create.bspline.basis(rangeval=c(0,12),nbasis=nBv[i],
                  norder=Orderv[i])
    EBBS = eval.basis(tv, basisobj=BBS)
    cv = solve(t(EBBS)%*%EBBS)%*%t(EBBS)%*%xv
    xv_p = EBBS%*%cv
```

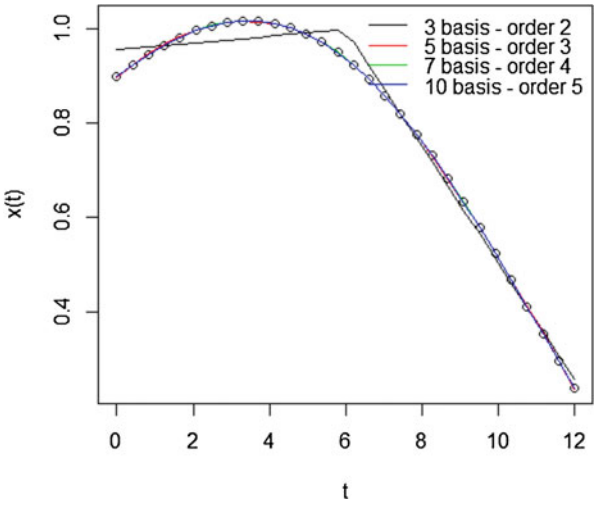


Fig. 14.7 Graphical representation of 30 evaluations (points) of a function in 30 equispaced time points in interval (0,12), and B-spline representation using 3, 5, 7, and 10 basis functions of orders 2, 3, 4, and 5, respectively

```
    lines(tv,xv_p,col=i)
}
legend('topright',paste(nBv,' basis - order ', Orderv,sep=''),
    lty=c(1,1,1,1),col=1:4,bty='n')
```

In general, more flexible curves can be obtained by increasing the order or the number of knots in the B-spline. However, overfitting and an increase in the variance can occur if the number of knots is increased, while an inflexible function with more bias may result by decreasing the number of knots (Perperoglou et al. 2019).

Example 14.2

Now to illustrate how to use the BIC and LOOCV as criteria to choose the number of basis functions in a Fourier or B-spline (for a chosen order) representation, we retake the data points in Example 14.1 but perturbed by a random Gaussian noise (see Fig. 14.8).

For these data, Fig. 14.9 shows the value of the BIC criterion corresponding to the use of different numbers of basis functions in Fourier and B-spline representations. In both cases, the lowest value of this criterion was obtained with five basis functions and in all cases, the BIC criterion was better with the B-spline. The representation of the data points with this optimal number of basis functions is also shown in Fig. 14.8, where we can visually judge the better representation of the B-spline, because this resembles the non-perturbed latent function presented in Example 14.1. Similar results were obtained when using the LOOCV strategy: 7 and 6 basis for Fourier and B-spline were required, respectively, and the implied representation can also be observed in Fig. 14.9. These figures can be reproduced by the following R code:

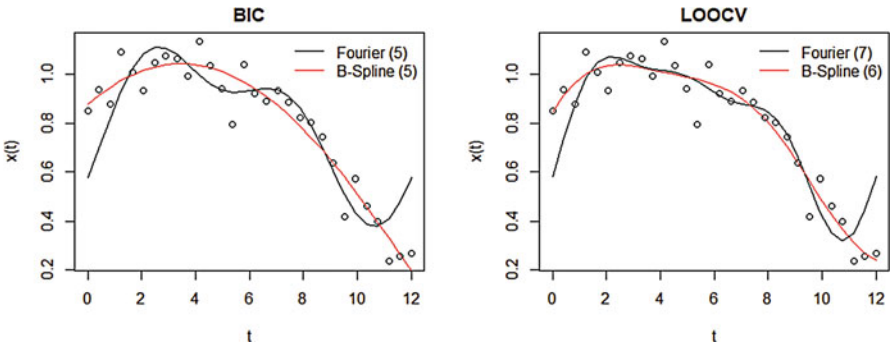


Fig. 14.8 Graphical representation of the perturbed 30 data points in Example 14.1 and Fourier and B-spline representations using 5, 6, and 7 basis functions (L_2) in both. The left panel is the optimal representation obtained with the BIC and the right panel corresponds to the optimal representation obtained with the LOOCV

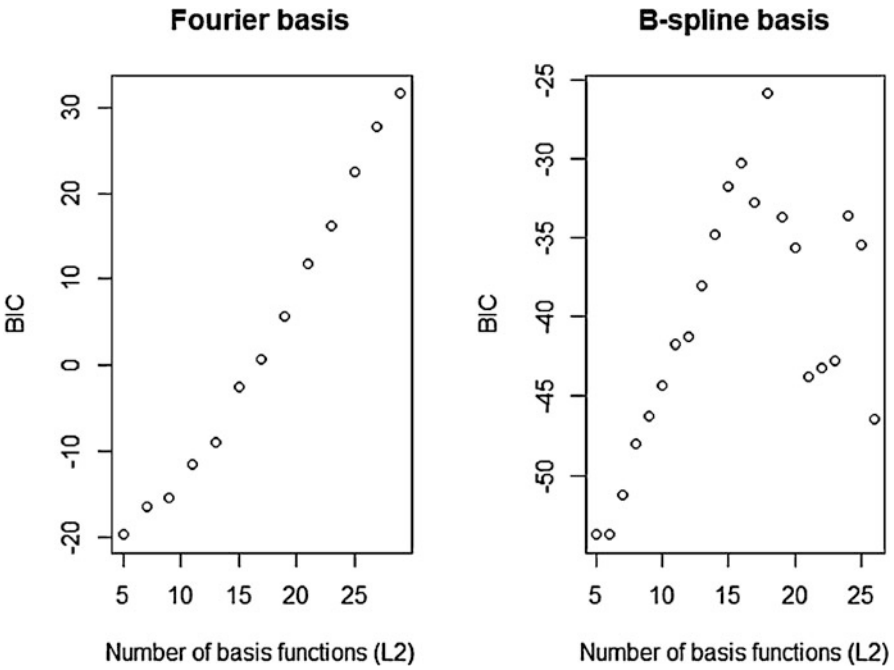


Fig. 14.9 Behavior of the BIC for different numbers of basis *functions* (L_2) in Fourier and B-spline representations, for the perturbed data set of Example 14.1

```
rm(list=ls())
#------
#Example 14.2: Perturbed data points of Example 14.1
#------
library(fda)
#Time points where the functions was observed
tv = seq(0,12,length=30)
#Values of the function in 30 points
xv = c(0.9, 0.924, 0.9461, 0.9658, 0.983, 0.9971, 1.008, 1.0152, 1.0187,
1.0181, 1.0133, 1.0042, 0.9906, 0.9727, 0.9504, 0.9237, 0.8928, 0.8579,
0.8192, 0.777, 0.7314, 0.683, 0.632, 0.5788, 0.5238, 0.4675, 0.4103,
0.3526, 0.2949, 0.2376)
set.seed(1)
xv = xv + rnorm(length(xv),0,0.10*mean(xv))
#plot(tv,xv,type='p',xlab='t',ylab='x(t)')
#Fourier
library(fda)
m = length(xv)
nbFv = seq(5,m-1,2)
BICFv = rep(0, length(nbFv))
AICFv = BICFv
for(l in 1:length(nbFv))
{ BF=create.fourier.basis(rangeval=c(0,12),nbasis=nbFv[l],
period=diff(range(tv)))
```

```
 EBF = eval.basis(tv, basisobj=BF)
 cv = solve(t(EBF)%*%EBF)%*%t(EBF)%*%xv
 xv_p = EBF%*%cv
 sigma2 = mean((xv-xv_p)^2)
 ll = sum(dnorm(xv,xv_p,sqrt(sigma2),log = TRUE))
 BICFv[l] = -2*ll+(dim(EBF)[2]+1)*log(m)
 AICFv[l] = -2*ll+2*(dim(EBF)[2]+1)
}
#B-spline
library(fda)
Order = 4
#No of basis functions = Order + length(breaks)-2
nbBSv = (Order+1):(m-4)
BICBSv = rep(0,length(nbBSv))
for(l in 1:length(nbBSv))
{
 BBS = create.bspline.basis(rangeval=c(0,12),norder=Order,nbasis =
 nbBSv[l])
 EBBS = eval.basis(tv, basisobj=BBS)
 cv = solve(t(EBBS)%*%EBBS)%*%t(EBBS)%*%xv
 xv_p = EBBS%*%cv
 muv = EBBS%*%cv
 sigma2 = mean((xv-muv)^2)
 ll = sum(dnorm(xv,muv,sqrt(sigma2),log = TRUE))
BICBSv[l] = -2*ll+(nbBSv[l]+1)*log(m)
}
#Behavior of the BIC for different models obtained using various
#number of Fourier or B-spline basis functions
par(mfrow=c(1,2))
plot(nbFv,BICFv,xlab='Number of basis functions (L2)', ylab='BIC',
   main='Fourier basis')
plot(nbBSv,BICBSv,xlab='Number of basis functions (L2)',ylab='BIC',
   main='B-spline basis')
#Fourier and B-spline representations with optimal number of
#basis functions as chosen by BIC
#Fourier
nboF_BIC = nbFv[which.min(BICFv)]
nboF_BIC
plot(tv,xv,type='p',xlab='t',ylab='x(t)',
   main='')#Optimal representation using BIC')
BF = create.fourier.basis(rangeval=c(0,12),nbasis=nboF_BIC,
            period=diff(range(tv)))
EBF = eval.basis(tv, basisobj=BF)
cv = solve(t(EBF)%*%EBF)%*%t(EBF)%*%xv
xv_p = EBF%*%cv
lines(tv,xv_p,col=1)
#B-spline
nboB_BIC = nbBSv[which.min(BICBSv)]
nboB_BIC
BBS = create.bspline.basis(rangeval=c(0,12),norder=Order,
             nbasis = nboB_BIC)
EBBS = eval.basis(tv, basisobj=BBS)
cv = solve(t(EBBS)%*%EBBS)%*%t(EBBS)%*%xv
```

```
xv_p = EBBS%*%cv
lines(tv,xv_p,col=2)
legend('topright',paste(c('Fourier (','B-Spline ('),
            c(nboF_BIC,nboB_BIC),c(')',')'),sep=''),
    col=1:2,lty=rep(1,2),bty='n')
#Choosing the optimal number of basis functions using 1FCV
PRESS_f<-function(A)
{
 Res = residuals(A)
 sum((Res/(1-hatvalues(A)))^2)
}
#Fourier basis
CVFv = nbFv
for(l in 1:length(nbFv))
{   BF=create.fourier.basis(rangeval=c(0,12),nbasis=nbFv[l],
period=diff(range(tv)))
 EBF = eval.basis(tv, basisobj=BF)
 A = lm(xv~0+EBF)
 CVFv[l] = PRESS_f(A)
}
plot(nbFv,CVFv,xlab='No. of basis functions',ylab='PRESS')
nboF = nbBSv[which.min(CVFv)]
nboF

