Utilities for Statistical Computing in Functional Data Analysis: The R Package **fda.usc**

Manuel Oviedo de la Fuente

Máster en Técnicas Estadísticas

Universidad de Santiago de Compostela

1 de julio de 2011

El presente documento recoge bajo el título *"Utilities for Statistical Computing in Functional Data Analysis: The R Package* ***fda.usc****"*, el trabajo realizado por D. Manuel Oviedo de la Fuente como Proyecto Fin de Máster del Máster interuniversitario en Técnicas Estadísticas bajo la dirección de D. Manuel Febrero Bande, Catedrático universitario del departamento de Estadística e Investigación Operativa de la Universidad de Santiago de Compostela.

Santiago de Compostela, 1 de julio de 2011

Fdo.: D. Manuel Oviedo de la Fuente

Fdo.: D. Manuel Febrero Bande

Abstract

This document presents the **R** package **fda.usc** which implements some utilities for treatment of functional data analysis. The **fda.usc** package has been uploaded to CRAN repository of **R** since January 10, 2011 and has been updated four times. This package carries out exploratory and descriptive analysis of functional data analyzing its most important features such as depth measurements or functional outliers detection, among others. The **R** package **fda.usc** also includes functions to compute functional regression models, that helps to explain and model the relationship between a scalar response and functional explanatory data via nonparametric functional regression, basis representation, functional principal components analysis and partial least squared. There are natural extensions such as functional linear models and semi-functional partial linear models, which allow non-functional covariates and factors and make predictions. The functions of this package complement and integrate the two main references of functional data analysis: the **R** package **fda** and the functions implemented by [Ferraty and Vieu (2006)].

Contents

Chapter 1

Introduction

In recent years statistical computing applied to various fields has led to important technological change. This technological change is to incorporate measurement equipment quicker and more accurate that provides more reliable and faster measurements. This technological evolution change or replace some of the paradigms in which it is based classical statistics, for example, the paradigm that in a provided dataset the number of data is greater than the number of variables. In many areas have begun working with large databases, which increasingly often these observations of a random variable taken over a continuous interval (or in increasingly larger discretizations of the continuous interval).

For example, in fields such as spectroscopy, the measurement result is a curve that, at least, has been evaluated in a hundred of points. This type of data, which we call functional data arise naturally in many disciplines. In economics, we could talk about curves intra-day stock quotes, in engineering, we could talk about electricity demand curves, in environment, provides continuous measurements of atmospheric monitoring networks and is also well known to the rise of image recognition or spatio-temporal information.

Undoubtedly, package **fda** ([Ramsay *et al.* (2010)]) is a basic reference to work in **R** programming environment ([**R** Development Core Team(2011)]) with functional data. The books, [Ramsay and Silverman (2005)] and [Ramsay and Silverman (2002)] have helped to popularize the statistical techniques for functional data. In both cases all the techniques included are restricted to the space of $\mathcal{L}_2$ functions (the Hilbert space of all square integrable functions over a certain interval). The other milestone is the book of [Ferraty and Vieu (2006)], where functional data are processed from a nonparametric point of view that establishes frameworks for treatment. In this case, the approach to consider normed or semi-normed functional spaces in some cases may be more appropriate to describe the reality. These authors are part of the French group STAPH maintaining the page http://www.lsp.ups-tlse.fr/staph/ where you can obtain **R** code of the functions. Other functional data packages that can be useful for representing functional data are: The package **rainbow** ([Shang and Hyndman (2010)]) for functional data display and outlier detection, the package **fds** ([Hyndman and Shang (2010a)]) with functional data sets and the package **ftsa** ([Hyndman and Shang (2010b)]) for functional time series analysis. Recently the package **refund** allows computing functional penalized regression of scalar responses on functional predictors and functional responses on scalar predictors. In **Matlab** generalized functional linear models (GFLM) are included in **PACE** package (see http://anson.ucdavis.edu/~ntyang/PACE/ for Functional Data Analysis and Empirical Dynamics. Finally, from a Bayesian perspective, [Crainiceanu and Goldsmith (2010)] have developed tools for GFLM using **Winbugs**.

The objective of the package **fda.usc** is to provide a framework in functional data analysis broader than the previous one. Therefore, we propose an integration of the work on nonparametric functional methods implemented by [Ferraty and Vieu (2006)] and implemented by the group of the University of Santiago de Compostela (USC), complementing and extending some of the functions of package **fda**.

We begin in Chapter 2 with the definition of functional data and a new **R** class for functional data: *"fdata"* with a examples of how displaying the functional data in **R**. Chapter 3 describes the functional representation of an object of class *"fdata"* by basis representation 3.1 or kernel smoothing 3.3. Sections 3.4 and 3.5 are devoted in how reduce the dimension of data using functional Principal Components and Partial Leas Squares components. Chapter 4 is focused in normed and semi-normed functional spaces and in how the metrics and semi-metrics can be used as measures of proximity between functional data. Chapter 5 is concerned with the concept of statistical depth for functional data referred to in the work of [Cuevas *et al.* (2007)]. These depth measures are used to calculate: Central and dispersion measures section 5.1, bootstrap procedure as a tool for functional data to analyze the variability of estimators (Section 5.2) and functional outliers methods as in [Febrero-Bande *et al.* (2008)] (Section 5.3). Chapter 6 is completely devoted to functional regression models with scalar response from different perspectives: Parametric, using basis representation (Section 6.1, 6.2, 6.3 and 6.4), nonparametric, using smoothing kernel (Section 6.5) and semi-parametric that mix the previous ones (Section 6.6). The Section 6.7 summarizes the functional regression models using a classical example to show the capabilities of the package **fda.usc**. Chapter 7 extend the functional linear model to generalized functional linear model. The **fda.usc** package has implemented other utilities within the field of Functional Data Analysis (FDA) not included in this document and other future works not implemented in **fda.usc** yet, see Chapter 8. Chapter 9 discuss the most important remarks of the package. Finally, the annex includes the code necessary to replicate the examples, figures and tables presented in this document.

Chapter 2

Functional Data Definition: The new R class fdata

2.1 Functional Data Definition

First of all, we must define in a formal way what is a functional variable. Because of its generality and ease of understanding we will follow next definitions.

Definition 2.1. A random variable $\mathcal{X}$ is called a functional variable if it takes values in a functional space $\mathcal{E}$ -complete normed (or semi-normed) space-, [Ferraty and Vieu (2006)].

Definition 2.2. A functional dataset $\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ is the observation of n functional variables $\mathcal{X}_1, \ldots, \mathcal{X}_n$ identically distributed as $\mathcal{X}$, [Ferraty and Vieu (2006)].

The first obstacle that we will always have when analyzing functional data is to find adequate representation for the data. Typically, the functional data set $\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ is evaluated on a number of discretization points $\{t_1, \ldots, t_m\}$ that can be observed in a non-equispaced way.

The **fda.usc** package avoids the basis transformation performed by the **fda** package and define an object called **fdata** as a list of the following components:

- **data**: Typically a matrix of *(n x m)* dimension which contains a set of n curves discretized in m points or **argvals**.
- **argvals**: Locations of the discretization points, by default: $\{t_1 = 1, \ldots, t_m = m\}$.
- **rangeval**: Range of discretization points, by default: **range(argvals)**.
- **names**: Optional list with three components: **main**, an overall title, **xlab**, a title for the x axis and **ylab**, a title for the y axis.

2.2 Displaying functional data using semi-metrics: example tecator

The representation of a functional dataset should be consistent with the physical interpretation of the phenomenon being described. The tecator dataset (**data("tecator")**) has two components. The first component, **tecator$absorp.fdata**, have the curves of absorbance for the analysis of pieces of meat stored in the format of class **fdata**. Each curve consists of a m=100-channel absorbance spectrum measured along the discretized points: **argvals**=$\{t_1 = 850, \ldots, t_m = 1050\}$. The second component, **tecator$y**, it is a data-frame that have **Fat**, **Water** and **Protein** contents of each spectrometric curve obtained by an analytical chemical processing. The boxplot for these variable are drawn in the figure 2.1. The following code loads the package and illustrates the components of tecator dataset (for more details, see the annex 9):

```
R> library(fda.usc)
R> data(tecator)
R> names(tecator)
[1] "absorp.fdata" "y"
R> names(tecator$y)
[1] "Fat"      "Water"    "Protein"
```

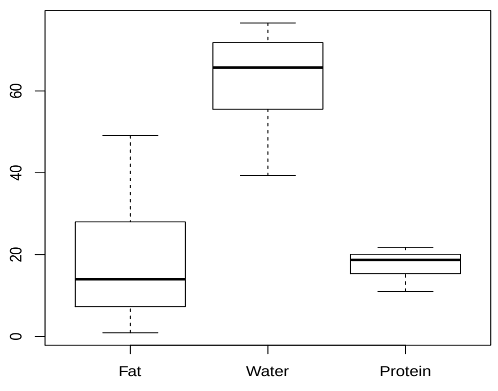

Figure 2.1: Fat (left), Water (middle) and Protein (right) content of each spectrometric curve.

The Tecator dataset (available at http://lib.stat.cmu.edu/datasets/tecator) presents interesting features and it is a well known example in Functional Data Analysis (FDA), therefore we have incorporated into the package .

All the methods in the package can work with the new class **fdata** that only uses the evaluations at the discretization points and it is no necessary to represent the functional data in a basis object. Also, some basic operations to handle this new class are introduced: "+", "-", "*", "/", "[]", "==", **is.fdata()**, **argvals()**, **rangeval()** **c()**, **dim()**, **ncol()**, **nrow()**. Below is an example of how to use these basic functions in an object of "fdata" class (absorbance curves).

```
R> absorp<-tecator$absorp.fdata
R> dim(absorp);nrow(absorp);ncol(absorp)
[1] 215 100
[1] 215
[1] 100
R> rangeval(absorp)
[1]  850 1050
R> is.fdata(absorp)
```

```
[1] TRUE
R> sum1<-absorp[1]+absorp[2]
R> res1<-sum1-absorp[2]
R> res1==absorp[1]
[1] TRUE
```

In this package, other utility functions (standard in **R**) are incorporated to work with the new class **fdata** as for example **plot()**. The following code displays the absorbance curves in function of the fat content, (see left panel of Figure 2.2).

```
R> Fat20<-ifelse(tecator$y$Fat<20,0,1)*2+2}
> names(absorp)
[1] "data"     "argvals"  "rangeval" "names"
R> plot(absorp,col=Fat20)
```

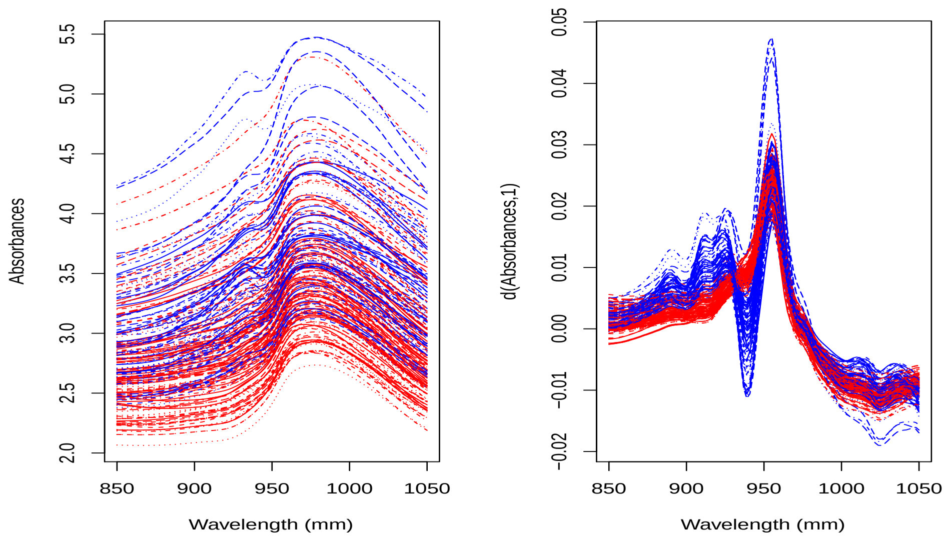

Figure 2.2: Spectrometric curves (left panel): curves with Fat<20% (red lines) and curves with Fat≥20% (blue lines). Corresponding spectrometic curves after first order differencing (right panel).

An important choice to do when analyzing functional data is the space where the data must be considered. For example, in left panel of Figure 2.2, the spectrometric curves are plotted showing with different colors the fat content. This representation which implicitly assumes a $\mathcal{L}_2$ space, is not related with the information of fat content. In other words, the vertical shift of these curves has no special relation with the fat content. So, if we are interested in the relationship between the spectrometric curve and the fat content, probably another choice of functional space would be more advisable as for example, that shown in right panel of Figure 2.2 with the first derivative of the spectrometric curves. In this case, the distance between curves are given by the semi-norm of the derivative $\left(d(f,g)=\sqrt{\int_T (f'(t)-g'(t))^2 dt}\right)$ instead of the norm as in the case of the $\mathcal{L}_2$ space . To calculate the derivative of a functional data the **fdata.deriv()** function has been implemented, which allows compute the derivative many argument by different methods. If **method**

= **“bspline”**, **“exponential”**, **“fourier”**, or **“monomial”** is selected, the command calculates the derivative of fdata object using **deriv.fd{fda}**. If **method=“fmm”**, **“periodic”**, **“natural”** or **“monoH.FC”** is selected, the **splinefun{stats}** function is employed. Raw derivation could be applied with **method=“diff”**, but this is not recommended when the values are not equally spaced o with sparse data.

```
R> absorp.d1 = fdata.deriv(absorp, nderiv = 1 , method = "bspline" )
R> plot(absorp.d1, col = Fat20)
```

2.3 Displaying functional data using fd class: expample aemet

The aemet dataset contains geographic information of 73 Spanish weather stations (see figure 2.3) and the average for the period 1980-2009 of dayly temperature, daily precipitation and daily wind speed in “fdata” class format. This dataset contains the following elements downloaded from Meteorological State Agency of Spain (AEMET), http://www.aemet.es/.

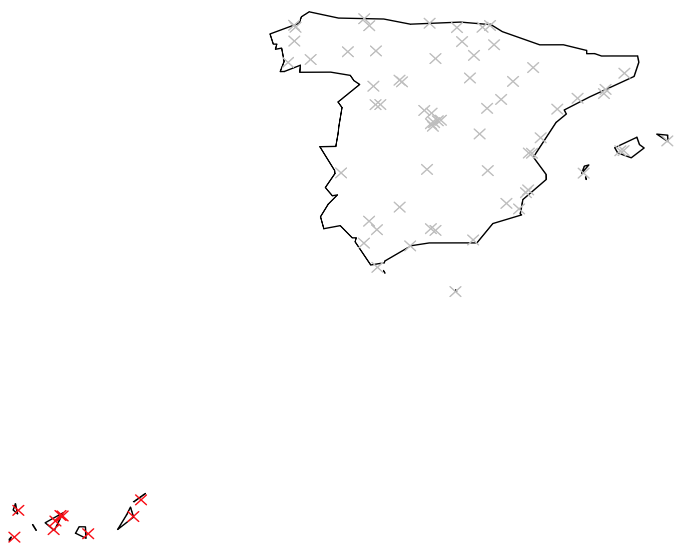

Figure 2.3: Geographic distribution of Spanish weather stations considered in the **aemet** dataset: (red crossing points) stations located on Canary Islands and (gray crossing points) the other weather stations.

1. **aemet$df**: Dataframe with information of each functional data. Index weather station **ind**, station name **name**, region of Spain **province**, altitude of the station in meters **altitude**, x and y geographic coordinate of the station in decimal degrees **longitude** and **latitude** respectively.

2. **aemet$temp**: average daily temperature (in degrees Celsius).

3. **aemet$wind.speed**: average wind speed (in m/sg).

4. **aemet$prec**: daily precipitation (in millimeters mm).

The function **fdata2fd()** converts an object of class **fdata** in to an object of class **fda** using the basis representation hat will be seen in Chapter 3. Inversely, the function **fdata()** converts data object of class: **fda**, **fds**, **fts**, **sfts** or other format (**vector**, **matrix**, **data.frame**) to an object of class **fdata**. A figure 2.4 displays in the left panel (as fdata class objects) the average annual temperature, wind speed and precipitation curves. In the right panel displays (as fd class objects) the corresponding curves represented by 15 basis Fourier basis.

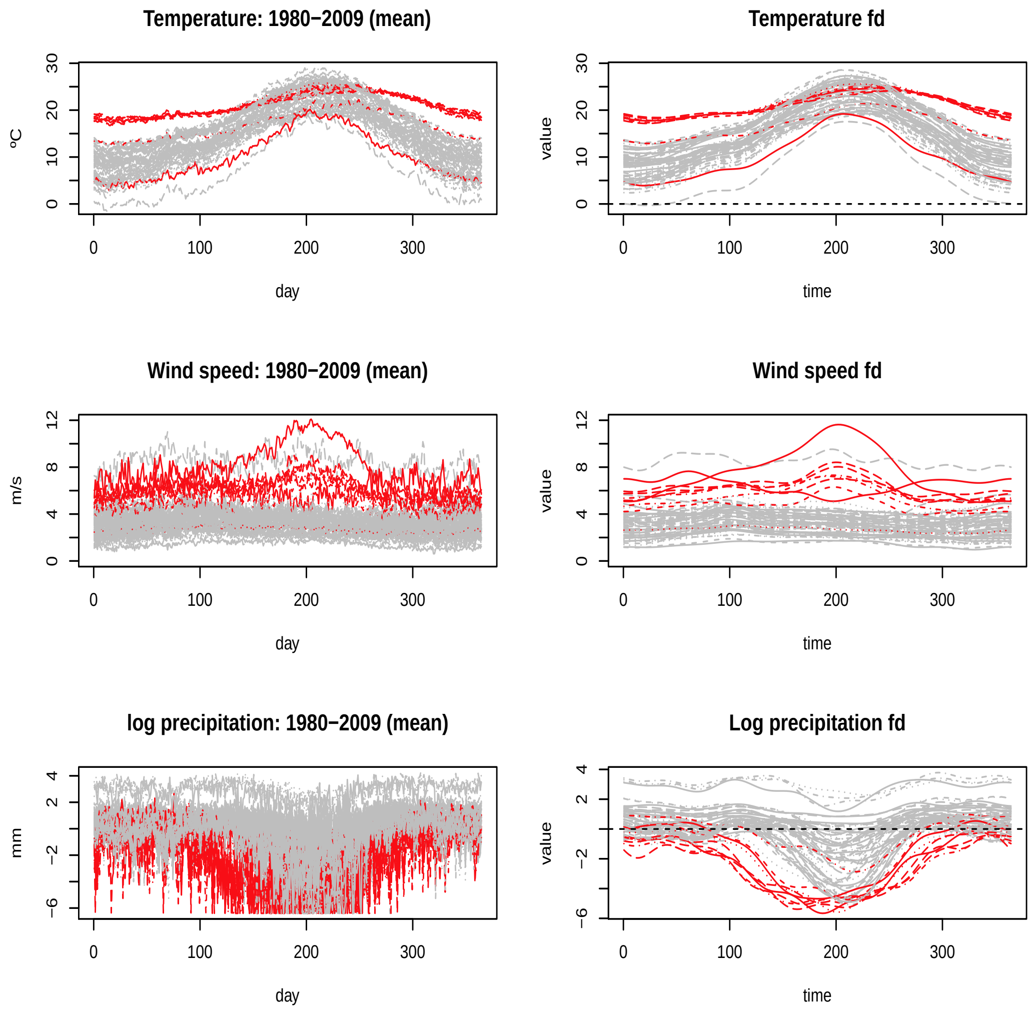

Figure 2.4: The average annual temperature (top row), wind speed (middle row) and precipitation (bottom row), in red are plotting the curves of station located on Canary Islands and the other stations in gray scale. In the left panel displays the data as fdata class objects and in the right panel as fd class objects.

```
R> data(aemet)
R> names(aemet)
```

```
[1] "df"         "temp"       "wind.speed" "logprec"
R> class( temp.fd <- fdata2fd(aemet$temp, type.basis= "bspline", nbasis= 7))
[1] "fd"
R> class(temp.fdata<-fdata(temp.fd)) #back to fdata
[1] "fdata"
```

Chapter 3

Functional data representation: Smoothing

The first step in a functional data analysis maybe the data representation. If we supposed that our functional data $Y(t)$ is observed through the model: $Y(t_i) = X(t_i) + \epsilon(t_i)$ where the residuals $\epsilon(t)$ are independent with $X(t)$, we can to get back the original signal $X(t)$ using a linear smoother:

$$\hat{x} = \sum_{i=1}^{n} s_{ij} y_i \Rightarrow \hat{x} = Sy$$

where s_{ij} is the weight that the point t_j gives to the point t_i and y_i is the observed value of the variable y at point t_i (respectively, $x_i := X(t_i)$). In the package are implemented four procedures for this purpose. The first is the representation in a $\mathcal{L}_2$ basis (or penalized basis) and the second is based on the smoothing kernel methods, the third on the functional principal components (PC) and the fourth on the functional partial least square (PLS) components.

3.1 Basis representation

A curve can be represented by a basis when you assume that the data belong to $\mathcal{L}_2$ space. A basis is a set of known functions $\{\phi_k\}_{k\in\mathbb{N}}$ that any function could be arbitrarily approximate by a linear combination of a sufficiently large number K of these functions, (see page 43 and 44 of [Ramsay and Silverman (2005)]).

The procedure approximates a function $X(t)$ by using a fixed truncated basis expansion in terms of K known basis functions:

$$X(t) = \sum_{k\in\mathbb{N}} c_k \phi_k(t) \approx \sum_{k=1}^{K} c_k \phi_k(t) = \mathbf{c}^{\mathbf{T}}\mathbf{\Phi} \tag{3.1}$$

Using the above basis representation, the projection matrix is given by:
$S = \Phi(\Phi^T W \Phi)^{-1}\Phi^T W$, with degrees of freedom: $df(\nu) = trace(S(\nu)) = K$.

If smoothing penalization is required, also a parameter λ will be provided and in this case the projection matrix S is: $S = \Phi(\Phi^T W\Phi + \lambda R)^{-1}\Phi^T W$, where R is the penalization matrix.

There are several different types of basis (Fourier, Bspline, Wavelets, Exponential, Power, Polynomial,...). Here, we present two of the most used basis (Fourier and Bspline) that can be created using the function **create.fdata.basis()** or directly through the **create.*basis-name*.basis()** function of package **fda** integrated into the functions of the package **fda.usc**.

Fourier basis

Is a periodic basis (period $2\pi/w$) when selecting data $\{t_j\}$ equally-spaced in $T = [0,T]$ and $w = 2\pi/T$ is composed by the following orthonormal functions:

$$\phi_0(t) = 1/\sqrt{T}, \quad \phi_{2r-1}(t) = \frac{sin(rwt)}{\sqrt{T/2}} \quad and \quad \phi_{2r}(t) = \frac{cos(rwt)}{\sqrt{T/2}}$$

The period T (by default) is the range of discretization points t. The right panel of figure 3.1 shows the elements of Fourier basis with period 200 and $K = 7$ constructed by the next command:

```
R> create.fourier.basis(rangeval=absorp$rangeval,nbasis=7)
```

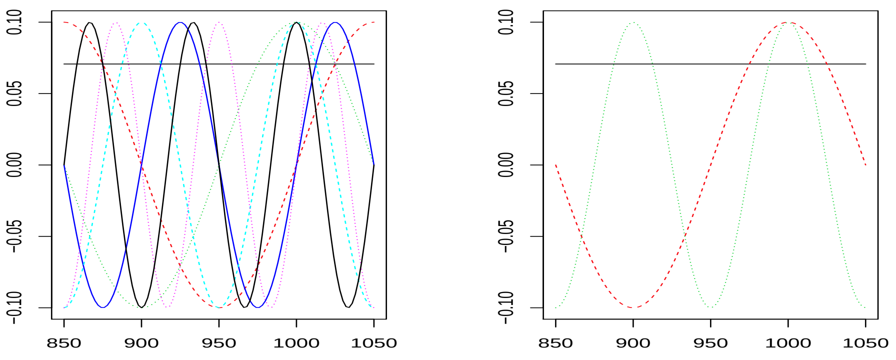

Figure 3.1: Fourier basis with the first 7 basis function (left panel) and with the 1st, 3rd and 5th basis function (right panel).

The following example (see right panel of figure 3.1) shows how to create a fourier basis expansion directly using the information from the functional object of “fdata” class and selecting the basis functions specified by the user.

```
R> create.fdata.basis(tecator$absorp.fdata,
l=c(1,3,5),maxl=7,type.basis="fourier")
```

Base B-spline

The most common basis for non-periodic functional data are the spline functions. A spline is a set of polynomials (of order m) defined in subintervals constructed so that at the end of a subinterval polynomial coincides with the start of the next subinterval polynomial. The cutoff points of subintervals are called “knots”: $\tau = \{\tau_l\}_{l=0}^{L}$.

Among the different types of splines we use the B-spline basis because they are fast in computation of polynomials and flexible in representation. The B-splines basis *"bspline"* are easily calculated with the de Boor algorithm and can be expressed as:

$$bspline(t) = \sum_{k=1}^{L-1+m} c_k B_k(t, \tau)$$

where $L-1$ is the number of interior nodes and m the order of the polynomial.

The following command set up a B-Spline basis of order 4 with 3 equally spaced interior knot over the interval $[0, 1]$.

```
R> create.bspline.basis(rangeval=c(0,1),nbasis=5)
```

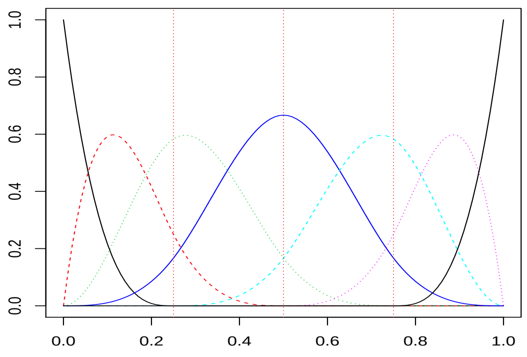

Figure 3.2: B-Splines basis with $L = 5$.

3.2 Validation criterion

Typically, the infinite dimension (or the high dimension) of the functional data is hard to manage but this method projects functional data into a K-dimensional space which is clearly more tractable.

The choice of the parameter number of basis and the most appropriate basis for the observed data is also vital and, in principle, there is no universal rule that would enable an optimal choice. The decision on what basis to choose should be based on the objective of the study and on the data. For example, is common to use the *"fourier"* basis for periodic data, and *"bspline"* basis for non-recurrent data. Among different selection criteria to select the parameter $\nu = (K, \lambda)$, we have implemented the following two: Cross Validation (CV) and Generalized Cross Validation (GCV). The package **fda.usc** incorporates **min.basis()** function that represent the functional data in basis by the selection of a reasonable number of basis $\nu_1 = K$ through CV or GCV criteria. Also, the penalty parameter $\nu_2 = \lambda$ is included in this process. Both criteria are defined as follows:

$$\text{Cross-validation} : CV(\nu) = \frac{1}{n} \sum_{i=1}^{n} \frac{\left(y_i - \hat{r}^{\nu}(x_i)\right)^2}{1 - S_{ii}} w(x_i) \tag{3.2}$$

where $\hat{r}^{\nu}_{-i}(x_i)$ is the prediction at point t_i obtained by omitting the i pair (x_i, y_i), S_{ii} is the i diagonal element of the smoothing matrix S (with $\nu = trace(S)$) and $w(x_i)$ is the weight of data x at point t_i. This criteria is implemented in the function **CV.S()**.

$$\text{Generalized Cross-validation} : GCV(\nu) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{r}^{\nu}_{i}(x_i)\right)^2 w(x_i)\Xi(\nu) \tag{3.3}$$

where Ξ denotes the type of penalizing function.