#B-spline basis
CVBSv = nbBSv
for(l in 1:length(nbBSv))
{
 BBS = create.bspline.basis(rangeval=c(0,12),norder=Order,nbasis =
 nbBSv[l])
 EBBS = eval.basis(tv, basisobj=BBS)
 A = lm(xv~0+EBBS)
 CVBSv[l] = PRESS_f(A)
}
plot(nbBSv[1:5],CVBSv[1:5],xlab='No. of basis',ylab='PRESS')
nboBS = nbBSv[which.min(CVBSv)]
nboBS

#Fourier and B-spline representations with optimal number of
#basis functions as choosen by BIC
par(mfrow=c(1,2))
plot(tv,xv,type='p',xlab='t',ylab='x(t)',
   main='BIC')
#Fourier
BF = create.fourier.basis(rangeval=c(0,12),nbasis=nboF_BIC,
             period=diff(range(tv)))
EBF = eval.basis(tv, basisobj=BF)
cv = solve(t(EBF)%*%EBF)%*%t(EBF)%*%xv
xv_p = EBF%*%cv
lines(tv,xv_p,col=1)
#B-spline
BBS = create.bspline.basis(rangeval=c(0,12),norder=Order,
              nbasis = nboB_BIC)
```

```
EBBS = eval.basis(tv, basisobj=BBS)
cv = solve(t(EBBS)%*%EBBS)%*%t(EBBS)%*%xv
xv_p = EBBS%*%cv
lines(tv,xv_p,col=2)
legend('topright',paste(c('Fourier (','B-Spline ('),c(5,5),
            c(')',')'),sep=''),
    col=1:2,lty=rep(1,2),bty='n')

#Optimal representation with 1FCV
plot(tv,xv,type='p',xlab='t',ylab='x(t)',
   main='LOOCV')
#Fourier
BF = create.fourier.basis(rangeval=c(0,12),nbasis=nboF,
             period=diff(range(tv)))
EBF = eval.basis(tv, basisobj=BF)
cv = solve(t(EBF)%*%EBF)%*%t(EBF)%*%xv
xv_p = EBF%*%cv
lines(tv,xv_p,col=1)
#B-spline
BBS = create.bspline.basis(rangeval=c(0,12),norder=Order,
              nbasis = nboBS)
EBBS = eval.basis(tv, basisobj=BBS)
cv = solve(t(EBBS)%*%EBBS)%*%t(EBBS)%*%xv
xv_p = EBBS%*%cv
lines(tv,xv_p,col=2)
legend('topright',paste(c('Fourier (','B-Spline ('),c(nboF,nboBS),
            c(')',')'),sep=''),
    col=1:2,lty=rep(1,2),bty='n')
```

Example 14.3

Now we will consider the prediction of wheat grain yield (tons/ha) using hyperspectral image data. For this example, we consider part of the data used in Montesinos-López et al. (2017a, b): 20 lines and three environments. For each individual plant, the reflectance, $x(t_j)$, of its leaves was measured at $m = 250$ wavelengths (from 392 to 851 nm were measured) and at different stages of its growth, but the information used here corresponds to one of these stages.

Figure 14.10 shows the measured reflectance corresponding to 60 observations, where the colors of these observations indicate that they belong to the same environment. The Fourier and B-spline representations of all these curves are shown in Fig. 14.11, where the number of basis used in each case were 29 and 16, respectively; they are the medians of the most frequently selected number of basis functions (29 for Fourier, and 12, 16, and 73 for B-spline) by the BIC across all the curves (see Appendix 1 for the R code to reproduce these results).

The observed values versus the predicted values of the response, corresponding to the Fourier and B-spline representations of the covariate, are shown in Fig. 14.12; in both cases, $L_1 = 21$ basis functions were used to represent the beta coefficient

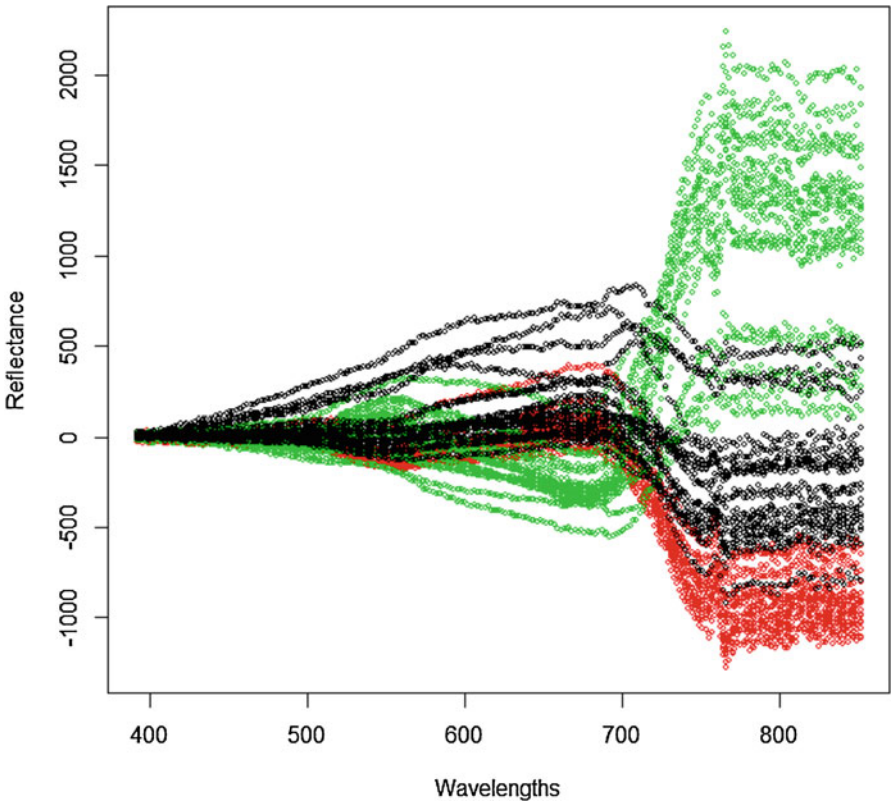


Fig. 14.10 Reflectance of leaves of 60 individuals measured in 250 wavelengths (29, 394, 394, . . ., 851). The colors indicate the environment where the individuals were measured

function $\beta(t)$. The fitted model appears to give almost the same results with both representations.

To let the data speak for themselves about a reasonable value for L_1 to represent $\beta(t)$, the BIC was used in both representations. For the Fourier case, $L_1 = 11$ was the optimal value, while $L_1 = 14$ was the optimal value for the B-spline basis (see Appendix 1 for the R code). The predicted and residual values obtained with these optimal representations are shown in Fig. 14.13, from which it is difficult to choose the best one because they gave almost the same results.

Because the B-spline appears to give a better covariate representation (see Fig. 14.11) and a similar predicted value as that given by the Fourier basis (Fig. 14.13), we can take a more informed decision in terms of the prediction accuracy of the response, with both representations. To this end, we used ten random partitions, where in each partition, 20% of the total data set was used to measure prediction accurary and the rest was used to fit (train) the model. The results are shown in Table 14.1 and we can see that, on average, the Fourier basis was favored,

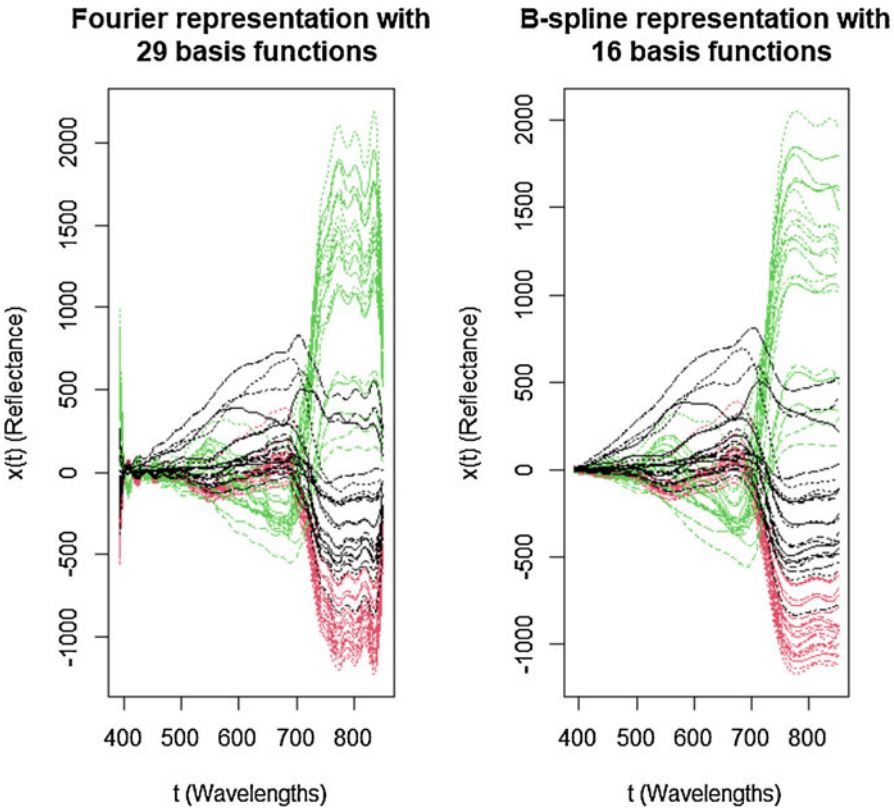


Fig. 14.11 Fourier and B-spline representations with the "optimal" number of basis *functions* obtained with the BIC across all the curves

because on average across all the partitions, this type of basis provided lower MSE in 6 out of 10 partitions; for this reason, it is considered the best option. The same conclusion was reached by comparing the BIC values of the corresponding Fourier (149.8048) and B-spline (159.5854) representations.

14.4 Functional Regression with a Smoothed Coefficient Function

As mentioned earlier, in the representation of the functional predictor ($x(t)$), one way to control the smoothness when determining the beta coefficient function, $\beta(t)$, is by introducing a regularization term:

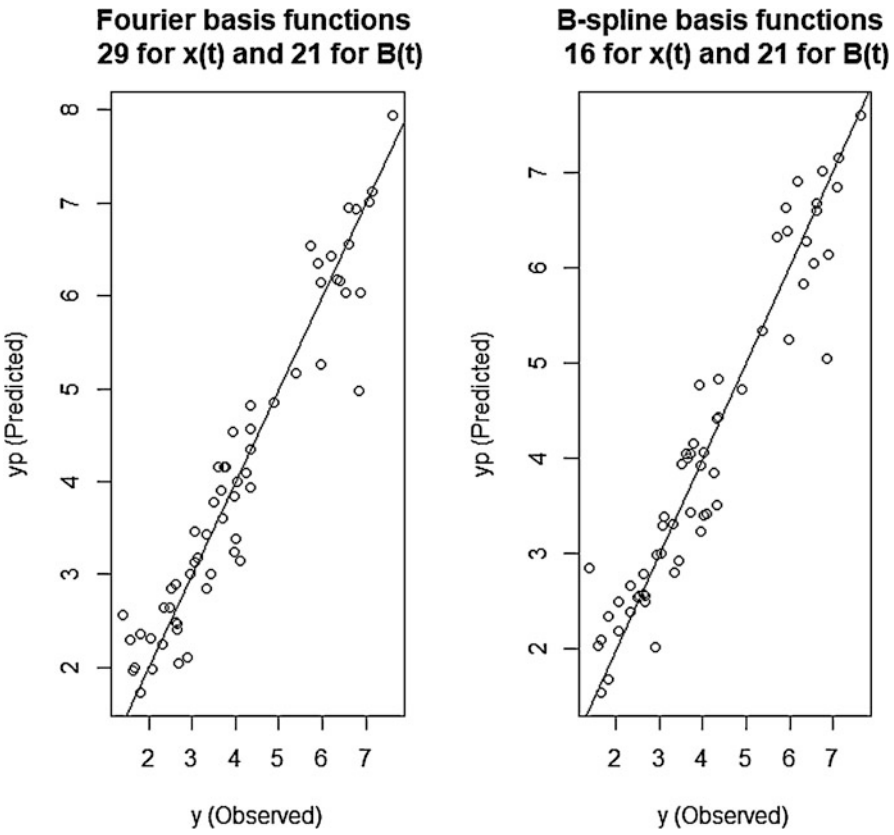


Fig. 14.12 Graph displaying the observed versus fitted values with two functional regression models: the left panel was obtained by using 29 Fourier basis *functions* for the covariate function and 21 Fourier basis *functions* for $\beta(t)$; the right panel was obtained by using 16 B-spline basis *functions* for the covariate function and 21 B-spline basis *functions* for $\beta(t)$

$$\mathrm{SSE}_\lambda(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y_i - \mu - \sum_{l=1}^{L_1} x_{il}\beta_l \right)^2 + \lambda J_\beta, \tag{14.10}$$

where J_β is the penalty term and λ is a smoothing parameter that represents a compromise between the fit of the model to the data (first term) and the smoothness of the function $\beta(\cdot)$ (second term). When $\lambda = 0$, the problem is reduced to that of least squares (or maximum likelihood under normal errors) where there is no penalty, and when λ increases, the roughness is highly penalized to the extent that $\beta(t)$ can be constant.

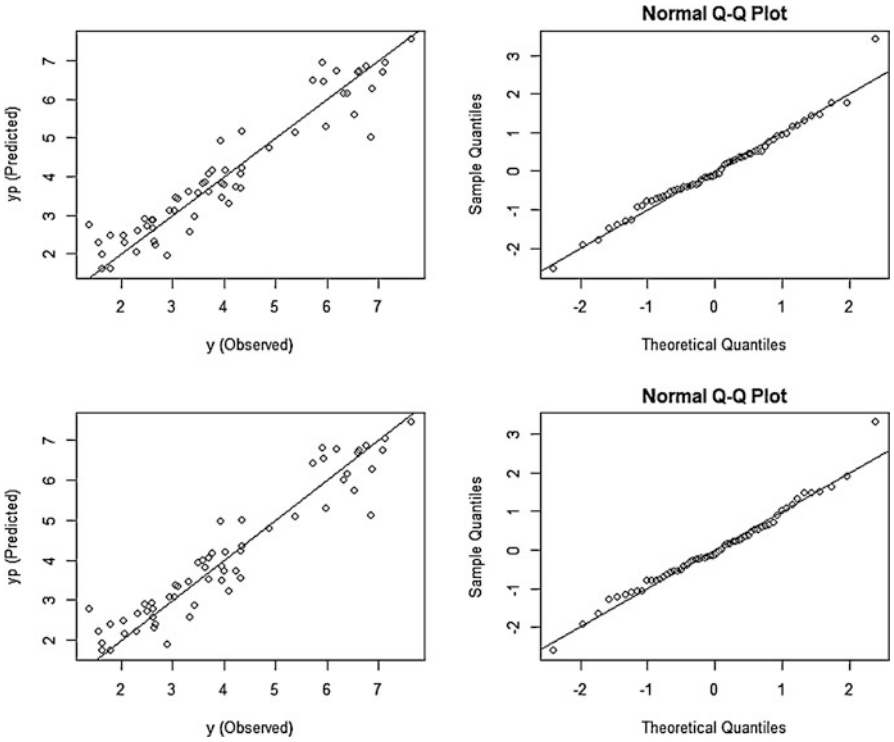


Fig. 14.13 Observed versus predicted values and normal Q-Q plot of the residuals obtained with Fourier (above) and B-spline (below) representations of both sets of covariates (L_2=29 Fourier basis functions and L_2=16 B-spline basis functions) and beta coefficient functions ($L_1 = 11$ Fourier basis functions and $L_1 = 14$ B-spline Fourier basis functions)

Table 14.1 Mean square error (MSE) of prediction for 10 random partitions for Fourier and B-spline representations

Partition	Fourier	B-spline
	MSE	MSE
1	0.1717	0.2993
2	0.6049	0.6487
3	0.4385	0.4559
4	0.5006	0.4910
5	0.6164	0.6048
6	0.5403	0.6453
7	0.1456	0.1396
8	0.5004	0.5737
9	0.7551	0.7435
10	0.8042	0.9250
Average (SD)	0.5077 (0.216)	0.5526 (0.2221)

Often the penalty term J_β is based on the integrated pth order derivatives (Usset et al. 2016):

$$J_\beta = \int_0^T \left[\frac{d^p}{dt^p}\beta(t)\right]^2 dt, \tag{14.11}$$

where $\frac{d^p}{dt^p}\beta(t)$ is a derivative of order p of the function $\beta(t)$. With the representation (14.2) of $\beta(t)$, J_β can be expressed as

$$J_\beta = \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{P}\boldsymbol{\beta},$$

where P is a square matrix with entries $P_{ij} = \int_0^T \phi_i^{(p)}(t)\phi_j^{(p)}(t)$, $i, j = 1, \ldots, L_1$, and $\phi_i^{(p)}(t)$ is a derivate of order p of $\phi_i(t)$. Typical chosen values of p are 1 and 2.

A smoothed solution of the function $\beta(t)$ can be obtained by minimizing (14.10) with respect to the parameters β_l, $l = 1, \ldots, L_1$. However, because this solution depends on the smoothing parameter, this needs to be determined. For this reason, as in the Ridge and Lasso regression models described in early chapters, here a cross-validation method is adopted first, and a Bayesian approach will be described later.

Under the penalty term (14.11), the penalized sum of squared errors (14.10) can be written as

$$\mathrm{SSE}_\lambda(\boldsymbol{\beta}) = ||\boldsymbol{y} - \mathbf{1}_n\mu - \boldsymbol{X}^*\boldsymbol{\beta}^*||^2 + \lambda\boldsymbol{\beta}^{*\mathrm{T}}\boldsymbol{D}\,\boldsymbol{\beta}^* = SSE_\lambda(\boldsymbol{\beta}^*), \tag{14.12}$$

where $\boldsymbol{X}^* = \boldsymbol{X}\boldsymbol{\Gamma}$, $\boldsymbol{\beta}^* = \boldsymbol{\Gamma}^{\mathrm{T}}\boldsymbol{\beta}$, and $\boldsymbol{P} = \boldsymbol{\Gamma}\boldsymbol{D}\boldsymbol{\Gamma}^{\mathrm{T}}$ is the spectral decomposition of the penalty matrix $\boldsymbol{P}$. Note that when the matrix $\boldsymbol{P}$ is not of full rank, the penalty term in (14.12) is reduced to $\lambda\boldsymbol{\beta}^{*\mathrm{T}}\boldsymbol{D}\,\boldsymbol{\beta}^* = \lambda\boldsymbol{\beta}_1^{*\mathrm{T}}\boldsymbol{D}_1\,\boldsymbol{\beta}_1^*$, where $\boldsymbol{D}_1$ is $\boldsymbol{D}$ but without the rows and columns corresponding to the eigenvalues equal to 0 of $\boldsymbol{P}$. So, the corresponding smoothed solution of $\beta(t)$ can be obtained as

$$\widehat{\beta}(t) = \sum\nolimits_{l=1}^{L_1} \widehat{\beta}_l\phi_l(t),$$

where $\widehat{\boldsymbol{\beta}} = \boldsymbol{\Gamma}\widehat{\boldsymbol{\beta}}^*$ and $\widehat{\boldsymbol{\beta}}^*$ is the solution of (14.12), which also can be obtained with the *glmnet* R package.

Example 14.4

To exemplify the penalized estimation of functional regression (14.10) with penalty (14.11), here we retake the data used in Example 14.3. To compare the prediction accuracy of this with the non-penalized functional regression described in the previous section, 100 random partitions were used, and in each, 80% of the data set was used to train the model and the rest to evaluate the prediction performance. When training the model, an inner five-fold cross-validation was used to choose the optimal parameter (λ) and estimate the β_l's coefficients. This was done using Fourier and B-spline basis in the representation of the beta function, and in both cases, two

basis were used. In both cases, the penalty matrix (14.11) and the elements of its spectral decomposition (14.12) can be computed in R as