Generalized Cross-Validation criteria is implemented in **GCV.S()** function with the following types of Ξ functions (see [Härdle (1990)]):

- Generalized Cross-validation (GCV): $\Xi(\nu) = (1 - tr(S)n^{-1})^{-2}$.
- Akaike's Information Criterion (AIC): $\Xi(\nu) = exp(2tr(S)n^{-1})$.
- Finite Prediction Error (FPE): $\Xi(\nu) = (1 + tr(S)n^{-1})/(1 - tr(S)n^{-1})$.
- Shibata's model selector (Shibata): $\Xi(\nu) = (1 + 2tr(S)n^{-1})$.
- Rice's bandwidth selector (Rice): $\Xi(\nu) = (1 - 2tr(S)n^{-1})^{-1}$.

3.3 Kernel smoothing

Another way to represent the functional data is the nonparametric one. The problem is to estimate the smoothing parameter or bandwidth $\nu = h$ that better represents the functional data using kernel smoothing.

Now,the nonparametric smoothing of functional data is given by the smoothing matrix S:

$$s_{ij} = \frac{1}{h}K\left(\frac{t_i - t_j}{h}\right)$$

Different types of kernels $K()$ are defined in the package (see figure 3.3):

- Gaussian: $K(u) = \frac{1}{\sqrt{2\pi}}exp\left\{-u^2/2\right\}$
- Epanechnikov: $K(u) = \frac{3}{4}1_{[-1,1]}(1 - u^2)$
- Triweigth: $K(u) = \frac{35}{32}1_{[-1,1]}(1 - u^2)^3$
- Uniform: $K(u) = \frac{1}{2}1_{[-1,1]}(u)$
- Cosine: $K(u) = \frac{\pi}{4}1_{[-1,1]}cos(\pi * u/2)$
- Quartic: $K(u) = \frac{15}{16}1_{[-1,1]}(1 - u^2)^2$

The **min.np()** function returns the value of the bandwidth parameter **h.opt** that best represents the functional data $\hat{x} = \hat{r}^{\nu}_{n}(x) = \sum_{i=1}^{n} s_i(x)Y_i$ for a range of values of bandwidth ν using the validation criteria shown above, (Equation 3.2 and 3.3).

Among other features, the package **fda.usc** allows calculate the smoothing matrix **S** by, (see [Wasserman (2006)] and [Ferraty and Vieu (2006)]):

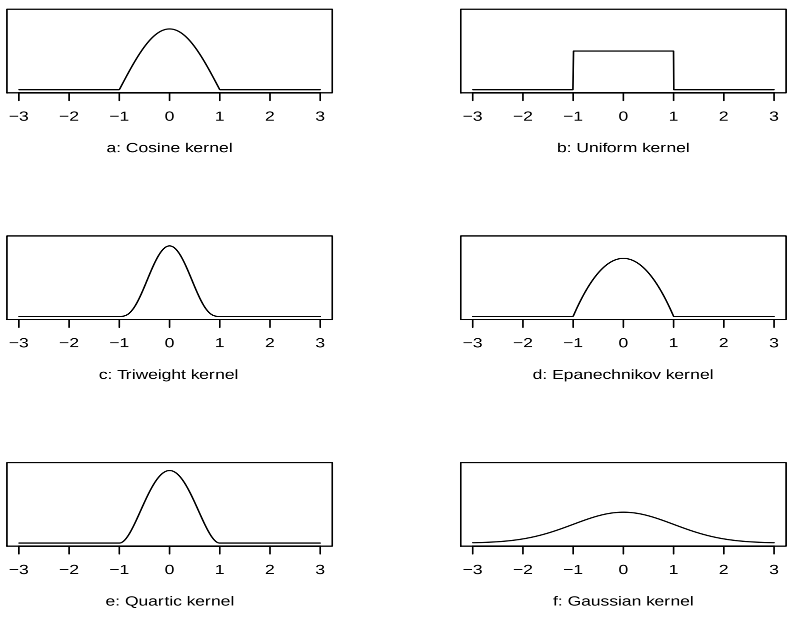

Figure 3.3: Symmetric Kernels

- k nearest neighbour (kNN) method (**S.KNN**): $s_i(x) = \frac{\sum_{\{i:\ y_i=g\}}^{n} K\left(\frac{t-t_i}{h_k}\right)}{\sum_{i=1}^{n} K\left(\frac{t-t_j}{h_k}\right)}$

  where K the kernel function and h_k is the bandwidth such the $card\,\{i : |t - t_i| < h_k\} = k$.

- Nadaraya-Watson method (**S.NW**): $s_i(x) = \frac{K\left(\frac{t-t_i}{h}\right)}{\sum_{j=1}^{n} K\left(\frac{t-t_j}{h}\right)}$

  where K is the kernel function adn h is the bandwidth $(h > 0)$.

- Local linear regression method (**S.LLR**): $s_i(x) = \frac{\frac{1}{h}K\left(\frac{t-t_i}{h}\right)(S_{n,2}(t)-(t-t_i)S_{n,1}(t))}{\sum_{j=1}^{n}\frac{1}{h}K\left(\frac{t-t_j}{h}\right)(S_{n,2}(t)-(t-t_i)S_{n,1}(t))}$.
  where $S_{n,j}(t) = \sum_{i=1}^{n} K\left(\frac{t-t_i}{h}\right)(t-t_i)^j,\ j = 1, 2$

The following describes a 3^{rd} functional dataset included in the package: **data(”phoneme”)**[1]. The aim of *Phoneme dataset* is to discriminate five phonemes from log-periodograms. The object **phoneme$classlearn** contains 250 speech frames with class membership: “aa” (1), “ao” (2), “dcl” (3), “iy” (4) and “sh” (5). From each speech frame, a log-periodogram of length 150 have been stored in **phoneme$learn** which is used as learning sample. The goal is to predict the class membership **phoneme$classtest** using the test sample **phoneme$test**, (see [Ferraty and Vieu (2006)]).

In the following example (see next **R** code), **phoneme$learn** is smoothed using **min.basis()** and **min.np()** functions. In left panel of Figure 6.6, the GCV criterion is drawn in function of the number of *bspline* basis $\nu_1 = nbasis$ and penalizing parameter $\nu_2 = \lambda$ using **min.basis()** function. While in right panel of Figure 6.6, the GCV criterion is drawn in function of the bandwidth $\nu = h$

[1]This dataset includes the changes realised in the file
`http://www.math.univ-toulouse.fr/staph/npfda/npfda-phondiscRS.txt`.

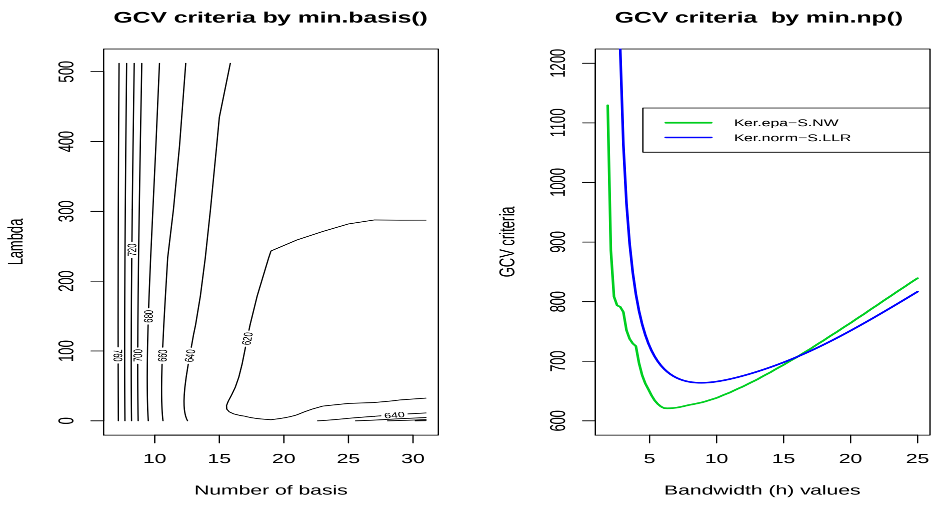

Figure 3.4: GCV criteria as a function of number of bspline basis and penalizing parameter λ (left panel). GCV criteria as a function of bandwidth choice (right panel): Normal kernel and local linear smoothing matrix (blue line); Epanechnikov kernel and Nadaraya-Watson smoothing matrix (green line).

using the function **min.np()**.

```
R> data("phoneme")
R> learn <- phoneme$learn
R> l <- c(0, 2^seq(-2, 9, len = 30))
R> nb <- seq(7, 31, by = 2)
R> out0 <- min.basis(learn, lambda = l, numbasis = nb)
 The minimum GCV (GCV.OPT=611.4684) is achieved with
 the number of basis (numbasis.opt=27)
 and lambda value    (lambda.opt=81.27995)
R> out1 <- min.np(learn, Ker = Ker.epa)
 The minimum GCV (GCV.OPT=621.0195) is achieved with
 the h value (h.opt=6.3131)
R> out2 <- min.np(learn, type.S = S.LLR)
 The minimum GCV (GCV.OPT=663.8132) is achieved with
 the h value (h.opt=8.8826)
```

Continuing the previous example, Figure 3.5 shows a 11^{th} curve of **phoneme$learn** and three smooth representations of the curve.

3.4 Functional Principal Components Analysis (FPCA)

Another tool to reduced dimensional space is the Principal Components Analysis (PCA). This method allows display the functional in a few components. The tool is designed to explain a functional data through a combination of orthonormal variables that satisfies property to maximize the variance following the next algorithm:

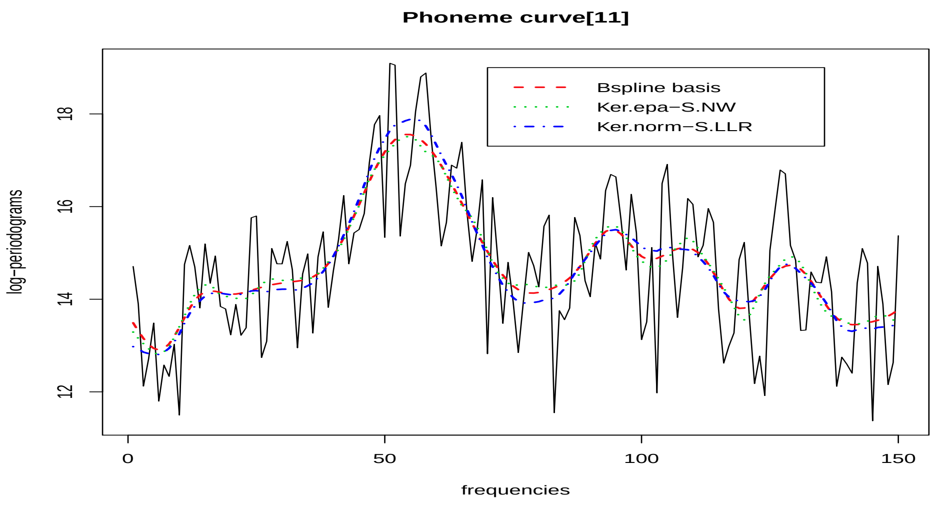

Figure 3.5: Phoneme curve[11]: Observed (black solid line), smoothed by 27 bspline basis and $\lambda = 81.28$ (red dashed line), smoothed by Epanechnikov kernel and Nadaraya-Watson smoothing matrix with bandwidth $h = 8.88$ (green dotted line) and smoothed by normal kernel and local linear smoothing matrix with bandwidth $h = 6.31$ (blue dashed line).

1. Find the function $\xi_1(t)$ of norm 1 $\left(\int \xi_i(t)^2 dt = 1\right)$ such that $f_i = \int \xi_i(t) X_i(t) dt = 1$ maximize $\sum_{i=1}^{p} \|f_{i1}\|^2$.
2. Repeat step 1, also in step m requires that $\xi_m(t) \perp \xi_k(t),\ k < m$

The functional data can be rewritten as a decomposition in an finite orthonormal basis:

$$\hat{X}_i(t) = \sum_{i=1}^{K} f_{ik} \xi_k(t) \tag{3.4}$$

where f_{ik} is the score of the principal component PC.

In **fdata2pc()** we uses a singular value decomposition **svd** to find the principal components ξ_k, and calculates the scores as the inner product between the center functional data X_c and ξ_k: $f_{ik} = \int X_i \xi_k = \langle X_c, \xi_k \rangle$.