```
#Penalty matrix of derivative of order p
P_mat = eval.penalty(basisobj =Phi,Lfdobj=p)
ei_P = eigen(P_mat)
#Espectral descompositin of P_mat
gamma = ei_P$vectors #Γ
dv = ei_P$values#Eigenvalues of P_mat, elements of diagonal of D
dv = ifelse(dv<1e-10,0,dv)
```

where Phi is a created basis in R (Fourier or B-spline) and p is a nonnegative integer for the order of the derivative in the penalty matrix to be used. Once the penalty matrix is computed, the training of the model in (14.12) can be done in R as

```
Xa = X_F%*%gamma
A_PFR = cv.glmnet(x =Xa,y = y ,alpha = 0, nfolds=k, penalty.factor=dv,
         standardize=FALSE,maxit=1e6)
```

where Xa is $\boldsymbol{X}^*$, y is the vector with the corresponding values of the response variable, k is an integer used to specify the inner k cross-validation to train the model and choose the "optimal" value of the smoothing parameter, and dv are the eigenvalues of the penalty matrix used to indicate different penalties of the β_j^* as required in (14.12), and standardize = FALSE to indicate the non-required standardization of the columns of $\boldsymbol{X}^*$. See Appendix 2 for a complete R code used to obtain the results of this example.

When using Fourier representation and first derivative penalization, for the 100 random partitions, in Fig. 14.14 is shown the MSE obtained with the penalized functional regression (PFR) against the MSE corresponding to the non-penalized functional regression (FR): in 78 out of 100 partitions, the PFR resulted in a better MSE (0.7465 vs. 0.5545 on average). Furthermore, in the partitions where the FR was better (20%), the average MSE of the PFR was 35.48% greater (0.4051 vs. 0.5561), while in those where the PFR was better, the average MSE obtained with the FR was 51.55% greater (0.8428 vs. 0.5561).

Now, for the B-spline representation and first derivative penalization, the corresponding results are also shown in Fig. 14.14. In this case, in 55 out of 100 partitions, the PFR resulted in a better MSE (0.5822 vs. 0.5630 on average). Furthermore, in the partitions where the FR was better (20%), the average MSE of the PFR was 27.18% greater (0.4887 vs. 0.6216), while in those where the PFR was better, the average MSE obtained with the FR was 27.87% greater (0.6587 vs. 0.5151). So, for the Fourier basis representation, the results obtained when using penalization in the estimation of the beta function $\beta(\cdot)$ differ from those obtained when no penalization is used, while with the B-spline representation the difference is negligible. This is perhaps because of the natural smoothing of the B-spline relative to the Fourier basis.

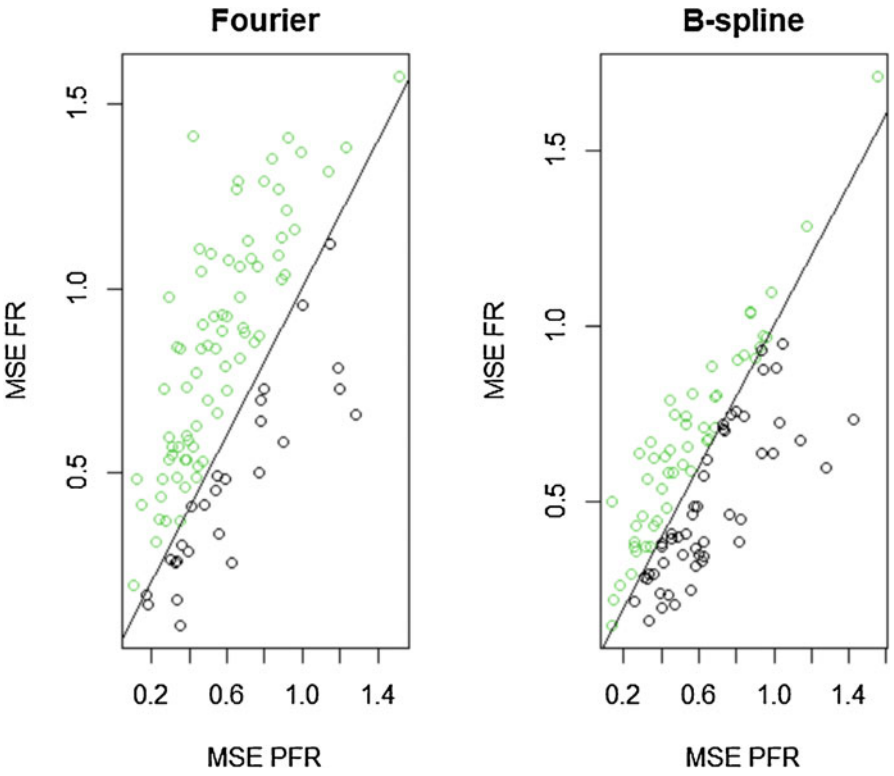


Fig. 14.14 Mean square error (MSE) of prediction for penalized functional regression (PFR) versus MSE of functional regression (FR). Shown in green are the cases where the MSE obtained with the PFR is less than the FR

For second derivative penalization using Fourier representation, in 74 out of 100 partitions and on average, the MSE of PFR also resulted better than the FR (0.7465 vs. 0.5754). Indeed, in the cases where PFR was worse, the average MSE of this was 34.87% greater than the corresponding FR, and in the cases where FR was worse, the average MSE was 51.78% greater.

For B-spline basis with second derivative penalization, on average the FR was better than the PFR (0.5822 vs. 0.6009), but this resulted in a smaller MSE only in 50 out of 100 partitions. Additionally, in the case where FR was better, the average MSE of the PFR was 34.48% greater than the average of FR, while in the other half of the cases, the average MSE of the FR was only 24.25% greater than the average MSE of PFR.

14.5 Bayesian Estimation of the Functional Regression

Similar to what was done in Chaps. 3 and 6, the penalized sum squared of error solution in (14.12) with penalty term (14.11) coincides with the mean/mode of the posterior distribution of $\boldsymbol{\beta}$, in the multiple linear regression $y_i = \mu + \sum_{l=1}^{L_1} x_{il}\beta_l + \epsilon_i$ ($\boldsymbol{y} = \mathbf{1}_n\mu + \boldsymbol{X\beta} + \boldsymbol{\epsilon}$), with prior distribution $\boldsymbol{\beta} \sim N\left(\mathbf{0}, \sigma_\beta^2 \boldsymbol{P}^{-1}\right)$, with $\sigma_\beta^2 = \frac{\lambda}{\sigma^2}$. So, from here a Bayesian formulation (PBFR) for the smoothed solution of the coefficient function ($\beta(t)$) in the functional regression model (14.1) can be completed by assuming the following priors for the rest of the parameters: $\sigma^2 \sim \chi_{\nu,S}^{-2}$ and $\sigma_\beta^2 \sim \chi_{\nu_\beta,\ S_\beta}^{-2}$, where $\chi_{\nu,S}^{-2}$ denotes a scaled inverse Chi-squared distribution with shape parameter ν and scale parameter S. When $\boldsymbol{P}$ is not of full rank, little change is needed over the prior distribution of the $\boldsymbol{\beta}$ coefficients: for example, in the Fourier basis, the first element of this is the constant function, so entries in the first row and first column of $\boldsymbol{P}$ are equal to 0.

Similarly, the Bayesian formulation of regression models can be expressed as $\boldsymbol{y} = \mathbf{1}_n\mu + \boldsymbol{X}^*\boldsymbol{\beta}^* + \boldsymbol{\epsilon}$, with the same prior distribution, except that now $\boldsymbol{\beta}^* \sim N\left(\mathbf{0}, \sigma_\beta^2 \boldsymbol{I}_{L_1}\right)$ and $\boldsymbol{X}^* = \boldsymbol{X\Gamma D}^{-1/2}$, where $\boldsymbol{D}^{-\frac{1}{2}}$ is the inverse of the squared root matrix of $\boldsymbol{D}$. This equivalence is the same as Bayesian Ridge Regression (BRR) described in Chap. 6, so this can be implemented with the BGLR R package.

Example 14.5
Here we continue with the data set of Example 14.3, but now in another 100 random partitions, we added the Bayesian prediction to explore the prediction performance. Part of the code for implementing this model is given next, but the complete code used is given in **Appendix 3**.

```
#Number of Fourier and B-spline basis functions to represent the
covariable
#function
nbFo = 29
nbBSo = 16

#Functional regression with Fourier and B-splines
#Computing X for Fourier representation with L1 = 21 Fourier basis
functions
#for the beta function
L1 = 21
P = diff(range(Wv))
Psi_F = create.fourier.basis(range(Wv), nbasis = nbFo,
                period = P)
SF_mat = smooth.basisPar(argvals = Wv,
             y =t(X_W), fdobj = Psi_F,lambda=0,
             Lfdobj = 0)
Phi_F = create.fourier.basis(range(Wv), nbasis = L1,
                period = P)
```

```
Q   = inprod(Phi_F,Psi_F)
c_mat = t(SF_mat$fd$coefs)
X_F  = c_mat%*%t(Q)
dim(X_F)

#Computing X for B-spline representation with L1 = 21 B-spline basis
functions
#for the beta function
L1 = 21
Psi_BS = create.bspline.basis(range(Wv), nbasis = nbBSo)
SBS_mat = smooth.basisPar(argvals = Wv,
             y =t(X_W), fdobj = Psi_BS,lambda=0,
             Lfdobj = 0)
Phi_BS = create.bspline.basis(range(Wv), nbasis = L1,
               norder= 4)
Q   = inprod(Phi_BS,Psi_BS)
c_mat = t(SBS_mat$fd$coefs)
X_BS  = c_mat%*%t(Q)
dim(X_BS)

#Smoothing estimation using Fourier representation for the functional
#covariable and L1 = 21 Fourier basis functions for the beta function
par(mfrow=c(1,2))
library(glmnet)
#Penalization with first derivate
P_mat_F = eval.penalty(Phi_F,Lfdobj=1)
ei_P = eigen(P_mat_F)
gamma = ei_P$vectors
dv = ei_P$values
dv = ifelse(dv<1e-10,0,dv)

X_Fa = X_F%*%gamma

#Grid of lambda values obtained by varying the proportion of variance
explained
#by the functional predictor
lamb_FR_f<-function(Xa,dv,K=100,li=1e-1,ls=1-1e-12)
{
 Pos = which(dv<1e-10)
 D_inv = diag(1/dv[-Pos])
 PEV = seq(li,ls,length=K)
 Xa  = Xa[,-Pos]
 lambv = (1-PEV)/PEV*mean(diag(Xa%*%D_inv%*%t(Xa)))
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}
lambda = lamb_FR_f(X_Fa,dv,K=1e2)

y=yv
library(BGLR)
#Linear predictor specification in BGLR for the PBFR model
Pos = which(dv>0)
```

```
#Matrix design for the non-penalized columns of X^a
X_Fa1 = X_Fa[,-Pos]
#Matrix design for the penalized columns of X^a
X_Fa2 = X_Fa[,Pos]%*%diag(1/sqrt(dv[Pos]))
ETA = list(list(X=X_Fa1,model='FIXED'),list(X=X_Fa2,model='BRR'))

#Linear predictor specification in BGLR for the BFR model
ETA_NP = list(list(X=X_F[,1],model='FIXED'),list(X=X_F[,-1],
model='BRR'))

#Random cross-validation
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:RP)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_F[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_F)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,
          penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)

 #BGLR
 y_NA = y
 y_NA[Pos_tst] = NA
 A_BGLR = BGLR(y_NA,ETA= ETA,nIter=1e4, burnIn = 1e3,verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR[p] = mean((y_tst-yp_tst)^2)

 A_BGLR = BGLR(y_NA,ETA= ETA_NP,nIter=1e4, burnIn = 1e3,
verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR_NP[p] = mean((y_tst-yp_tst)^2)

 cat('Partition = ', p,'\n')

}
MSEP_df
```

With the Fourier basis and first derivative penalization, the average MSE (SD) across 100 random partitions were 0.7744(0.3412), 0.6338(0.2871), 0.8585 (0.2929), and 0.8092(0.2848) for the functional regression (FR), penalized functional regression (PFR), penalized Bayesian functional regression (PBFR), and Bayesian functional regression (BFR, $\boldsymbol{\beta} \sim N\left(\mathbf{0}, \sigma_{\beta}^{2}\boldsymbol{I}_{L_1}\right)$), respectively. In 75, 81, and 77 out of the 100 random partitions, the MSE of the PFR was better than the partitions obtained with the FR, PBFR, and BFR, respectively. And only in 5 out of 100 cases, the MSE of the PBFR was better than the BFR, making the penalty term in the Bayesian estimation non-important and indeed harmful (see **Appendix 3** for the R code used).

With the B-spline basis and first derivative penalization, the PFR (0.5718 (0.2669)) also resulted better on average than FR (0.6012(0.2681)), PBFR (0.8475 (0.3333)), and BFR (0.7982(0.3112)). Here, in 62 out of 100 random partitions, the MSE of the PFR was less than the MSE of the FR, while in 84 and 80 out of the 100 random partitions, they were better than the PBFR and BFR, respectively. Also, taking into account the penalty term in the Bayesian prediction was not so important because in only 8 out of the 100 random partitions, the MSE of the PBFR was less than the MSE corresponding to the BFR (see **Appendix 3** for the R code used).

When using the penalty matrix based on second derivatives, in each case (Fourier and B-spline), the results were similar.

The Bayesian formulation can be extended easily to take into account the effects of other factors. For example, in Example 14.5, the effects of the environment can be added as

$$\boldsymbol{y} = \mathbf{1}_n\mu + \boldsymbol{X}_E\boldsymbol{\beta}_E + \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{14.13}$$

where $\boldsymbol{X}_E$ is the design matrix of the environments and $\boldsymbol{\beta}_E$ is the vector with the environment effects, and the Bayesian formulation can be completed by assuming a prior distribution for $\boldsymbol{\beta}_E$. As was described in Chap. 6 in the BGLR package, there are several options for this: FIXED, BRR, BayesA, BayesB, BayesC, and BL. In the next example, the first one is used.

Example 14.6

This is a continuation of Example 14.5 used to illustrate the performance when adding environmental information to the prediction task by using the Bayesian formulation (14.13). The resulting models were evaluated with 100 random partitions with both Fourier and B-spline basis and first derivative penalization.

The key code for implementing this example is given below.

For Fourier basis:

```
#Matrix design of environment
X_E = model.matrix(~0+Env,data=dat_F)[,-1]
#Linear predictor to PBFR + Env effect
ETA = list(list(X=X_E,model='FIXED'),list(X=X_Fa1,model='FIXED'),
list(X=X_Fa2,model='BRR'))
```

```
# Linear predictor to BFR + Env effect
ETA_NP = list(list(X=X_E,model='FIXED'),list(X=X_F[,1],
                      model='FIXED'),list(X=X_F[,-1],model='BRR'))
```

For B-spline basis:

```
#Matrix design of environment
X_E = model.matrix(~0+Env,data=dat_F)[,-1]
#Linear predictor to PBFR + Env effect
ETA = list(list(X=X_E,model='FIXED'),list(X=X_Fa1,model='FIXED'),
     list(X=X_Fa2,model='BRR'))
#Linear predictor to BFR + Env effect
ETA_NP = list(list(X=X_E,model='FIXED'),list(X=X_BS,model='BRR'))

#Random cross-validation
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:RP)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_F[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_F)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,
          penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)

 #BGLR
 y_NA = y
 y_NA[Pos_tst] = NA
 A_BGLR = BGLR(y_NA,ETA= ETA,nIter=1e4, burnIn = 1e3,verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR[p] = mean((y_tst-yp_tst)^2)

 A_BGLR = BGLR(y_NA,ETA= ETA_NP,nIter=1e4, burnIn = 1e3,
verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR_NP[p] = mean((y_tst-yp_tst)^2)
  cat('Partition = ', p,'\n')
 }
```

Table 14.2 Mean square error (MSE) of prediction for 100 random partitions for Fourier and B-spline representations, where the PFR and FR are classic functional regression models used in Example 14.4, and PBFR and BFR are model (14.13) with prior $\boldsymbol{\beta} \sim N\left(\mathbf{0}, \sigma^2_{\beta}\boldsymbol{P}^{-1}\right)$ (with penalization matrix based on the first derivative) and $\boldsymbol{\beta} \sim N\left(\mathbf{0}, \sigma^2_{\beta}\boldsymbol{I}_{L_1}\right)$ (without penalization), respectively, for the functional term

		PFR	FR	PBFR	BFR
Fourier	**Mean**	0.6029	0.7797	0.4580	0.4382
	SD	0.2988	0.3593	0.2058	0.1986
B-spline	**Mean**	0.6138	0.6133	0.4304	0.4176
	SD	0.2695	0.2568	0.1729	0.1668

The results are shown in Table 14.2, where the third and fourth rows correspond to the average (Mean) and standard deviation (SD) of the MSE, when using Fourier basis (L_2=29 for covariate representation and $L_1 = 21$ for the beta function $\beta(\cdot)$) in the four fitted models (PFR, FR, PBFR, and BFR, with the environment effects added in the predictor of the model). With respect to the classical functional regression models (PFR and FR), both Bayesian models with environment (PBFR and BFR) effects resulted in a better performance, but again, the Bayesian model without penalization matrix was better (0.4382 vs. 0.4580) (see **Appendix 4** for the whole code).

For the B-spline basis, similar results were obtained, but again, the difference between the PFR and FR was not so important as observed before (see **Appendix 4**). So, in both cases (Fourier and B-spline), adding the environment information to the model improved the prediction performance. In general, the extra information can be added and explored easily with the BGLR package to determine the importance of this in the prediction task of interest.

Further information can be easily added to the model without many complications. See Chap. 6 for more detailed information on how to do this with the BGLR R package.

Example 14.7

This example is an extension of Example 14.6 by adding the interaction of the environment with the hyper-spectral data (the functional covariable) in the predictor of the model. The corresponding term is given by

$$\int_0^T x(t)\beta_e(t)dt,$$

where $\beta_e(t)$ is the coefficient function corresponding to the functional part that represents the interaction between the eth environment and reflectance for wavelength t (in general, the functional covariate measured in time t), to allow the effect of reflectance to vary by environment (see Montesinos-López et al. 2017b). By assuming that there are n_e observations in environment e, $e = 1, \ldots, I$, the corresponding

re-expressed model after representing the coefficient function $\beta_e(t)$ in terms of the same basis functions used for $\beta(t)$, $\beta_e(t) = \sum_{l=1}^{L_{1e}} \beta_{el}\phi_l(t)$, is given by

$$\boldsymbol{y} = \mathbf{1}_n\mu + \boldsymbol{X}_E\boldsymbol{\beta}_E + \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{X}_{EF}\boldsymbol{\beta}_{EF} + \boldsymbol{\epsilon}, \tag{14.14}$$

where $\mathbf{1}_n$ is a vector of dimension $n \times 1$ with all its entries equal to 1, $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]^{\mathrm{T}}$ and $\boldsymbol{x}_i = [x_{i1}, \ldots, x_{iL_1}]^{\mathrm{T}}, i = 1, \ldots, n = n_1 + \cdots + n_I$, as defined in (14.4) and (14.5), $\boldsymbol{X}_E$ is the design matrix of the environments and $\boldsymbol{\beta}_E$ is the vector with the environment effects (see 14.13), $\boldsymbol{X}_{EF}$ is the design matrix of the interaction effecs of enviroment-reflectance and $\boldsymbol{\beta}_{EF}$ is the corresponding interaction effects of environment-reflectance, as given by

$$\boldsymbol{X}_{EF} = \begin{bmatrix} \boldsymbol{x}_1^{\mathrm{T}} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{x}_{n_1}^{\mathrm{T}} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{x}_{n_1+1}^{\mathrm{T}} & \vdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \boldsymbol{x}_{n_1+n_2}^{\mathrm{T}} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{x}_{n-n_I+1}^{\mathrm{T}} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{x}_n^{\mathrm{T}} \end{bmatrix} \text{ and } \boldsymbol{\beta}_{EF} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_I \end{bmatrix}.$$

This model was also evaluated with 100 random partitions with both Fourier and B-spline basis and with (PBFR (14.14)) and without (BFR (14.14)) first derivative penalization. The results are shown in Table 14.3, together with the prediction performance of the model with no interaction term (model 14.13). We can observe that by adding the interaction of environment with the functional covariate, both Bayesian models (PBFR and BFR) resulted in a reduction on average of about 35% of the MSE (PBFR (14.13) vs. PBFR (14.14) and BFR (14.13) vs. BFR (14.14)), and again the Bayesian model without penalization matrix was better (0.2955 vs. 0.2899)

Table 14.3 Mean squared error of prediction (MSE) for 100 random partitions for Fourier and B-spline representations, where PBFR (14.13) and BFR (14.13) are MSE of the model (14.13) with and without penalization matrix based on the first derivative in the functional term ($\boldsymbol{X\beta}$), and PBFR (14.14) and BFR (14.14) are for model (14.14) with and without penalization matrix based on the first derivative in the functional terms ($\boldsymbol{X\beta}$,$\boldsymbol{X}_{EF}\boldsymbol{\beta}_{EF}$)

		PBFR (14.13)	BFR (14.13)	PBFR (14.14)	BFR (14.14)
Fourier	**Mean**	0.4563	0.4387	0.2955	0.2899
	SD	0.2056	0.1992	0.1363	0.1313
B-spline	**Mean**	0.4510	0.4361	0.2814	0.2818
	SD	0.1892	0.1837	0.1149	0.1224

in the Fourier basis, while in the B-spline basis the Bayesian model with penalization matrix was better (0.2814 vs. 0.2818).

Finally, in this chapter, we gave the basic theory of functional regression and we provided examples to illustrate this methodology for genomic prediction using the *glmnet* and BGLR packages. The examples show in detail how to implement functional regression analysis in a more straightforward way by taking advantage of the existing software for genomic selection. Also, the examples are done with small data sets so that the user can run them on his/her own computer and can understand the implementation process well.

Appendix 1

```
rm(list=ls(all=TRUE))
#Example 14.3
load('dat_ls.RData')
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
#Wavelengths data
dat_W = dat_ls$dat_WL
colnames(dat_W)[1:8]
head(dat_W)[,1:8]

#Wavelengths used
Wv = as.numeric(substring(colnames(dat_W)[-(1:2)],2))
#Reflectance in each individual
X_W = unique(dat_W[,-(1:2)])
X_W = scale(X_W,scale=FALSE)
#Reflectance behavior as a function of wavelength for 10 individuals
W = matrix(Wv,nr=length(Wv),nc=60,byrow = FALSE)
matplot(W,t(X_W),xlab='Wavelengths',ylab='Reflectance',
    col=dat_W$Env,pch=1,cex=0.5)

#Optimal number of Fourier basis functions to represent the functional
covariable
#of each individual
library(fda)
Perd = diff(range(Wv))
plot(Wv,X_W[1,])
m = length(Wv)
n = dim(dat_F)[1]
nbF = seq(3,m/2,2)
BIC_mat = matrix(NA,nr=length(nbF),nc = n)
for(l in 1:length(nbF))
{
 #Fourier basis for x(t)
 Psi_F = create.fourier.basis(range(Wv)+c(0,0), nbasis = nbF[l],
```

```
                period = Perd)
 SF_mat = smooth.basisPar(argvals = Wv,
              y =t(X_W), fdobj = Psi_F,lambda=0,
              Lfdobj = 0)
 #plot(SF_mat,col=dat_F$Env,xlab='t',ylab='x(t)')
 Res_mat = t(residuals(SF_mat))
 sigmav = sqrt(rowMeans(Res_mat^2))
 ll_f<-function(Res)
 {
  sigma = sqrt(mean(Res^2))
  sum(dnorm(Res,0,sigma,log=TRUE))
 }
 llv = apply(Res_mat,1,ll_f)
 BIC_v = -2*llv+log(m)*(nbF[l]+1)
 BIC_mat[l,] = BIC_v
 cat('l=',l,'\n')
}
#Optimal nbF in each curve obtained with the BIC
nbFov = apply(BIC_mat,2,function(x)nbF[which.min(x)])
plot(nbFov,xlab='Individual',ylab='Optimal number of basis functions
choosen by BIC')

#The median value of the more often selected number of basis functions
across all curves
Tb  = table(nbFov)
nbFo = as.numeric(names(Tb))[which(Tb==max(Tb))]
nbFo
nbFo = median(nbFo)
nbFo

#Optimal number of B-spline basis functions to represent the functional
covariable
#of each individual
nbBS = seq(4,m/2,1)
BIC_mat = matrix(NA,nr=length(nbBS),nc = n)
for(l in 1:length(nbBS))
{
 #Fourier basis for x(t)
 Psi_BS = create.bspline.basis(range(Wv)+c(0,0), nbasis = nbBS[l],
norder = 4)
 SBS_mat = smooth.basisPar(argvals = Wv,
              y =t(X_W), fdobj = Psi_BS,lambda=0,
              Lfdobj = 0)
 #plot(SBS_mat,col=dat_F$Env,xlab='t',ylab='x(t)')
 Res_mat = t(residuals(SBS_mat))
 sigmav = sqrt(rowMeans(Res_mat^2))
 ll_f<-function(Res)
 {
  sigma = sqrt(mean(Res^2))
  sum(dnorm(Res,0,sigma,log=TRUE))
 }
 llv = apply(Res_mat,1,ll_f)
 #BIC_v = -2*llv+log(m)*(nbBS[l]-4+nbBS[l]+1)
```