In FDA this method can be use for represent the functional data and summary quickly information from the sample. In the next code we use the function **fdata2pc()** for compute the first 3 principal components of absorbace curves (see left panel of figure 3.6) where the norm of eigenvectors (or rotation) is 1.

```
R> pc.svd<-fdata2pc(tecator$absorp,ncomp=3)
R> norm.fdata(pc.svd$rotation[1:2])
     [,1]
[1,]    1
[2,]    1
attr(,"call")
```

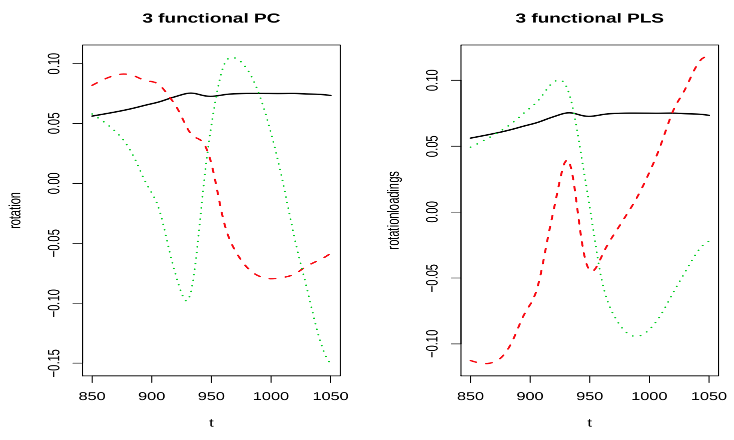

Figure 3.6: In left panel, first 3 PC of absorbances curves and in right, first 3 PLS of absorbances curves. 1st component (black line), 2nd component (red line) and 3rd component (green line).

```
metric(fdata1 = fdataobj, fdata2 = z0)
R> plot(pc.svd$rotation,main="",lwd=2)
```

The output of **fdata2pc()** function is of "fdata.comp" class such **summary.fdata.comp()** function returns the variability of each PC and the biplot (see figure 3.7). For absorbances curves, the first principal component explain the 88% of the variability of data and the next PC less of 1%.

```
R> summary(pc.svd)

     - SUMMARY:   fdata2pc  object   -

-With 3  components are explained  89.27 %
 of the variability of explicative variables.

-Variability for each component (%):
  PC1   PC2   PC3
88.18  0.81  0.28
```

If a scalar response is observed, we can compute correlation between the functional principal components and scalar response. Example correlation between *Fat* content and the principal components of absorbances curves (or one of two first derivatives):

```
R> pc.svd<-fdata2pc(X,ncomp=3)
R> pc.svd1<-fdata2pc(X.d1,ncomp=3)
R> pc.svd2<-fdata2pc(X.d2,ncomp=3)
R> summary(pc.svd,fat,biplot=FALSE,corplot=TRUE)

-Correlations:
cor(y,PC1) cor(y,PC2) cor(y,PC3)
     0.443     -0.132     -0.468
R> summary(pc.svd1,fat,biplot=FALSE,corplot=TRUE)
-Correlations:
```

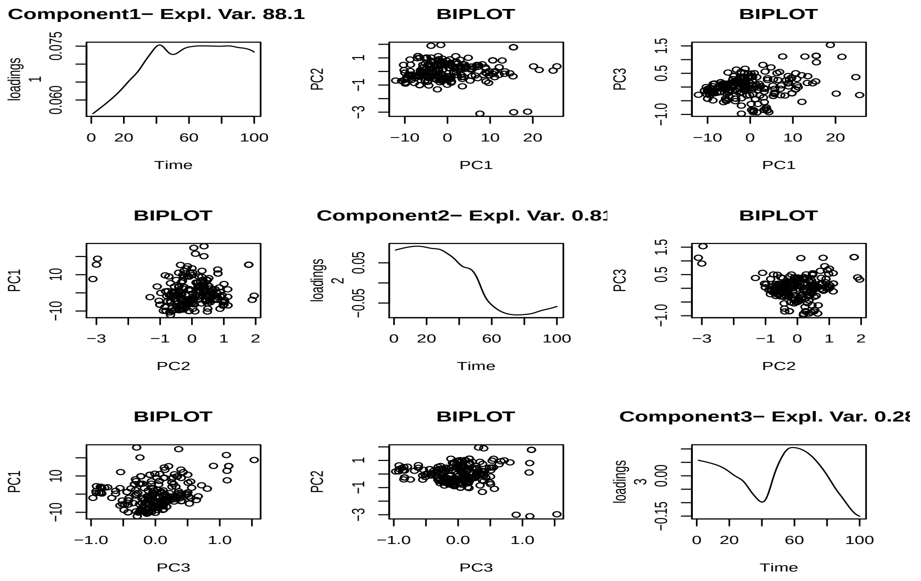

Figure 3.7: The diagonal of the figure shows the three PC and out are drawing the biplot between the scores of each PC.

```
cor(y,PC1) cor(y,PC2) cor(y,PC3)
     0.940      0.114      0.048
R> summary(pc.svd2,fat,biplot=FALSE,corplot=TRUE)
-Correlations:
cor(y,PC1) cor(y,PC2) cor(y,PC3)
     0.950     -0.193     -0.009
```

The correlation between response and the first principal component is greater when the first and second derivatives of the curves of absorbance (94% and 95% respectively) than the curves without deriving (44%).

In addition, in functional principal components analysis can be many uses: for computing proximities between curves (see section 3.3), or in functional regression model (section) and to detect outliers (although they can hide).

3.5 Functional Partial Least Squares (FPLS)

In the previous section, functional principal components (FPC) is good solution for represent the data in a reduced dimension spaced, [Martens and Naes (1989)]. But when a scalar variable is observed additionally you can use directly this information by Functional Partial Least Squares. The basic idea is to construct a set of PLS components in the linear space spanned by $X(t)$, taking into account the correlation between Y and $X(t)$.

This package uses the function **plsr()** [Mevik and Wehrens (2007)] that uses iterative algorithmic methods to extract these linear spans that have a large covariance with y. The following code shows how to calculate the first 3 partial least squares of absorbance curves and content of fat as additional real response. In right panel of figure 3.6 are plotting the 3 PLS, where now, the 2nd and 3rd PLS component changes its shape over the 2nd and 3rd PC, while the 1st PLS and 1st PC have the same shape. As previous section, the output of **fdata2pls()** function is of "fdata.comp" class. We use **summary.fdata.com()** to resume the information about variability of each PLS and displaying the biplot and/or correlation between the y response and the PLS components.

```
R> pls1<-fdata2pls(X,fat,ncomp=3)
R> summary(pls1,biplot=FALSE)

     - SUMMARY:   fdata2pls  object   -

-With 3  components are explained  100 %
 of the variability of explicative variables.

-Variability for each component (%):
 PLS1  PLS2  PLS3
97.81  1.37  0.82
```

Different application of FPLS methods can be done as in [Preda et al. (2007)] for classification or in [Escabias *et al.* (2007)] fit functional PLS logit model for regression, see section 7. Section 4.4 shows how to calculate proximities between curves using the PLS components and section 6.3 describes the functional PLS regression model.

Chapter 4

Measuring distances of Functional Data

It is difficult to find the best plot given a particular functional dataset because the shape of graphics depends strongly on the proximity measure. As shown in Figure 2.2, the plot of $X(t)$ against t is not necessarily the most informative and other semi-metric can allow to extract much information from functional variables. This package collects several metric and semi-metric functions which allow to extract as much information possible from the functional variable.

The most simple spaces for functional data are the complete metric spaces where only the notion of distance between elements of the space is given. If the metric can be expressed as $d(X(t), Y(t)) = \|X(t) - Y(t)\|$ with a norm $\|\cdot\|$ verifying the triangle inequality, we have a normed space (or a Banach space). In these spaces, there is also the notion of size of the elements of the space. If the norm verifies the paralelogram law $(\|x+y\|^2 + \|x-y\|^2 = 2(\|x\|^2 + \|y\|^2))$, an inner product can be defined in the space in the following way: $\langle x, y\rangle = (1/4)(\|x+y\|^2 - \|x-y\|^2)$. A complete space with an inner product is called a Hilbert space which is a special kind of Banach spaces where $\|X(t)\| = \sqrt{\langle X(t), X(t)\rangle}$.

Utilities for computing distances are included in the package. Below are described those but of course, many others can be built with the only restriction that the first two arguments correspond to class **fdata**. In addition, the procedures of the package **fda.usc** that contains the argument **metric** allow the use of metric or semi-metric functions as shown in the following sections.

4.1 metric.lp() function

If we focused on $\mathcal{L}_p$ spaces (the set of functions whose absolute value raised to the p-th power has finite integral), **metric.lp()** uses Simpson's rule to compute distances between elements:
Let $f(t) = X_1(t) - X_2(t)$.

$$\|f\|^p = \left(\frac{1}{\int_a^b w(t)dt}\int_a^b |f(t)|^p\, w(t)dt\right)^{1/p}$$

where w are the weight. The observed points on each curve t are equally spaced (by default) or not.

For example, the distances between two curves (see figure 4.1) $X_1(t) = \{sin(x)\}_{x=0}^{2\pi}$ y $X_2(t) = \{0\}_{x=0}^{2\pi}$ are calculated in the next code:

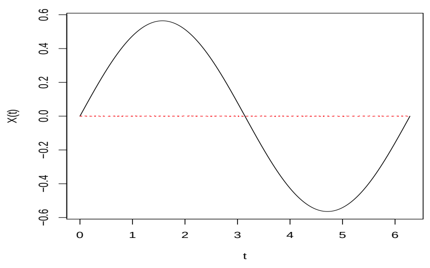

Figure 4.1: FPCA for tecator datatset.

```
R>  r<-rnorm(1001,sd=.001)
R>  x<-seq(0,2*pi,length=1001)
R>  fx<-fdata(sin(x)/sqrt(pi),x)
R>  fx0<-fdata(rep(0,length=length(x))+r,x)
R>  plot(c(fx,fx0))
R>  metric.lp(fx,fx0)
            [,1]
[1,] 0.9999135
attr(,"call")
metric.lp(fdata1 = fx, fdata2 = fx0)
R>  integrate(function(x){(sin(x)/sqrt(pi))^2},0,2*pi)
1 with absolute error < 7.3e-10
```

4.2 cos.cor.dist() function

This function computes the cosine correlation distance between two functional dataset. In other words, calculate the cosine of the angle between the two curves such that for distances close to 1, indicates uncorrelated curves, and distances close to 0, indicates that the curves are highly correlated .

$$d(X_1(t), Y_2(t)) = \left| \frac{\langle X_1(t), Y_2(t)\rangle}{\sqrt{\|X_1(t)\| \, \|Y_2(t)\|}} \right|$$

```
R> 1-dis.cos.cor(fx,fx0)
          [,1]
[1,] 1.000211
```

4.3 semimetric.basis() function

This function approximates semi-metric distances for functional data of two “fdata” (or “fd”) class objects. If functional data are not functional fd class, the function creates a basis to represent the functional data, by default is used **create.bspline.basis()** and the fdata class object is converted to “fd” class using the **Data2fd()**. The function calculates distances between the derivative of order **nderiv** of curves using **deriv.fd()** function.

```
R>  semimetric.basis(fx,fx0)
          [,1]
[1,] 0.9999592
attr(,"call")
metric.lp(fdata1 = fd1, fdata2 = fd2)
```

4.4 Other semi-metrics

A collection of semi-metrics proposed by [Ferraty and Vieu (2006)] are also included in the package.

(i) semimetric.deriv() function. This function computes proximities between curves based on derivatives of functional data. The semi-metric given two curves X_1 and X_2 is:

$$d(X_1, X_2) = \sqrt{\int_T (X_1^{(q)}(t) - X_2^{(q)}(t))^2 dt}$$

where $X_1^q(t)$ denot the q-th derivative of X. [Ferraty and Vieu (2006)] use a B-spline approximation for each curve in a sufficiently fine grid and the derivatives are directly computed by differentiating several times their analytic form.

Using the equation 3.1 the derivatives of the approximated curves by B-spline are:
$\hat{X}_i^{(q)} = \sum_{b=1}^{B} \hat{B}_{ib} B_b^{(q)}$ and the semi-metric can written now as:

$$d(X_1, X_2) = \sqrt{\int_T (\hat{X}_1^{(q)}(t) - \hat{X}_2^{(q)}(t))^2 dt}$$

where the integral is computing by "Gauss method" following next approximation:

$$\int_a^b f(t)dt \approx \frac{b-a}{2} \sum_{k=1}^{k} w_k f\left(\frac{b-a}{2} + \frac{b-a}{2}\delta_k\right)$$

where w_k are the weights and δ_k are the gauss points, see [Ferraty and Vieu (2006)] pag 32-33.

```
R> semimetric.deriv(fx,fx0)/sqrt(2*pi)
          [,1]
[1,] 0.9992886
attr(,"call")
semimetric.deriv(fdataobj1 = fx, fdataobj2 = fx0)
```

(ii) semimetric.pca() function. This function computes proximities between curves based on the functional principal components analysis (FPCA) method. The FPCA reduces the functional data in a reduced dimensional space (q functional principal components).

$$d_q^{FPCA} = \sqrt{\sum_{k=1}^{q} \left(\int [X_i(t) - f_i(t)]\, \xi_k(t) dt \right)^2}$$

where f_i is the score of the principal component $\int X_i \xi$.

The FPCA semi-metric for two curves X_1 and X_2 is calculated as:

$$d_q^{FPCA}(X_1, X_2) = \sqrt{\sum_{k=1}^{q} \left(\sum_{j=1}^{J} w_j (f_1(t_j) - f_2(t_j)) \left[\xi_k\right]_j \right)^2}$$

where ξ_i is the i-th orthonormal eigenvector of the covariance matrix $W = diag(w_1, \ldots, w_J)$ with quadrature weights, in this case we use $w_j = t_j - t_{j-1}$.

(iii) semmimetric.mplsr() function. This function computes proximities between curves a PLS semi-metric based on the functional partial least-squared (FPLS) method. The FPLS uses a scalar response observed additionally to reduce the functional data in a q functional PLS components.

We consider n scalar responses, the PLS semi-metric with q factors is defined as:

$$d_q^{PLS}(X_1, X_2) = \sqrt{\sum_{k=1}^{p} \left(\sum_{j=2}^{J} w_j (f_1(t_j) - f_2(t_j)) \left[\xi_k^q\right]_j \right)^2}$$

with quadrature weights $w_j = t_j - t_{j-1}$.

PCA and PLS semi-metrics can be used only if the curves are observed at the same discretized points and in a grid sufficiently fine.

Other semi-metric implemented by [Ferraty and Vieu (2006)] are: **(iv) semimetric.fourier()** that computes proximities between curves based on Fourier expansion. Similarly to the previous **semimetric.deriv()**, but now Fourier expansion is used instead of a Bspline expansion.

```
R>  semimetric.fourier(fx,fx0)*sqrt(2*pi)
         [,1]
[1,] 1.000776
attr(,"call")
semimetric.fourier(fdataobj1 = fx, fdataobj2 = fx0)
```

(v) semimetric.hshift() that computes proximities between curves taking into account an horizontal shift effect.

4.5 Semi-metric as classification rule: example phoneme dataset

In the next example, the distances between some training sample curves (**phoneme$learn**) of phoneme dataset are calculated by metric function: **metric.lp()**, and semi-metrics functions: **semimmetric.basis()** based on their Bspline expansion, **semmimetric.pca()** based on the functional principal components analysis method and **semmimetric.mplsr()** based on the partial least squares method. Figure 4.2 shows the dendograms for a selection of 11 curves of class **(3)** (indexes of 110 to 120 curves) and 11 curves of class **(5)** (index curves of 220 to 230). This example can be understood as a classification problem in which the goal is to classify the curves in 2 classes. It is noted that in this example the **semimmetric.mplsr()** function (which uses memberclass information) is the only one that properly classifies the 22 curves.

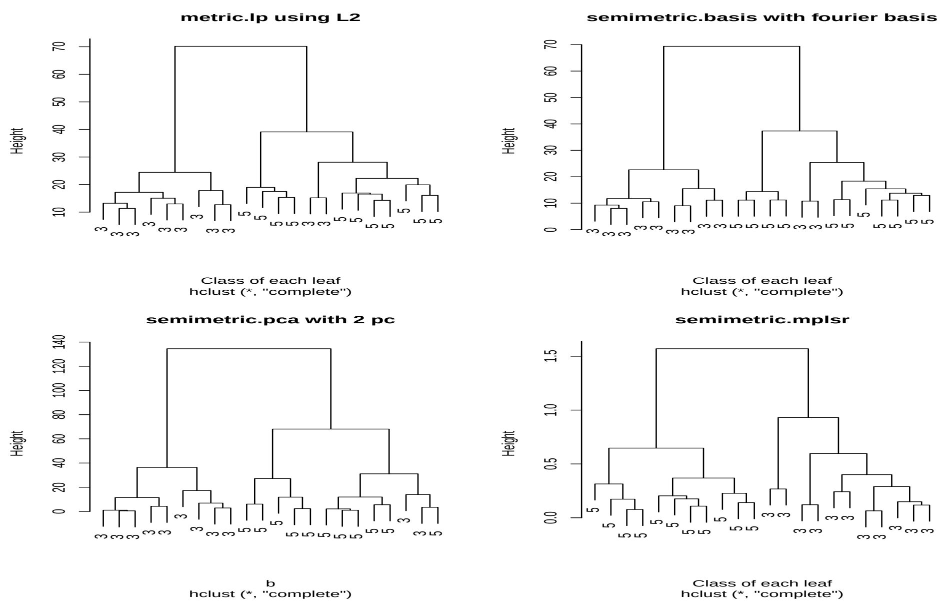

Figure 4.2: Dendograms for 22 phoneme curves: Dendogram using $\mathcal{L}_2$ metric **metric.lp()** (top left), dendogram using $\mathcal{L}_2$ metric with bspline basis representation **semimetric.basis()** (bottom left), dendogram using 2 principal components **semimetric.pca()** (top right) and dendogram using a semi-metric based on the partial least squares method **semimetric.mplsr()** (bottom right).

```
R> glearn = phoneme$classlearn
R> mdist1 <- metric.lp(learn)
R> mdist2 <- semimetric.basis(learn, type.basis1 = "fourier")
R> mdist4 <- semimetric.pca(learn, learn)
R> mdist5 <- semimetric.mplsr(learn, learn, q = 3, class1 = glearn)
```

Chapter 5

Descriptive analysis of Functional Data

Any statistical study should begin with an exploratory part. In **fda.usc**, the usual tools for summarize functional data are included:

- functional mean: $\overline{X}(t) = N^{-1} \sum_{i=1}^{N} X_i(t)$, by **func.mean()** function.
- functional variance:$Var(X(t)) = (N-1)^{-1} \sum_{i=1}^{N} (X_i(t) - \overline{X}(t))^2$, by **func.var()** function

The return of this tools is always an object of class **fdata**.

5.1 Functional Depth

Although, most often are other measures used to summarize a set of functional data such as depth measures. The depth is a concept emerged in the literature of robustness which measures how deep (or central) is a datum respect to a population. In univariate data, the median would be the deepest point of clouds of points. Although there is more depth measures, this package includes those that are contained in the work of [Cuevas *et al.* (2007)]:

- **depth.FM()**: The depth measure is based on the median, [Fraiman and Muniz (2001)]. For every $t \in [0,1]$, let $F_{n,t}$ be the empirical distribution of the sample $X_1(t), \ldots, X_n(t)$ and let $Z_i(t)$ denote the univariate depth of the data $X_i(t)$ in this sample, given by $D_i(t) = 1 - |1/2 - F_{n,t}(X_i(t))|$. Then, define for $i = 1, \ldots, n$,

  $$I_i = \int_0^1 D_i(t)dt$$

  and rank the observations $X_i(t)$ according to the values of I_i.

- **depth.mode()**: The depth measure is based on how surrounded the curves are respect to a metric or a semi-metric distance, selecting the trajectory most densely surrounded by other trajectories of the process [Cuevas *et al.* (2007)]. The population h-depth of a datum z is given by the function:

  $$f_h(z) = E\left(K_h(\|z - X\|)\right)$$

  where X is the random element describing the population, $\|.\|$ is a suitable norm and $K_h(t)$ is a re-scaled kernel and tuning parameter h. An given a random sample $X_1, \ldots, X_n$ of X, the empirical h-depth is defined as:

$$\hat{f}_h(z) = n^{-1} \sum_{i=1}^{n} K_h(\|z - X_i\|)$$

- **depth.RP()**: The depth measure is calculated through random projections (RP) based on the Tukey depth, [Cuevas *et al.* (2007)].

  Given a sample $X_1, \ldots, X_n$ let us take a random direction a (independent from the X_i) and project the data along this direction. Then, the sample depth of a datum X_i is defined as the univariate depth of the corresponding one-dimensional projection (expressed in terms of order statistics so that the median is the deepest point). When the sample is made of functional data, we will assume throughout that the X_i belong to the Hilbert space $\mathcal{L}_2[0,1]$ so that the projection of a datum X is given by the standard inner product $\langle a, X\rangle = \int_0^1 a(t)X(t)dt$. In the finite-dimensional case the projection of $X = (\xi_1, \ldots, \xi_d)$ along the direction a is evaluated through the usual Euclidean inner product $a_1\xi_1 + \ldots + a_d\xi_d$, denoted also by $\langle a, X\rangle$.

- **depth.RPD()**: The depth measure is calculated through random projections of the curves and theirs derivatives, [Cuevas *et al.* (2007)].

  Let $X_1, \ldots, X_n$ be a sample of differentiable functions defined on $[0, 1]$. The basic idea is to use the method of random projections simultaneously (for the functions and their derivatives) thus incorporating the information on the function smoothness provided. The sample of functions is reduced to a sample in $\Re^2$ defined by:
  $\langle a, X_1\rangle, \left\langle a, X^{'}\right\rangle, \ldots, \langle a, X_n\rangle), \left\langle a, X_n^{'}\right\rangle$, where a is a randomly chosen direction. Then the depth of the bidimensional sample data is evaluate in a second step using the depth function defined above: FM depth, mode depth or RP depth.

All depth functions implemented in the package return:

- **median**: Deepest curve.
- **lmed**: Index of the deepest curve.
- **mtrim**: Mean of $(1 - \alpha)\%$ deepest curves.
- **ltrim**: Index of $(1 - \alpha)\%$ deepest curves.
- **dep**: Depth of each curve.

If argument *draw=TRUE* this depth function also displays (see figure 5.1) the curves (in terms of its depth on a gray scale), the median curve (red line) and the mean of $(1 - \alpha)\%$ deepest curves (yellow line).

Another interesting use of functional depth is as measure of dispersion. The top row of Figure 5.2 displays how the functional depth are used as measures of central tendency, the α-trimmed means (top left) and the medians (top right). The bottom row of Figure 5.2 shows the marginal variance using trimmed subsets (bottom left), and the depths calculated by **depth.FM()** and **depth.mode()** (bottom right).

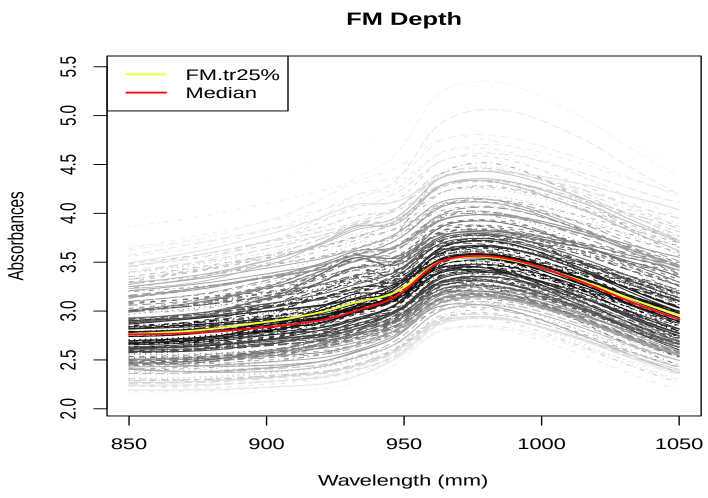

Figure 5.1: Descriptive statistics for Tecator dataset based on depth (in grayscale), 15% trimmed mean curve (red line) and the median curve (yellow line)

```
R> plot(func.mean(absorp), main = "Centrality measures (15% trimmed mean)",
+     ylim = c(2.6, 3.6))
R> lines(func.trim.FM(absorp, trim = 0.15), col = 2)
R> lines(func.trim.mode(absorp, trim = 0.15), col = 3, lty = 3)
R> lines(func.trim.RP(absorp, trim = 0.15), col = 4, lty = 4)
R> plot(func.mean(absorp), main = "Centrality measures (median)",
+ ylim = c(2.6,3.6))
R> lines(func.med.FM(absorp), col = 2)
R> lines(func.med.mode(absorp), col = 3, lty = 3)
R> lines(func.med.RP(absorp), col = 4, lty = 4)
R> plot(func.var(absorp), main = "Dispersion measures", ylim = c(.07,.31))
R> lines(func.trimvar.FM(absorp, trim = 0.15), col = 2)
R> lines(func.trimvar.mode(absorp, trim = 0.15), col = 3, lty = 3)
R> lines(func.trimvar.RP(absorp, trim = 0.15), col = 4, lty = 4)
R> out.FM = depth.FM(absorp, trim = 0.1, draw = FALSE)
R> out.mode = depth.mode(absorp, trim = 0.1, draw = FALSE)
R> plot(out.mode$dep, out.FM$dep, main = "FM depth vs mode depth",
+     xlab = "mode depth", ylab = "FM depth")
```

In the previous code we have employed some shortcut functions as follows:

- For central tendency: $func.\{centr\}.\{depth\}$,
- For dispersion measure (or marginal variability measure): $func.trimvar.\{depth\}$

where $centr = \{med, trim\}$ indicates whether it is used the median or trimmed mean and $depth = \{FM, mode, RP, RPD\}$ indicates the type of depth used.

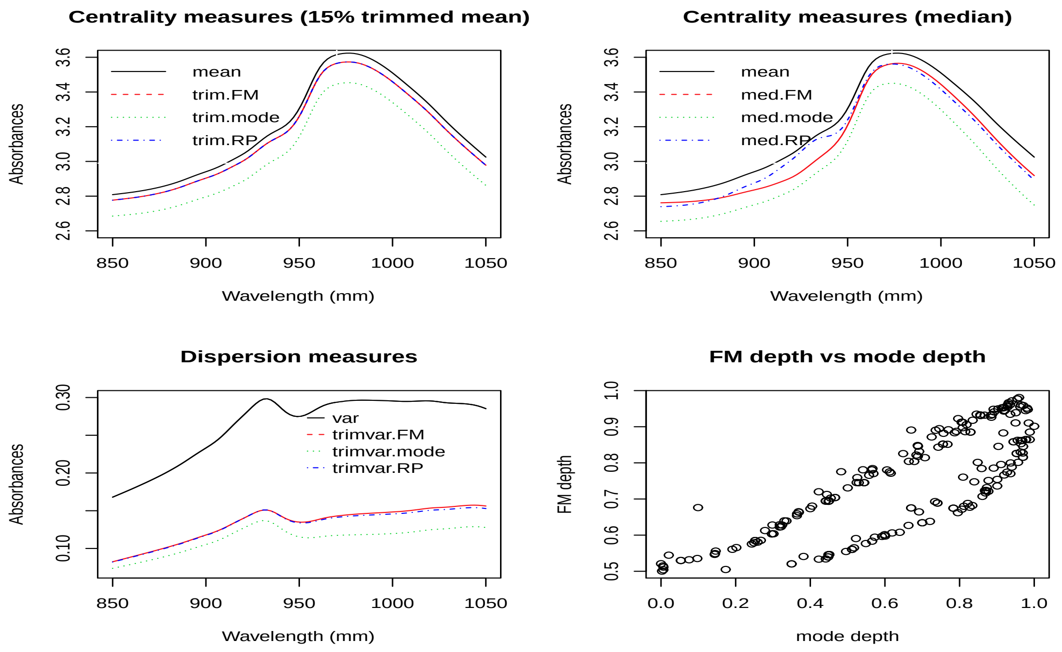

Figure 5.2: Descriptive statistics for Tecator dataset based on depth: 15% trimmed mean (top left), medians (top right), dispersion measures (bottom left) and FM depth versus mode depth (bottom right).

5.2 Bootstrap for functional data

This section presents the smoothed bootstrap procedure as resampling methodology for functional data. Once we obtain our estimates we can represent them using bootstrap techniques that allow us to calculate the confidence bands for statisticians such this procedure fills the holes in the functional space taking into account the covariance structure in the smooth,[Cuevas *et al.* (2006)].

Given the original data $X_1(t),\ldots,X_n(t)$, the bootstrap confidence bands is constructed as follows:

- Calculate the nb bootstrap samples from X: $X_1^*(t),\ldots,X_n^*(t)$, such $X_i^*(t) = X(t)_i + Z(t)$ where $Z(t)$ is normally distributed with mean 0 and covariance matrix $\gamma\Sigma_x$, where Σ_x is the covariance matrix of $(X_1(t),\ldots,X_n(t))$ and γ is the smoothing parameter.
- The estimator $T(X_1^*(t),\ldots,X_n^*(t))$ is defined by calculating the value $D(X_1(t),\ldots,X_n(t))$ such that the $1-\alpha\%$ of the bootstrap replications $T(X_1^*(t),\ldots,X_n^*(t))$ are within a distance from their average smaller than $D(X_1(t),\ldots,X_n(t))$.