```
 BIC_v = -2*llv+log(m)*(nbBS[l]+1)
 BIC_mat[l,] = BIC_v

 cat('l=',l,'\n')
}
#Optimal nbBS for each curve obtained with the BIC
nbBSov = apply(BIC_mat,2,function(x)nbBS[which.min(x)])
plot(nbBSov,xlab='Individual',ylab='Optimal number of basis
functions choosen by BIC')

Tb = table(nbBSov)
barplot(Tb)
nbBSov = as.numeric(names(Tb))[which(Tb==max(Tb))]
nbBSo = median(nbBSov)
nbBSo

#Fourier and B-spline representations with the "optimal" number
#of basis functions obtained across the curves using the BIC
par(mfrow=c(1,2))
#Fourier representation of all curves using 29 Fourier basis functions
Psi_F = create.fourier.basis(range(Wv), nbasis = nbFo,
             period = Perd)
SF_mat = smooth.basisPar(argvals = Wv,
            y =t(X_W), fdobj = Psi_F,lambda=0,
            Lfdobj = 0)
matplot(Wv,fitted(SF_mat),col=dat_F$Env,xlab='t (Wavelengths)',
   ylab='x(t) (Reflectance)',pch=1,type='l',
   main=paste('Fourier representation with\n ',nbFo,'basis
functions',sep=' '))

#B-spline representation of all curves using 16 B-spline basis
functions
Psi_BS = create.bspline.basis(range(Wv), nbasis = nbBSo)
SBS_mat = smooth.basisPar(argvals = Wv,
            y =t(X_W), fdobj = Psi_BS,lambda=0,
            Lfdobj = 0)
matplot(Wv,fitted(SBS_mat),col=dat_F$Env,xlab='t (Wavelengths)',
   ylab='x(t) (Reflectance)',pch=1,type='l',
   main=paste('B-spline representation with\n ',nbBSo,
            'basis functions'',sep=' '))

#Functional regression with Fourier and B-splines
#Computing X for Fourier representation with L1 = 21 Fourier basis
#functions for the beta function
L1 = 21
Phi_F = create.fourier.basis(range(Wv), nbasis = L1,
              period = Perd)
Q   = inprod(Phi_F,Psi_F)
c_mat = t(SF_mat$fd$coefs)
X_F  = c_mat%*%t(Q)
dim(X_F)
```

```
X_F_df = data.frame(dat_W[,1:2],X_F)
#write.csv(X_F_df,file='X_F_df.csv')

#Computing X for B-spline representation with L1 = 21 B-spline basis
functions
#for the beta function
L1 = 21
Phi_BS = create.bspline.basis(range(Wv), nbasis = L1,
                 norder= 4)
Q   = inprod(Phi_BS,Psi_BS)
c_mat = t(SBS_mat$fd$coefs)
X_BS  = c_mat%*%t(Q)
dim(X_BS)
X_BS_df = data.frame(dat_W[,1:2],X_BS)
#write.csv(X_BS_df,file='X_BS_df.csv')

#OLS estimation using Fourier representation with L1 = 21 Fourier basis
#functions for the beta function
par(mfrow=c(1,2))
yv = dat_F$y
A_F = lm(yv~X_F)
betav = coef(A_F)
#Plot of the 21 estimated Bj's
#par(mar=c(5,5.1,2,2))
#plot(betav[-1],xlab=expression(j),ylab=expression(hat(beta)[j]))
#betaf_F = eval.basis(Wv,Phi_F)%*%betav[-1]
#plot(Wv,betaf_F)
#Fitted values
yp = fitted(A_F)
plot(yv,yp,xlab='y (Observed)',ylab='yp (Predicted)',
   main='Fourier basis functions \n 29 for x(t) and 21 for B(t)')
abline(a=0,b=1)
mean((yv-yp)^2)

#OLS estimation using Fourier representation with L1 = 21 B-spline basis
#functions for the beta function
yv = dat_F$y
A_BS = lm(yv~X_BS)
betav = coef(A_BS)
#Plot of the 21 estimated Bj's
#par(mar=c(5,5.1,2,2))
#plot(betav[-1],xlab=expression(j),ylab=expression(hat(beta)[j]))
#betaf_BS = eval.basis(Wv,Phi_BS)%*%betav[-1]
#plot(Wv,betaf_F)
#Fitted values
yp = fitted(A_BS)
plot(yv,yp,xlab='y (Observed)',ylab='yp (Predicted)',
   main='B-spline basis functions \n 16 for x(t) and 21 for B(t)')
abline(a=0,b=1)
mean((yv-yp)^2)
#BIC to choose the optimal number of basis functions for the beta
#function (L1) in the Fourier representation of this
nbF_B = seq(3,29,2)
```

```
Tab = data.frame()
for(i in 1:length(nbF_B))
{
 Phi_F = create.fourier.basis(range(Wv), nbasis = nbF_B[i],
                  period = Perd)
 Q = inprod(Phi_F,Psi_F)
 c_mat = t(SF_mat$fd$coefs)
 X_F = c_mat%*%t(Q)
 A = lm(yv~X_F)
 BIC = BIC(A)
 Tab = rbind(Tab,data.frame(nb = nbF_B[i],BIC=BIC))
}
plot(Tab$nb,Tab$BIC,xlab='Número de bases',ylab='BIC')
nbFo_B = Tab$nb[which.min(Tab$BIC)]
nbFo_B

#Observed and fitted values with "optimal" number of basis functions for
#the beta function (L1) with the Fourier representation
L1 = nbFo_B
Phi_F = create.fourier.basis(range(Wv), nbasis = L1,
                period = Perd)
Q   = inprod(Phi_F,Psi_F)
c_mat = t(SF_mat$fd$coefs)
X_F  = c_mat%*%t(Q)
dim(X_F)
A_F = lm(yv~X_F)
betav = coef(A_F)
betaf_F = eval.basis(Wv,Phi_F)%*%betav[-1]
plot(Wv,betaf_F)

#Fitted values
yp = fitted(A_F)
plot(yv,yp,xlab='y (Observed)',ylab='yp (Predicted)')
abline(a=0,b=1)
mean((yv-yp)^2)

#Q-Q Normal
ev = residuals(A_F)
sigma2 = mean(ev^2)
qqnorm(ev/sqrt(sigma2))
abline(a=0,b=1)

#BIC to choose the optimal number of basis functions for the beta
#function (L1) in the B-spline representation of this
nbBS_B = seq(5,45,1)
Tab = data.frame()
for(i in 1:length(nbBS_B))
{
 Phi_BS = create.bspline.basis(range(Wv), nbasis = nbBS_B[i],
                  norder = 4)
 Q = inprod(Phi_BS,Psi_BS)
 c_mat = t(SBS_mat$fd$coefs)
 X_BS = c_mat%*%t(Q)
```

```
 A = lm(yv~X_BS)
 BIC = BIC(A)
 Tab = rbind(Tab,data.frame(nb = nbBS_B[i],BIC=BIC))
}
plot(Tab$nb,Tab$BIC,xlab='Number of basis functions',ylab='BIC')
nbBSo_B = Tab$nb[which.min(Tab$BIC)]
nbBSo_B

#Observed and fitted values with "optimal" number of basis functions for
#the beta function (L1) with the B-spline representation
L1 = nbBSo_B
Phi_BS = create.bspline.basis(range(Wv), nbasis = L1, norder = 4)
Q   = inprod(Phi_BS,Psi_BS)
c_mat = t(SBS_mat$fd$coefs)
X_BS  = c_mat%*%t(Q)
dim(X_BS)
A_BS = lm(yv~X_BS)
betav = coef(A_BS)
betaf_BS = eval.basis(Wv,Phi_BS)%*%betav[-1]
plot(Wv,betaf_BS)
#lines(Wv,betaf_F,col=2)

par(mfrow=c(2,2))
#Fourier
#Fitted values
yp = fitted(A_F)
plot(yv,yp,xlab='y (Observed)',ylab='yp (Predicted)')
abline(a=0,b=1)
mean((yv-yp)^2)

#Q-Q Normal
ev = residuals(A_F)
sigma2 = mean(ev^2)
qqnorm(ev/sqrt(sigma2))
abline(a=0,b=1)
#B-spline
#Fitted values
yp = fitted(A_BS)
#par(mfrow=c(1,2),oma = c(0, 0, 2, 0))
plot(yv,yp,xlab='y (Observed)',ylab='yp (Predicted)')#,main='B-
spline representation')
abline(a=0,b=1)
mean((yv-yp)^2)
#Q-Q Normal
ev = residuals(A_BS)
sigma2 = mean(ev^2)
qqnorm(ev/sqrt(sigma2))
abline(a=0,b=1)
#mtext("Title for Two Plots", outer = T, cex = 1.5)
mean((yv-yp)^2)
```

#BIC of the models corresponding to the optimal Fourier and B-spline #representations

```
BIC(A_F)
BIC(A_BS)

#Random partition to measure the prediction performance
#of Fourier and B-spline representations
set.seed(1)
MSEP = data.frame()
for(p in 1:10)
{
 Pos_j = sample(n,.20*n)
 dat_F_j = dat_F[-Pos_j,]
 X_W_j  = X_W[-Pos_j,]
 #Fourier
 A_F = lm(yv[-Pos_j]~X_F[-Pos_j,])
 #Fitted values
 #yp = predict(A_F,newdata = data.frame(X_F[Pos_j,]))
 yp_F = cbind(1,X_F[Pos_j,])%*%coef(A_F)
 #B-spline
 A_BS = lm(yv[-Pos_j]~X_BS[-Pos_j,])
 yp_BS = cbind(1,X_BS[Pos_j,])%*%coef(A_BS)

 MSEP = rbind(MSEP,data.frame(MSEP_Fourier = mean((yv[Pos_j]-yp_F)
^2),
                MSEP_BS = mean((yv[Pos_j]-yp_BS)^2)))
}
MSEP
```

Appendix 2 (Example 14.4)

```
rm(list=ls(all=TRUE))
library(fda)
#Example 14.4 (data set of Example 14.3)
load('dat_ls.RData')
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
yv = dat_F$y
n = length(yv)

#Wavelengths data
dat_W = dat_ls$dat_WL
colnames(dat_W)[1:8]
head(dat_W)[,1:8]

#Wavelengths used
Wv = as.numeric(substring(colnames(dat_W)[-(1:2)],2))
#Reflectance in each individual
X_W = unique(dat_W[,-(1:2)])
```

```
X_W = scale(X_W,scale=FALSE)
#Reflectance behavior as a function of wavelength for 10 individuals
W = matrix(Wv,nr=length(Wv),nc=60,byrow = FALSE)
matplot(W,t(X_W),xlab='Wavelengths',ylab='Reflectance',
    col=dat_W$Env,pch=1,cex=0.5)

#Number of Fourier and B-spline basis functions to represent the
covariable
#function
nbFo = 29
nbBSo = 16