The **fdata.bootstrap()** function allows to define an statistic calculated on the **nb** resamples, control the degree of smoothing by γ parameter(**smo** argument) and represent the confidence bands with level α. The **statistic** used by default is the mean **func.mean()** but also other functions based on the depth can be used (see **help(Descriptive)**). The confidence bands are drawn as those resamples which are at some distance from the estimator, see Figure 5.3. Different distances can be used in the construction of this confidence bands, by default the $\mathcal{L}_2$-metric.

```
R> out.boot1=fdata.bootstrap(absorp,statistic=func.mean,nb=1000,draw=TRUE)
R> out.boot2=fdata.bootstrap(absorp,statistic=func.trim.FM,nb=1000,draw=TRUE)
```

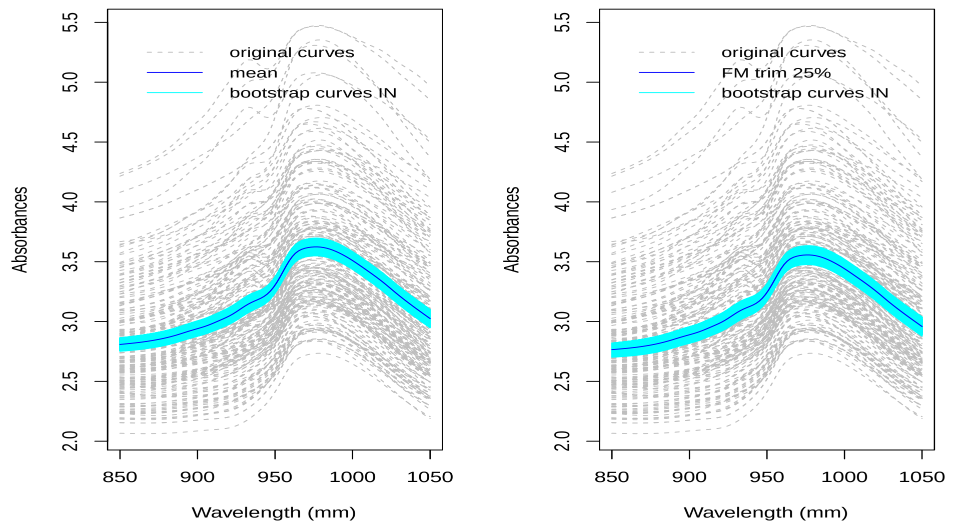

Figure 5.3: Bootstrap replies of spectrometric curves: Using mean statistic (left) and using 25%-trimmed mean statistic with FM depth (right).

5.3 Functional Outlier Detection

Detecting and examining functional outliers is important because they may bias our functional estimates and they may allow to discover which sources produce these outlying curves. In order to identify outliers in functional datasets, [Febrero-Bande *et al.* (2008)] make use that depth and outlyingness are inverse notions, so that if an outlier is in the dataset, the corresponding curve will have a significant low depth. Therefore, a way to detect the presence of functional outliers is to look for curves with lower depths.

The functional outlier detection procedure for detecting outliers is:

Given functional dataset $X_1, \ldots, X_n$:

1. Obtain the functional depths $Dn(X_1), \ldots, D_n(X_n)$, for one of the functional depths defined previously: FM-Depth, Mode-Depth, RP-depth or RP-depth.
2. Let $X_{i_1}, \ldots, X_{i_k}$ be the k curves such that $D_n(X_{i_k}) \leq C$, for a given cutoff C. Then, assume that $X_1, \ldots, X_n$: are outliers and delete them from the sample.
3. Come back to step 1 with the new dataset after deleting the outliers found in step 2. Repeat this until no more outliers are found.

The procedure selects C such that, in the absence of outliers: $Pr(D_n(X_i) \leq C) = c,\ i = 1, \ldots, n$, by default $c = 0.01$. Thus, the C taken is the c-th percentile of the distribution of the functional depth under consideration.

Two procedures for detecting outliers are implemented in the package. The 1st one based on weighting **outliers.depth.pond()** and the 2nd one based on trimming **outliers.depth.trim()**.

(i) Detecting outliers one based on trimming: outliers.depth.trim():

Given functional dataset $X_1, \ldots, X_n$:

1. Obtain the functional depths $Dn(X_1), \ldots, D_n(X_n)$, for one of the functional depths defined above: FM-depth, Mode-depth, RP-depth or RPD-depth.

2. Obtain B standard bootstrap samples X_i^b from the dataset of curves obtained after deleting the $\alpha\%$ less deepest curves, for $i = 1, \ldots, n$ and $b = 1, \ldots, B$.

3. Obtain smoothed bootstrap samples $Y^b = X_i^b + Z_i^b$, where z_i^b is such that $Z_i^b(t_1), \ldots, Z_i^b(t_m)$ is normally distributed with mean 0 and covariance matrix $\gamma\Sigma_x$, where Σ_X is the covariance matrix of $X(t_1), \ldots, X(t_m)$ and γ is a bootstrap smoothing parameter.

4. For each bootstrap set $b = 1, \ldots, B$ obtain C^b as the empirical $c\%$ percentile of the distribution of the depths, $D(Y_i^b)$, $i = 1, \ldots, n$.

5. Take C as the median of the values of C^b, $b = 1, \ldots, B$.

where the estimation of the cutoff C is based on bootstrapping the curves of the original dataset with probability proportional to their depth.

(ii) Detecting outliers one based on weighting: outliers.depth.pond():

1. Obtain the functional depths $Dn(X_1), \ldots, D_n(X_n)$, for one of the functional depths defined above: FM-depth, Mode-depth, RP-depth or RP-depth.

2. Obtain B standard bootstrap samples X_i^b from the curves in which each original curve is sampled with a probability proportional to its depth.

3. Do step 3, 4 an 5 of the bootstrap procedure **(i)**.

The following example we use (**data("poblenou")**) that collects 127 curves of NO_x levels measured every hour $\{t_i\}_{0:23}$ by a control station in Poblenou in Barcelona (Spain). This dataset is used by [Febrero-Bande *et al.* (2008)] as an illustration to outliers detect procedures. Figure 5.4 shows the result of applying the outliers detection method **(i)** based on trimming **outliers.depth.trim** with mode depth for the case of working days and non-working days.

```
R> data("poblenou")
R> nox <- poblenou$nox
R> nb = 20
R> dd <- as.integer(poblenou$df$day.week)
R> working = poblenou$nox[poblenou$df$day.festive == 0 & dd < 6]
R> nonworking = poblenou$nox[poblenou$df$day.festive == 1 | dd > 5]
R> out = outliers.depth.trim(nonworking, dfunc = depth.RP, nb = nb,
+     smo = 0.1, trim = 0.06)
R> out2 = outliers.depth.trim(working, dfunc = depth.FM, nb = nb,
+     smo = 0.1, trim = 0.06)
```

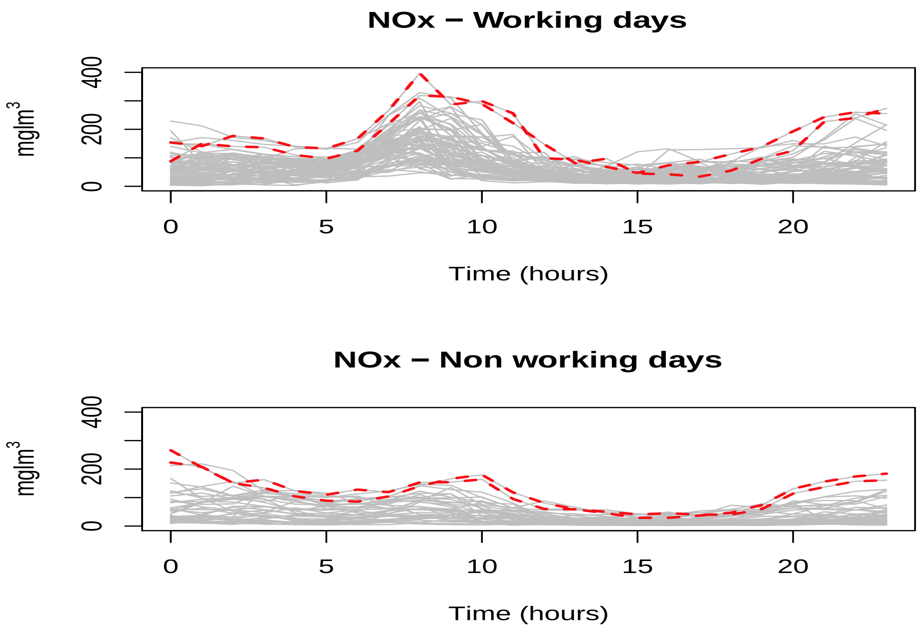

Figure 5.4: NO_x levels measured by a control station split into two groups (gray lines): Working days (top) and non-working days (bottom). Days with NO_x levels detected as outliers (red lines), working days (03/18/2005 and 04/28/2005) and non-working days (03/19/2005 and 04/30/2005).

Chapter 6

Functional regression models (FRM)

Regression models are those techniques for modeling and analyzing the relationship between a dependent variable and one or more independent variables. When one of the variables have a functional nature, we have functional regression models. This section is devoted to all the functional regression models where the response variable is scalar and at least, there is one functional covariate. For illustration, we will use the Tecator dataset to predict the fat contents from the absorbance curves as functional covariate $X(t)$=**X** and/or water contents as a non functional covariate (**Z=Water**). For this set of data is often used the first 129 records as training sample and the last 86 records as test sample for prediction. The explanatory variables to introduce in the models are: The curves of absorbance **X** as functional data or one of its two first derivatives (**X.d1,X.d2**) and/or water content (**Water**) as non functional variable.

```
R> ind = 1:129
R> tt = absorp[["argvals"]]
R> y = tecator$y$Fat[ind]
R> X = absorp[ind, ]
R> X.d1 = fdata.deriv(X, nbasis = 19, nderiv = 1)
R> X.d2 = fdata.deriv(X, nbasis = 19, nderiv = 2)
```

In the following sections, regression methods implemented in the package are presented one by one and illustrated with examples for estimating the **Fat** content of the Tecator dataset. Finally, we show in Section 6.7 a summary of the prediction methods.

6.1 Functional linear model (FLR) with basis representation

This section assumes that the relationship between the scalar response Y and the functional covariate $X(t)$ has a linear structure. Thus, the functional linear model under the parametric approach is given by the expression:

$$y_i = \langle X, \beta \rangle + \epsilon_i = \int_T X_i(t)\beta(t)dt + \epsilon_i \tag{6.1}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product on $\mathcal{L}_2$ and ϵ_i are random errors with mean zero and finite variance σ^2.

[Ramsay and Silverman (2005)] model the relationship between the scalar response and the functional covariate by basis representation of the observed functional data $X(t)$ and the unknown functional parameter $\beta(t)$. The functional linear model in Equation 6.1 is estimated by the expression:

$$\hat{y}_i = \int_T X_i(t)\beta(t)dt \approx \mathbf{C_i^T}\psi(\mathbf{t})\phi^{\mathbf{T}}(\mathbf{t})\hat{\mathbf{b}} = \tilde{\mathbf{X}}\hat{\mathbf{b}} \tag{6.2}$$

where $\tilde{\mathbf{X}}_\mathbf{i}(\mathbf{t})$ is the scores such $\tilde{\mathbf{X}}_\mathbf{i}(\mathbf{t}) = \mathbf{C_i^T}\psi(\mathbf{t})\phi^{\mathbf{T}}(\mathbf{t})$, and $\hat{\mathbf{b}} = (\tilde{\mathbf{X}}^\mathbf{T}\tilde{\mathbf{X}})^{-\mathbf{1}}\tilde{\mathbf{X}}^\mathbf{T}y$ and so, $\hat{y} = \tilde{\mathbf{X}}\hat{\mathbf{b}} = \tilde{\mathbf{X}}(\tilde{\mathbf{X}}^\mathbf{T}\tilde{\mathbf{X}})^{-\mathbf{1}}\tilde{\mathbf{X}}^\mathbf{T}y = \mathbf{H}y$ where $\mathbf{H}$ is the hat matrix with degrees of freedom: $df = trace(\mathbf{H})$.

Many authors, for example [Cardot *et al.* (2003)], [Goldsmith *et al.* (2011)] incorporate a roughness penalty λ (in the so-called Penalized Functional Regression) such the above expression is now: $\hat{y} = \tilde{\mathbf{X}}\hat{\mathbf{b}} = \tilde{\mathbf{X}}(\tilde{\mathbf{X}}^\mathbf{T}\tilde{\mathbf{X}} + \lambda\mathbf{R_0})^{-\mathbf{1}}\tilde{\mathbf{X}}^\mathbf{T}y = \mathbf{H}y$ where $\mathbf{R}_0$ is the penalty matrix.

fregre.basis() function computes functional regression between functional explanatory variable and scalar response using basis representation. This functions is presented as an alternative to the function **fRegress()** of **fda** package because the function allows covariates of class **fdata** and other class format as **matrix** or **data.frame**. The function also gives default values to arguments **basis.x** and **basis.b** for representation on the basis of functional data $X(t)$ and the functional parameter $\beta(t)$, respectively.

Continuing with the example of tecator, the next code illustrates how to estimate the fat contents (**y=Fat**) using a training sample of absorbances curves **X**.

```
R> rangett <- absorp[ind, ]$rangeval
R> basis1 = create.bspline.basis(rangeval = rangett, nbasis = 17)
R> basis2 = create.bspline.basis(rangeval = rangett, nbasis = 7)
R> res.basis0 = fregre.basis(X, y, basis.x = basis1, basis.b = basis2)
```

The fitted object contains useful information as the smoothing parameter (ν), the degrees of freedom (df), the residual variance (S_R^2): $S_R^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 / (n - df)$ (an unbiased estimator of σ^2), the coefficient of determination (R^2) or the hat matrix ($\mathbf{H}$). This information is used in **summary.fregre.fd()** and **print.fregre.fd()** to show summaries of the functional regression fitted model, see Figure 6.1.

```
R> summary(res.basis0)
 *** Summary Functional Data Regression with representation in Basis ***

Call:
fregre.basis(fdataobj = X, y = y, basis.x = basis1, basis.b = basis2)

Residuals:
     Min       1Q   Median       3Q      Max
-10.5079  -1.8778   0.0192   2.5561   5.5249

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  18.2357     0.2883  63.242  < 2e-16 ***
```

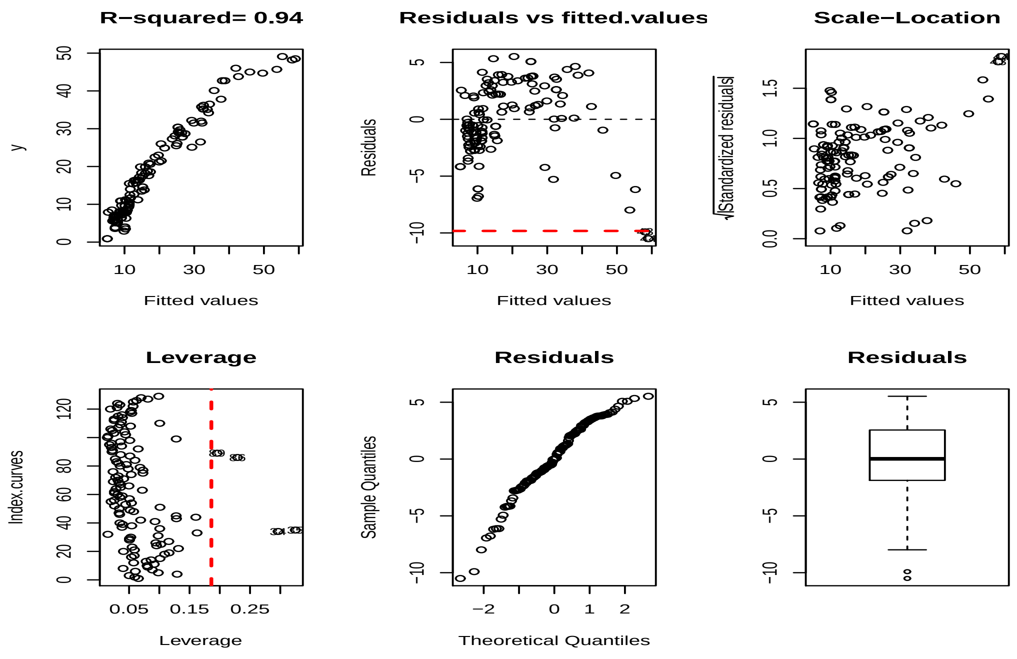

Figure 6.1: Summary plots for fitted object (**res.basis0**) by functional linear model with basis representation (**fregre.basis()**).

```
X.bspl4.1    47.8937     4.7485  10.086  < 2e-16 ***
X.bspl4.2   -53.7862     6.5834  -8.170 3.43e-13 ***
X.bspl4.3    31.7550     7.8395   4.051 9.06e-05 ***
X.bspl4.4   -15.4777     7.1935  -2.152   0.0334 *
X.bspl4.5    16.2149     9.5483   1.698   0.0920 .
X.bspl4.6   -20.5031    11.5959  -1.768   0.0796 .
X.bspl4.7    18.7593     9.2149   2.036   0.0440 *
---
Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1

Residual standard error: 3.275 on 121 degrees of freedom
Multiple R-squared: 0.9367,Adjusted R-squared: 0.9331
F-statistic:   256 on 7 and 121 DF,  p-value: < 2.2e-16

-Names of possible atypical curves: 43 44
-Names of possible influence curves: 34 35 86 89
```

In addition, the function **fregre.basis.cv()** uses validation criteria defined in Section 3.2. In the next example (see figure 6.2), with 33 bspline basis (and 29 fourier basis) gives the minimum value of the validation criterion (by default GCV criteria).

```
R>bb<-seq(9,53,by=4)
R>bsp.cv=fregre.basis.cv(X,y,basis.x=71,basis.b=bb,type.basis="bspline")
R>fou.cv=fregre.basis.cv(X,y,basis.x=71,basis.b=bb,type.basis="fourier")
R>plot(bb,bsp.cv$gcv,type="l",lwd=2,xlab="number of basis",ylab="GCV")
R>lines(bb,fou.cv$gcv,type="l",col=2,lwd=2)
```

```
R> bsp.cv$basis.b.opt$nbasis
[1] 33
R> fou.cv$basis.b.opt$nbasis
[1] 29
```

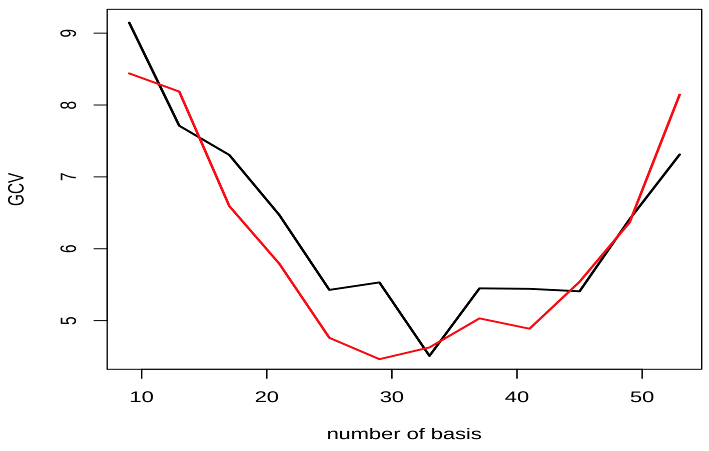

Figure 6.2: GCV criteria

6.2 Functional linear model (FLR) with functional PCA basis

Similarly, [Cardot *et al.* (1999)] used a basis of functional principal components to represent the functional data $X(t)$ and the functional parameter $\beta(t)$ in the so-called functional principal components regression (FPCR).

Now, the estimation of β can be made by a few principal components (PC) of the functional data and the integral can be approximated by:

$$\hat{y}_i = \int_T X_i(t)\beta(t)dt \approx \sum_{k=1}^{k_n} \gamma_{ik_n}\hat{\beta}_{k_n} \tag{6.3}$$

where, $\hat{\beta}_{(1:k_n)} = \left(\frac{\gamma_{.1}^T y}{n\lambda_1}, \ldots, \frac{\gamma_{.k_n}^T y}{n\lambda_{k_n}}\right)$ and $\gamma_{(1:k_n)}$ is the $(n\text{x}k_n)$ matrix with k_n principal components estimation of β scores and λ_i the eigenvalues of the PC.

The model of Equation 6.3 is expressed as: $\hat{y} = \mathbf{H}y$
where $\mathbf{H} = \left(\frac{\gamma_{.1}\gamma_{.1}^T y}{n\lambda_1}, \ldots, \frac{\gamma_{.k_n}\gamma_{.k_n}^T y}{n\lambda_{k_n}}\right)$ with degrees of freedom: $df = trace(\mathbf{H}) = k_n$.

We have implemented the functional principal regression in the function **fregre.pc()**. The call for Tecator example is shown below where the user can select the indexes to estimate coefficients of β, in this case the 1st, 3rd and 6th principal components:

```
R> res.pc0=fregre.pc(X,y,l=c(1,3,6))
```

For the fitted object of **fregre.pc()** function, the function **summary.fregre.fd()** also shows:

- Variability of explicative variables explained by Principal Components.

- Variability for each principal components -PC-.

```
R> summary(res.pc0)
 *** Summary Functional Data Regression with Principal Components ***
Call: fregre.pc(fdataobj = X, y = y, l = c(1, 3, 6))

Residuals:
     Min       1Q   Median       3Q      Max
-22.4864  -3.8617   0.5181   5.2843  13.4575

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  18.2357     0.6517  27.984  < 2e-16 ***
PC1           3.5581     0.5130   6.936 1.92e-10 ***
PC3         -22.5974     1.8046 -12.522  < 2e-16 ***
PC6          83.6575    17.8650   4.683 7.26e-06 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 7.401 on 125 degrees of freedom
Multiple R-squared: 0.6662,     Adjusted R-squared: 0.6582
F-statistic: 83.16 on 3 and 125 DF,  p-value: < 2.2e-16

-With 3 Principal Components is  explained  88.53 %
 of the variability of explicative variables.

-Variability for each  principal components -PC- (%):
  PC1   PC3   PC6
88.19  0.27  0.07
```

6.3 Functional linear model (FLR) with functional PLS basis

In the previous section, the dimension reduction by functional principal components (FPC) is good solution for the estimation of the functional linear model. Another good alternative is uses the criterion that maximizes the covariance between $X(t)$ and the scalar response Y via the PLS components. The basic idea is to construct a set of PLS components in the linear space spanned by $X(t)$, taking into account the correlation between Y and $X(t)$. This FPLS components are more directly related to variability in y than are the FPC components.

Now, the estimation of β can be made by a few partial least squares (PLS factors) of the functional data and the integral can be approximated by:

$$\hat{y}_i = \int_T X_i(t)\beta(t)dt \approx \sum_{j=1}^{q} c_{j,i}\nu_j \tag{6.4}$$

where, $c_{i,j} \in \Re^p$, $i = 1, \ldots, p$ $j = 1, \ldots, q$. and ν is the eigenvector associated to the largest eigenvalue of $W^X W^y$, such $\{\nu_i\}_1^q$ forms an orthogonal system in $\mathcal{L}_2$.

We have implemented the FPLS regression in the function **fregre.pls()** where the PLS factors have been calculated using the **plsr()** function [Mevik and Wehrens (2007)]. The call for Tecator example and the print of the fitted object are shown below:

```
R> res.pc0=fregre.pls(X,y,l=1:6)
R> res.pls0

-Call: fregre.pls(fdataobj = X, y = y, l = 1:6)

-Coefficients:
(Intercept)          PLS1          PLS2          PLS3          PLS4          PLS5
     18.236         4.409        37.519        40.825        33.111        62.422
       PLS6
     34.706

-R squared:  0.9440603
-Residual variance:  9.40637
```

6.3.1 How to select the components?

In FPCA, the dilemma of deciding which components to use in the regression is solved by choosing an optimum subset of the functional PC that best estimated the response. The select is done by cross-validation (CV) or Model Selection Criteria (MSC) and finally, the regression model is fitted using the best selection of functional PC.

- Predictive Cross-Validation: $PCV(k_n) = \frac{1}{n}\sum_{i=1}^n \left(y_i - \left\langle X_i, \hat{\beta}_{(-i,k_n)} \right\rangle\right)^2$, **criteria**="CV"

- Model Selection Criteria: $MSC(k_n) = log\left[\frac{1}{n}\sum_{i=1}^n \left(y_i - \left\langle X_i, \hat{\beta}_{(i,k_n)} \right\rangle\right)^2\right] + p_n\frac{k}{n}$
  $p_n = 2$, **criteria**="AIC"
  $p_n = \frac{2n}{n-k_n-2}$, **criteria**="AICc"
  $p_n = \frac{log(n)}{n}$, **criteria**="SIC"
  $p_n = \frac{log(n)}{n-k_n-2}$, **criteria**="SICc"

where **criteria** is an argument of **fregre.pc.cv()** and **fregre.pls.cv()** function that controls the type of validation used in the selection of the smoothing parameter k_n.

For above example, the result is:

```
R> res.pc2 = fregre.pc.cv(X.d2, y, kmax = 8)
R> res.pc2$pc.opt
Var1 Var2 Var3 Var4 Var5 Var6
   1    4    2    8    6    5
R> res.pc2$MSC.min
```

```
[1] 2.136051
R> res.pc2$MSC
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]     [,8]
[1,] 2.874226 2.395084 2.140198 2.138155 2.136051 2.147605 2.184047 2.221696
attr(,"names")
[1] "PC1" "PC2" "PC7" "PC3" "PC6" "PC8" "PC4" "PC5"
```

But, as in functional PLS regression each component is obtained by maximizing the covariance between $X(t)$ and y we can select directly the first functional PLS by cross-validation (CV) or Model Selection Criteria (MSC), instead of using the best combination of the components. In the example below are selected the three PLS components from among the 8 PLS components using the criterion "SICc".

```
R> res.pls=fregre.pls.cv(X.d1,y,kmax=8,criteria="SICc")
R> res.pls$pls.opt
[1] 1 2 3
R> res.pls$MSC
    PLS1     PLS2     PLS3     PLS4     PLS5     PLS6     PLS7     PLS8
3.022503 2.951925 2.379544 2.405675 2.392936 2.411932 2.396016 2.404884
```

6.3.2 Functional influence measures

This section is devoted to the procedures discussed by [Febrero-Bande *et al.* (2010)] who studied how to identify influential observations in the FLM seen in the previous sections (Section 3.1 3.2 and 3.3). Three statistics are introduced for measuring the influence: Distance Cook Prediction (CP_i), Distance Cook Estimation (CE_i) and Peña Distance (P_i), respectively.

1. The functional Cook's measure for prediction (CP_i) allows to detect observations which their deletion may imply important changes in prediction of the rest of the data and is defined as follows:

$$CP_i = \frac{(\hat{y} - \hat{y}_{-i})^T (\hat{y} - \hat{y}_{-i})}{S_R^2}$$

   where $\hat{y}_{-i}$ is the prediction of the response y excluding the i-th observation (X_i, y_i) in the estimation.

2. The functional Cook's measure for estimation (CE_i) allows to detect observations which their deletion may imply important changes in estimation.

$$CE_i = \frac{\left\| \hat{\beta} - \hat{\beta}_{-i} \right\|^2}{\frac{S_R^2}{n} \sum_{k=1}^{k_x} \frac{1}{\lambda_k}}$$

   where $\hat{\beta}_{-i}$ is the estimation of the parameter β excluding the i-th observation (X_i, y_i) in the process.