#Matrix design X computed with Fourier and B-spline bases
#Computing X for Fourier representation with L1 = 21 Fourier basis
functions
#for the beta function
L1 = 21
P = diff(range(Wv))
Psi_F = create.fourier.basis(range(Wv), nbasis = nbFo,
              period = P)
SF_mat = smooth.basisPar(argvals = Wv,
            y =t(X_W), fdobj = Psi_F,lambda=0,
            Lfdobj = 0)
Phi_F = create.fourier.basis(range(Wv), nbasis = L1,
              period = P)
Q   = inprod(Phi_F,Psi_F)
c_mat = t(SF_mat$fd$coefs)
X_F  = c_mat%*%t(Q)
dim(X_F)

#Computing X for B-spline representation with L1 = 21 B-spline basis
functions
#for the beta function
L1 = 21
Psi_BS = create.bspline.basis(range(Wv), nbasis = nbBSo)
SBS_mat = smooth.basisPar(argvals = Wv,
             y =t(X_W), fdobj = Psi_BS,lambda=0,
             Lfdobj = 0)
Phi_BS = create.bspline.basis(range(Wv), nbasis = L1,
               norder= 4)
Q   = inprod(Phi_BS,Psi_BS)
c_mat = t(SBS_mat$fd$coefs)
X_BS  = c_mat%*%t(Q)
dim(X_BS)

#Smoothing estimation using Fourier representation for the functional
#covariable and L1 = 21 Fourier basis functions for the beta function
par(mfrow=c(1,2))
library(glmnet)
#Penalization with first derivate (Changing the value Lfdobj=1 to
Lfdobj=2, the penalty matrix #with second derivates is obtained)
P_mat_F = eval.penalty(Phi_F,Lfdobj=1)
ei_P = eigen(P_mat_F)
```

```
gamma = ei_P$vectors
dv = ei_P$values
dv = ifelse(dv<1e-10,0,dv)

X_Fa = X_F%*%gamma

#Grid of lambda values obtained by varying the proportion of variance
explained
#by the functional predictor
lamb_FR_f<-function(Xa,dv,K=100,li=1e-1,ls=1-1e-12)
{
 Pos = which(dv<1e-10)
 D_inv = diag(1/dv[-Pos])
 PEV = seq(li,ls,length=K)
 Xa = Xa[,-Pos]
 lambv = (1-PEV)/PEV*mean(diag(Xa%*%D_inv%*%t(Xa)))
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}

lambda = lamb_FR_f(X_Fa,dv,K=1e2)

#Random cross-validation
y=yv
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:RP)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_F[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_F)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 #plot(A_PFR)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)
 cat('Partition = ', p,'\n')

}
MSEP_df
```

```
#Mean and SD of MSEP across 100 RP
apply(MSEP_df[,-1],2,function(x)c(mean(x),sd(x)))
mean(MSEP_df$MSEP_PFR<MSEP_df$MSEP_FR)

plot(MSEP_df$MSEP_PFR,MSEP_df$MSEP_FR,
   col = ifelse(MSEP_df$MSEP_PFR<MSEP_df$MSEP_FR,
         3,1),xlab='MSEP PFR',ylab='MSEP FR',
   main='Fourier')
abline(a=0, b=1)

#Smoothing estimation using B-spline for the functional
#covariable and L1 = 21 B-spline basis functions for the beta function
library(glmnet)
#Penalization with first derivate (Changing the value Lfdobj=1 to
Lfdobj=2, the penalty #matrix with second derivates is obtained)
P_mat_F = eval.penalty(Phi_BS,Lfdobj=1)
ei_P = eigen(P_mat_F)
gamma = ei_P$vectors
dv = ei_P$values
dv = ifelse(dv<1e-10,0,dv)
X_Fa = X_BS%*%gamma

#Grid of lambda values obtained by varying the proportion of variance
explained
#by the functional predictor
lamb_FR_f<-function(Xa,dv,K=100,li=1e-1,ls=1-1e-12)
{
 Pos = which(dv<1e-10)
 D_inv = diag(1/dv[-Pos])
 PEV = seq(li,ls,length=K)
 Xa = Xa[,-Pos]
 lambv = (1-PEV)/PEV*mean(diag(Xa%*%D_inv%*%t(Xa)))
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}

lambda = lamb_FR_f(X_Fa,lambv,K=1e2)

#Random cross-validation
y=yv
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:RP)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_BS[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_BS)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
```

```
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 #plot(A_PFR)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)
 cat('Partition = ', p,'\n')

}
MSEP_df

#Mean and SD of MSEP across 100 RP
apply(MSEP_df[,-1],2,function(x)c(mean(x),sd(x)))
mean(MSEP_df$MSEP_PFR<MSEP_df$MSEP_FR)

plot(MSEP_df$MSEP_PFR,MSEP_df$MSEP_FR,
   col = ifelse(MSEP_df$MSEP_PFR<MSEP_df$MSEP_FR,
         3,1),xlab='MSEP PFR',ylab='MSEP FR',
   main='B-spline')
abline(a=0,b=1)
```

Appendix 3 (Example 14.5)

```
rm(list=ls(all=TRUE))
library(fda)
#Example 14.5
load('dat_ls.RData')
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
yv = dat_F$y
n = length(yv)

#Wavelengths data
dat_W = dat_ls$dat_WL
colnames(dat_W)[1:8]
head(dat_W)[,1:8]

#Wavelengths used
Wv = as.numeric(substring(colnames(dat_W)[-(1:2)],2))
#Reflectance in each individual
X_W = unique(dat_W[,-(1:2)])
X_W = scale(X_W,scale=FALSE)
```

```
#Number of Fourier and B-spline basis functions to represent the
covariable
#function
nbFo = 29
nbBSo = 16

#Functional regression with Fourier and B-splines
#Computing X for Fourier representation with L1 = 21 Fourier basis
functions
#for the beta function
L1 = 21
P = diff(range(Wv))
Psi_F = create.fourier.basis(range(Wv), nbasis = nbFo,
              period = P)
SF_mat = smooth.basisPar(argvals = Wv,
            y =t(X_W), fdobj = Psi_F,lambda=0,
            Lfdobj = 0)
Phi_F = create.fourier.basis(range(Wv), nbasis = L1,
              period = P)
Q   = inprod(Phi_F,Psi_F)
c_mat = t(SF_mat$fd$coefs)
X_F  = c_mat%*%t(Q)
dim(X_F)

#Computing X for B-spline representation with L1 = 21 B-spline basis
functions
#for the beta function
L1 = 21
Psi_BS = create.bspline.basis(range(Wv), nbasis = nbBSo)
SBS_mat = smooth.basisPar(argvals = Wv,
            y =t(X_W), fdobj = Psi_BS,lambda=0,
            Lfdobj = 0)
Phi_BS = create.bspline.basis(range(Wv), nbasis = L1,
               norder= 4)
Q   = inprod(Phi_BS,Psi_BS)
c_mat = t(SBS_mat$fd$coefs)
X_BS  = c_mat%*%t(Q)
dim(X_BS)

#Smoothing estimation using Fourier representation for the functional
#covariable and L1 = 21 Fourier basis functions for the beta function
par(mfrow=c(1,2))
library(glmnet)
#Penalization with first derivate
P_mat_F = eval.penalty(Phi_F,Lfdobj=1)
ei_P  = eigen(P_mat_F)
gamma = ei_P$vectors
dv = ei_P$values
dv = ifelse(dv<1e-10,0,dv)

X_Fa =  X_F%*%gamma
```

```
#Grid of lambda values obtained by varying the proportion of variance
explained
#by the functional predictor
lamb_FR_f<-function(Xa,dv,K=100,li=1e-1,ls=1-1e-12)
{
 Pos = which(dv<1e-10)
 D_inv = diag(1/dv[-Pos])
 PEV = seq(li,ls,length=K)
 Xa = Xa[,-Pos]
 lambv = (1-PEV)/PEV*mean(diag(Xa%*%D_inv%*%t(Xa)))
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}
lambda = lamb_FR_f(X_Fa,dv,K=1e2)

y=yv
library(BGLR)
#Linear predictor specification in BGLR for the PBFR model
Pos = which(dv>0)
#Matrix design for the non-penalized columns of X^a
X_Fa1 = X_Fa[,-Pos]
#Matrix design for the penalized columns of X^a
X_Fa2 = X_Fa[,Pos]%*%diag(1/sqrt(dv[Pos]))
ETA = list(list(X=X_Fa1,model='FIXED'),list(X=X_Fa2,model='BRR'))

#Linear predictor specification in BGLR for the BFR model
ETA_NP = list(list(X=X_F[,1],model='FIXED'),list(X=X_F[,-1],
model='BRR'))

#Random cross-validation
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:RP)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_F[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_F)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,
          penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
```

```
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)

 #BGLR
 y_NA = y
 y_NA[Pos_tst] = NA
 A_BGLR = BGLR(y_NA,ETA= ETA,nIter=1e4, burnIn = 1e3,verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR[p] = mean((y_tst-yp_tst)^2)

 A_BGLR = BGLR(y_NA,ETA= ETA_NP,nIter=1e4, burnIn = 1e3,
verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR_NP[p] = mean((y_tst-yp_tst)^2)

 cat('Partition = ', p,'\n')

}
MSEP_df

#Mean and SD of MSEP across 10 RP
Tab = apply(MSEP_df[,-1],2,function(x)c(mean(x),sd(x)))
Tab

#Smoothing estimation using B-spline representation for the functional
#covariable and L1 = 21 B-spline basis functions for the beta function
library(glmnet)
#Penalization with first derivate
P_mat_F = eval.penalty(Phi_BS,Lfdobj=1)
ei_P = eigen(P_mat_F)
gamma = ei_P$vectors
dv = ei_P$values
dv = ifelse(dv<1e-10,0,dv)
X_Fa = X_BS%*%gamma

#Grid of lambda values obtained by varying the proportion of variance
explained
#by the functional predictor
lamb_FR_f<-function(Xa,dv,K=100,li=1e-1,ls=1-1e-12)
{
 Pos = which(dv<1e-10)
 D_inv = diag(1/dv[-Pos])
 PEV = seq(li,ls,length=K)
 Xa = Xa[,-Pos]
 lambv = (1-PEV)/PEV*mean(diag(Xa%*%D_inv%*%t(Xa)))
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}

lambda = lamb_FR_f(X_Fa,dv,K=1e2)
```

```
#Random cross-validation
y=yv
library(BGLR)
Pos = which(dv>0)
X_Fa1 = X_Fa[,-Pos]
X_Fa2 = X_Fa[,Pos]%*%diag(1/sqrt(dv[Pos]))
ETA = list(list(X=X_Fa1,model='FIXED'),list(X=X_Fa2,model='BRR'))
ETA_NP = list(list(X=X_BS,model='BRR'))
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:100)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_BS[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_BS)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 #plot(A_PFR)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)

 #BGLR
 y_NA = y
 y_NA[Pos_tst] = NA
 A_BGLR = BGLR(y_NA,ETA= ETA,nIter=1e4, burnIn = 1e3,verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR[p] = mean((y_tst-yp_tst)^2)

 A_BGLR = BGLR(y_NA,ETA= ETA_NP,nIter=1e4, burnIn = 1e3,
verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR_NP[p] = mean((y_tst-yp_tst)^2)

 cat('Partition = ', p,'\n')

}
MSEP_df

#Mean and SD of MSEP across 100 RP
Tab = apply(MSEP_df[,-1],2,function(x)c(mean(x),sd(x)))
Tab
```

Appendix 4 (Example 14.6)

It is the same as the R code used in Example 14.5 but now to the corresponding predictor need to be added the matrix design of environments:

```
rm(list=ls(all=TRUE))
library(fda)
#Example 14.6
load('dat_ls.RData')
#Phenotypic data
dat_F = dat_ls$dat_F
head(dat_F)
yv = dat_F$y
n = length(yv)

#Wavelengths data
dat_W = dat_ls$dat_WL
colnames(dat_W)[1:8]
head(dat_W)[,1:8]

#Wavelengths used
Wv = as.numeric(substring(colnames(dat_W)[-(1:2)],2))
#Reflectance in each individual
X_W = unique(dat_W[,-(1:2)])
X_W = scale(X_W,scale=FALSE)
#Reflectance behavior as a function of wavelength for 10 individuals
W = matrix(Wv,nr=length(Wv),nc=60,byrow = FALSE)
matplot(W,t(X_W),xlab='Wavelengths',ylab='Reflectance',
    col=dat_W$Env,pch=1,cex=0.5)

#Number of Fourier and B-spline basis functions to represent the
covariable
#function
nbFo = 29
nbBSo = 16

#----
#Functional regression with Fourier and B-splines
#---
#Computing X for Fourier representation with L1 = 21 Fourier basis
functions
#for the beta function
L1 = 21
P = diff(range(Wv))
Psi_F = create.fourier.basis(range(Wv), nbasis = nbFo,
              period = P)
SF_mat = smooth.basisPar(argvals = Wv,
            y =t(X_W), fdobj = Psi_F,lambda=0,
            Lfdobj = 0)
Phi_F = create.fourier.basis(range(Wv), nbasis = L1,
              period = P)
```

```
Q   = inprod(Phi_F,Psi_F)
c_mat = t(SF_mat$fd$coefs)
X_F  = c_mat%*%t(Q)
dim(X_F)

#Computing X for B-spline representation with L1 = 21 B-spline basis
functions
#for the beta function
L1 = 21
Psi_BS = create.bspline.basis(range(Wv), nbasis = nbBSo)
SBS_mat = smooth.basisPar(argvals = Wv,
             y =t(X_W), fdobj = Psi_BS,lambda=0,
             Lfdobj = 0)
Phi_BS = create.bspline.basis(range(Wv), nbasis = L1,
               norder= 4)
Q   = inprod(Phi_BS,Psi_BS)
c_mat = t(SBS_mat$fd$coefs)
X_BS  = c_mat%*%t(Q)
dim(X_BS)

#----
#Smoothing estimation using Fourier representation for the functional
#covariable and L1 =  21 Fourier basis functions for the beta function
#----
par(mfrow=c(1,2))
library(glmnet)
#Penalization with first derivate
P_mat_F = eval.penalty(Phi_F,Lfdobj=1)
ei_P  =  eigen(P_mat_F)
gamma = ei_P$vectors
dv = ei_P$values
dv = ifelse(dv<1e-10,0,dv)

X_Fa =  X_F%*%gamma

#Grid of lambda values obtained by varying the proportion of variance
explained
#by the functional predictor
lamb_FR_f<-function(Xa,dv,K=100,li=1e-1,ls=1-1e-12)
{
 Pos =  which(dv<1e-10)
 D_inv = diag(1/dv[-Pos])
 PEV = seq(li,ls,length=K)
 Xa  = Xa[,-Pos]
 lambv = (1-PEV)/PEV*mean(diag(Xa%*%D_inv%*%t(Xa)))
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}

lambda = lamb_FR_f(X_Fa,dv,K=1e2)
```

```
y=yv
library(BGLR)
#Linear predictor specification in BGLR for the PBFR model
Pos = which(dv>0)
#Matrix design for the non-penalized columns of X^a
X_Fa1 = X_Fa[,-Pos]
#Matrix design for the penalized columns of X^a
X_Fa2 = X_Fa[,Pos]%*%diag(1/sqrt(dv[Pos]))
#Matrix design of environment
X_E = model.matrix(~0+Env,data=dat_F)[,-1]
#Linear predictor to PBFR + Env effect
ETA = list(list(X=X_E,model='FIXED'),list(X=X_Fa1,model='FIXED'),
list(X=X_Fa2,model='BRR'))

# Linear predictor to BFR + Env effect
ETA_NP = list(list(X=X_E,model='FIXED'),list(X=X_F[,1],
                      model='FIXED'),list(X=X_F[,-1],model='BRR'))

#Random cross-validation
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:RP)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_F[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_F)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,
          penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)

 #BGLR
 y_NA = y
 y_NA[Pos_tst] = NA
 A_BGLR = BGLR(y_NA,ETA= ETA,nIter=1e4, burnIn = 1e3,verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR[p] = mean((y_tst-yp_tst)^2)

 A_BGLR = BGLR(y_NA,ETA= ETA_NP,nIter=1e4, burnIn = 1e3,
verbose=FALSE)
```

```
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR_NP[p] = mean((y_tst-yp_tst)^2)

 cat('Partition = ', p,'\n')

}
MSEP_df

#Mean and SD of MSEP across 10 RP
Tab = apply(MSEP_df[,-1],2,function(x)c(mean(x),sd(x)))
Tab

#---
#Smoothing estimation using B-spline representation for the functional
#covariable and L1 = 21 B-spline basis functions for the beta function
#---
library(glmnet)
#Penalization with first derivate
P_mat_F = eval.penalty(Phi_BS,Lfdobj=1)
ei_P = eigen(P_mat_F)
gamma = ei_P$vectors
dv = ei_P$values
dv = ifelse(dv<1e-10,0,dv)
X_Fa = X_BS%*%gamma

#Grid of lambda values obtained by varying the proportion of variance
explained
#by the functional predictor
lamb_FR_f<-function(Xa,dv,K=100,li=1e-1,ls=1-1e-12)
{
 Pos = which(dv<1e-10)
 D_inv = diag(1/dv[-Pos])
 PEV = seq(li,ls,length=K)
 Xa = Xa[,-Pos]
 lambv = (1-PEV)/PEV*mean(diag(Xa%*%D_inv%*%t(Xa)))
 lambv = exp(seq(min(log(lambv)),max(log(lambv)),length=K))
 sort(lambv,decreasing = TRUE)
}

lambda = lamb_FR_f(X_Fa,dv,K=1e2)

#Random cross-validation
y=yv
library(BGLR)
Pos = which(dv>0)
X_Fa1 = X_Fa[,-Pos]
X_Fa2 = X_Fa[,Pos]%*%diag(1/sqrt(dv[Pos]))
#Matrix design of environment
X_E = model.matrix(~0+Env,data=dat_F)[,-1]
#Linear predictor to PBFR + Env effect
ETA = list(list(X=X_E,model='FIXED'),list(X=X_Fa1,model='FIXED'),
      list(X=X_Fa2,model='BRR'))
```

```
#Linear predictor to BFR + Env effect
ETA_NP = list(list(X=X_E,model='FIXED'),list(X=X_BS,model='BRR'))
RP = 100
set.seed(1)
MSEP_df = data.frame(RP=1:RP,MSEP_PFR=NA,MSEP_FR=NA,lamb_o=NA)
for(p in 1:100)
{
 Pos_tst = sample(n,0.20*n)
 X_F_tr = X_BS[-Pos_tst,]; n_tr = dim(X_F_tr)[1]
 y_tr  = y[-Pos_tst]; y_tst = y[Pos_tst]

 #FR
 dat_df = data.frame(y=y,X=X_BS)
 A_F = lm(y~.,data=dat_df[-Pos_tst,])
 yp_tst = predict(A_F,newdata = dat_df[Pos_tst,])
 MSEP_df$MSEP_FR[p] = mean((y_tst-yp_tst)^2)

 #PFR with first derivative
 A_PFR = cv.glmnet(x =X_Fa[-Pos_tst,],y = y[-Pos_tst] ,alpha = 0,
          nfolds=5,lambda=lambda/n_tr,penalty.factor=dv,
          standardize=FALSE,maxit=1e6)
 #plot(A_PFR)
 MSEP_df$lamb_o[p] = A_PFR$lambda.min
 yp_tst = predict(A_PFR,newx=X_Fa[Pos_tst,], s="lambda.min")[,1]
 MSEP_df$MSEP_PFR[p] = mean((y_tst-yp_tst)^2)

 #BGLR
 y_NA = y
 y_NA[Pos_tst] = NA
 A_BGLR = BGLR(y_NA,ETA= ETA,nIter=1e4, burnIn = 1e3,verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR[p] = mean((y_tst-yp_tst)^2)

 A_BGLR = BGLR(y_NA,ETA= ETA_NP,nIter=1e4, burnIn = 1e3,
verbose=FALSE)
 yp_tst = A_BGLR$yHat[Pos_tst]
 MSEP_df$MSEP_BGLR_NP[p] = mean((y_tst-yp_tst)^2)

 cat('Partition = ', p,'\n')

}
MSEP_df

#Mean and SD of MSEP across 100 RP
Tab = apply(MSEP_df[,-1],2,function(x)c(mean(x),sd(x)))
Tab
```

References

Cardot H, Sarda P (2006) Linear regression models for functional data. In: Sperlich S, Härdle W, Aydınlı G (eds) The art of semiparametrics. Contributions to statistics. Physica-Verlag HD

Górecki T, Krzyśko M, Waszak Ł, Wołyński W (2018) Selected statistical methods of data analysis for multivariate functional data. Stat Pap 59(1):153–182

Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference, and prediction. Springer, USA

Montesinos-López OA, Montesinos-López A, Crossa J, de Los Campos G, Alvarado G, Mondal S, Rutkoski J, González-Pérez L, Burgueño J (2017a) Predicting grain yield using canopy hyperspectral reflectance in wheat breeding data. Plant Methods 13(4). https://doi.org/10.1186/s13007-016-0154-2

Montesinos-López A, Montesinos-López OA, Cuevas J, Mata-López WA, Burgueño J, Mondal S, Huerta J, Singh R, Autrique E, González-Pérez L, Crossa J (2017b) Genomic Bayesian functional regression models with interactions for predicting wheat grain yield using hyperspectral image data. Plant Methods 13(62). https://doi.org/10.1186/s13007-017-0212-4

Perperoglou A, Sauerbrei W, Abrahamowicz M, Schmid M (2019) A review of spline function procedures in R. BMC Med Res Methodol 19(46):1–16

Quarteroni A, Sacco R, Saleri F (2000) Numerical mathematics. Springer

Ramsay J, Hooker G, Graves S (2009) Functional data analysis with R and MATLAB. Springer

Ruppert D, Wand MP, Carroll RJ (2003) Semiparametric regression. Cambridge University Press, Cambridge

Shizgal BD, Jung JH (2003) Towards the resolution of the Gibbs phenomena. J Comput Appl Math 161(1):41–65

Usset J, Staicu AM, Maity A (2016) Interaction models for functional regression. Comput Stat Data Anal 94:317–329. https://doi.org/10.1016/j.csda.2015.08.020