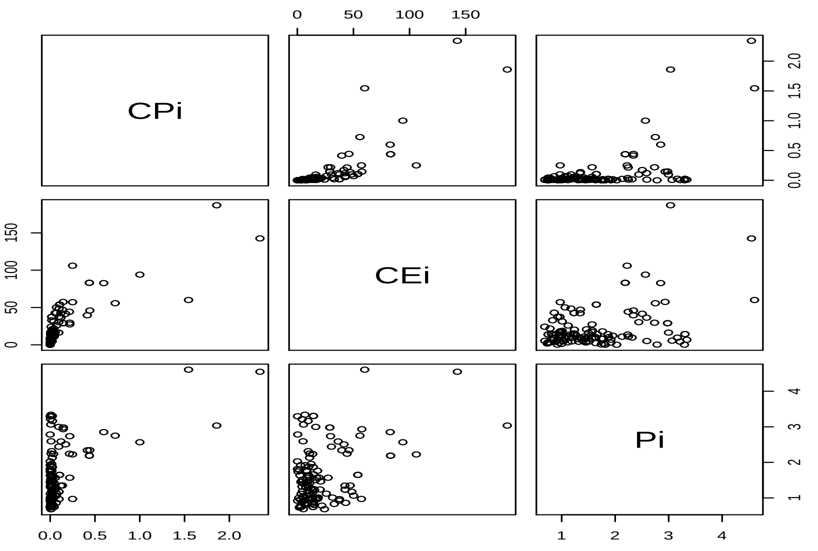

Figure 6.3: Influence measure for Tecator dataset calculated from fitted model (**res.basis0**): Matrix of scatterplots for Distance Cook Prediction (CP_i), Distance Cook Estimation (CE_i) and Peña Distance (P_i).

3. The functional Peña's measure for prediction (P_i) allows to detect observations which the prediction is most affected by the deletion of other data.

$$P_i = \frac{(\hat{y}_i - \hat{y}_{(-1,i)}, ..., \hat{y}_i - \hat{y}_{(-n,i)})^T (\hat{y}_i - \hat{y}_{(-1,i)}, ..., \hat{y}_i - \hat{y}_{(-n,i)})}{S_R^2 H_{ii}}$$

where $\hat{y}_{(-h,i)}$ is the i-th component of the prediction vector $\hat{\mathbf{y}}_{(-h)}$ for $h = 1, ..., n$.

Once estimated the functional regression model with scalar response (by **fregre.pc()** or **fregre.basis()**), **influence.fdata()** function is used to obtain the influence measures, see Figure 6.3. Below are the names of the four curves with high measure of influence. The curve 44 has a high value for all measures and is also detected as a potential atypical curve, but it has not a high leverage value. The indices 34, 35 and 44 correspond to curves with high influence measures both in estimation and prediction.

```
R>  order(res.infl$DCE)[125:129]
[1] 51 34 10 44 35
R>  order(res.infl$DCP)[125:129]
[1] 129  34  43  35  44
R>  order(res.infl$DP)[125:129]
[1]  40 123  42  44  43
R>  summary.fregre.fd(res.basis0)$i.atypical
-Names of possible atypical curves: 43 44
R>  summary.fregre.fd(res.basis0)$i.influence
-Names of possible influence curves: 34 35 86 89  #Leverage
```

[Febrero-Bande *et al.* (2010)] propose to approximate quantiles of the above statistics by means of a procedure which uses smoothed bootstrap samples of the set of observations of the above statistics. This package includes the above procedure in **influence.quan()** function. When the goal is to make inferences about the functional parameter and not on influence measures, one of the gain derived from this procedure is that the calculation takes the **mue.boot** curves of the functional parameter β in the functional Cook's measure for estimation. However, this procedure has a very high computation time. In order to reduce the computational time we have created the **fregre.bootstrap()** function presented in the next section.

```
R> res.infl=influence.fdata(res.basis0)
```

6.3.3 Bootstrap for Functional linear model (FLR)

For functional regression models based on parametric representation, principal components or partial least squared can study the behavior of functional beta parameter β by bootstrap procedure. To do this, we follow the smoothed bootstrap procedure proposed by [Febrero-Bande *et al.* (2010)].

Given a n scalar responses Y and a n functional data X.

1. Fit the functional linear model in Equation 6.1, and compute the residuals e.
2. Obtain B standard bootstrap samples of size n of the residuals: $e_1^b, \ldots, e_n^b$, for $b = 1, \ldots, B$. Similarly, obtain B standard bootstrap samples of the functional data: $X_1^b, \ldots, X_n^b$.
3. Smooth the two previous samples as follows:

   (i). Obtain $\tilde{X}_i^b = X_i^b + Z_i^b$ such Z_i^b is a Gaussian process with zero mean and the covariance operator $\gamma_X \Gamma_X$, where $\gamma_X$is a smoothed bootstrap parameter.
   (ii). Obtain $\tilde{e}_i^b = e_i^b + Z_i^b$ such z_i^b is a normally distributed with mean 0 and variance $\gamma_e S_R^2$, where γ_e is a smoothed bootstrap parameter.
4. Obtain: $\tilde{y}_i^b = \left\langle \tilde{X}_i^b, \tilde{\beta}_\nu \right\rangle + \tilde{e}_i^b$ for $b = 1, \ldots, B$, $i = 1, \ldots, n$, where $\tilde{\beta}_\nu$ is the estimate ob β obtained in step 1.
5. Fit the model 6.1 to the smoothed bootstrap $\tilde{Y}^b$ and $\tilde{X}^b$.
6. The α bootstrap confidence based on $\hat{\beta}_\nu$ is defined by calculating the value D_α such that the $100\alpha\%$ of the bootstrap replications $\hat{\beta}_\nu^b$ are within a distance from $\hat{\beta}_\nu$ smaller than D_α.
7. A confidence based of a functional parameter $\hat{\beta}_\nu$ at the confidence level $\alpha\%$ is the set of bootstrap estimates of functional parameter $\hat{\beta}_\nu^b$ such that:
$$CB(\hat{\beta}_\nu) = \left\{ \hat{\beta}_\nu^b : \left\| \hat{\beta}_\nu - \hat{\beta}_\nu^b \right\| < D_\alpha \right\}$$
   where D_α is such that $P(CB(\hat{\beta}_\nu)) = \alpha$.

In the function **fregre.boot()**, the smoothed bootstrap parameters γ_X and γ_e are the arguments **smoX** and **smo**, respectively. In step 4, the argument **fix.knn** can be selected by an appropriated cutoff ν in each b iteration or fix its value in the step 1.

In Figure 6.4 are plotting the bootstrap confidence band for the $\hat{\beta}$ (blue line) with level of $(1-\alpha)\%$. For the three fitted models (basis, PC and PLS), with 100 $\hat{\beta}^*$ curves, 99 curves belonging to the confidence band are drawn in gray and the curve fell outside the band is drawn in red. The effect of β is significative (different from 0) for different wavelength, although these values are different depending on model.

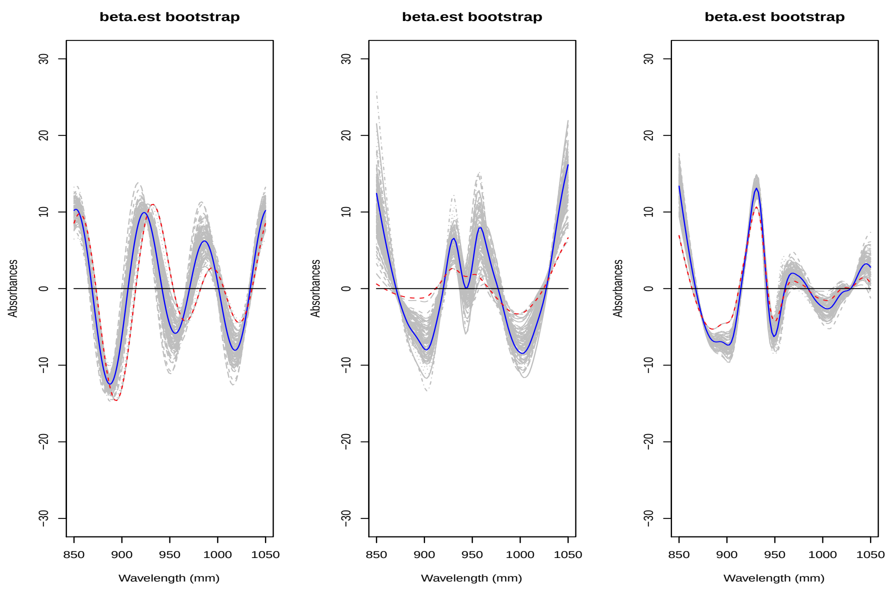

Figure 6.4: Estimated regression function $\hat{\beta}$ joint with a 99% bootstrap confidence band by: fregre.basis (left), fregre.pc (midle) and fregre.pls (right).

```
R>fregre.bootstrap(res.basis1,nb=100,kmax.fix=TRUE,alpha=.99)
R>fregre.bootstrap(res.pc0,nb=100,kmax.fix=TRUE,alpha=.99)
R>fregre.bootstrap(res.pls0,nb=100,kmax.fix=TRUE,alpha=.99)
```

6.4 FLR with functional and non functional covariate

This section is presented as an extension of the previous linear regression models. Now, the scalar response Y is estimated by more than one functional covariate $X^j(t)$ and also more than one non functional covariate Z^j. The regression model is given by:

$$y_i = \alpha + \beta_1 Z_i^1 + \cdots + \beta_p Z_i^p + \int_{T_1} X_i^1(t)\beta_1(t)dt + \cdots + \int_{T_f} X_i^q(t)\beta_q(t)dt + \epsilon_i \tag{6.5}$$

where $Z = \left[Z^1, \cdots, Z^p\right]$ are the non functional covariates and $X(t) = \left[X^1(t_1), \cdots, X^q(t_q)\right]$ are the functional covariates.

The functional linear model 6.5 is estimated by the expression:

$$\hat{y} = \tilde{\mathbf{X}}\mathbf{b} = \tilde{\mathbf{X}}(\tilde{\mathbf{X}}^{\mathbf{T}}\tilde{\mathbf{X}})^{-\mathbf{1}}\tilde{\mathbf{X}}^{\mathbf{T}}y = \mathbf{H}y$$

where the first columns of $\tilde{X}$ are the p non-functional covariates Z and the following columns are the q scores. This scores can be done by:

(1) basis expansion (of class "fd") analogous to equation 6.2:

$$\tilde{X} = \left[Z^1, \cdots, Z^p, (\mathbf{C^1})^{\mathbf{T}}\psi(\mathbf{t_1})\phi^{\mathbf{T}}(\mathbf{t_1}), \cdots, (\mathbf{C^q})^{\mathbf{T}}\psi(\mathbf{t_q})\phi^{\mathbf{T}}(\mathbf{t_q})\right]$$

(2) functional principal components basis as in equation 6.3:

$$\tilde{X} = \left[Z^1, \cdots, Z^p, \{f_{i1}^1, \ldots, f_{ik_1}^1\}, \cdots, \left\{f_{i1}^q, \ldots, f_{ik_q}^q\right\}\right]$$

The arguments are as follows:

- **formula**: A symbolic description of the model to be fitted.
- **data**: List that containing the variables in the model. The first item in the data list is a *"data.frame"* called **df** with the response and non functional explanatory covariates. Functional covariates (*"fdata"* or *"fd"* class) are introduced in the following items in the data list.
- **basis.x**: List with a basis object for every functional covariate.
- **basis.b**: List with a basis object for estimate the functional parameter β.

For Tecator data example, the content of **Fat** is estimated from the second derivative of absorbance curves **X.d2** and the content of **Water** by **fregre.lm()** function.

```
R> ind <- 1:129
R> dataf = as.data.frame(tecator$y[ind, ])
R> newdataf = as.data.frame(tecator$y[-ind, ])
R> ldata = list(df = dataf, X = X, X.d1 = X.d1, X.d2 = X.d2)
R> f2 = Fat ~ Water + X.d2
R> basis.x1 = list(X.d2 = basis1)
R> basis.b1 = list(X.d2 = basis2)
R> res.lm2 = fregre.lm(f2, ldata, basis.x = basis.x1, basis.b = basis.b1)
[1] "Non functional covariate: Water"
[1] "Functional covariate: X.d2"
```

For illustration, the fitted object returned (**res.lm2**) can be used in other functions of the *"lm"* class such as: **summary()**, **coefficients()** or **predict()**, among other.

```
R> round(coefficients(res.lm2),4)
 (Intercept)         Water X.d2.bspl4.1 X.d2.bspl4.2
     78.0349       -0.9472    -428.7644    3006.3576
X.d2.bspl4.3 X.d2.bspl4.4 X.d2.bspl4.5
  -3329.0297    2445.4036   -1513.1237
R> summary(res.lm2)

Call: lm(formula = pf, data = XX, x = TRUE, y = TRUE)

Residuals:
    Min      1Q  Median      3Q     Max
-4.2750 -0.9800  0.2044  1.2410  2.6524

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)     78.0349     3.0395  25.673  < 2e-16 ***
```

```
Water           -0.9472     0.0481 -19.694  < 2e-16 ***
X.d2.bspl4.1  -428.7644   736.7328  -0.582 0.561653
X.d2.bspl4.2  3006.3576   793.6162   3.788 0.000237 ***
X.d2.bspl4.3 -3329.0297   839.9824  -3.963 0.000125 ***
X.d2.bspl4.4  2445.4036   551.9874   4.430 2.07e-05 ***
X.d2.bspl4.5 -1513.1237   425.2327  -3.558 0.000533 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 1.553 on 122 degrees of freedom
Multiple R-squared: 0.9857,     Adjusted R-squared: 0.985
F-statistic:  1397 on 6 and 122 DF,  p-value: < 2.2e-16
```

6.5 Nonparametric functional regression model

An alternative of functional linear models is the nonparametric functional regression studied by [Ferraty and Vieu (2006)]. In this case, the regression model is written as:

$$y_i = r(X_i(t)) + \epsilon_i \tag{6.6}$$

where the unknown smooth real function r is estimated using kernel estimation by means of:

$$\hat{r}(X) = \frac{\sum_{i=1}^{n} K(h^{-1}d(X, X_i))y_i}{\sum_{i=1}^{n} K(h^{-1}d(X, X_i))}$$

where K is an asymmetric kernel function, h is the smoothing parameter and d is a metric or a semi-metric.

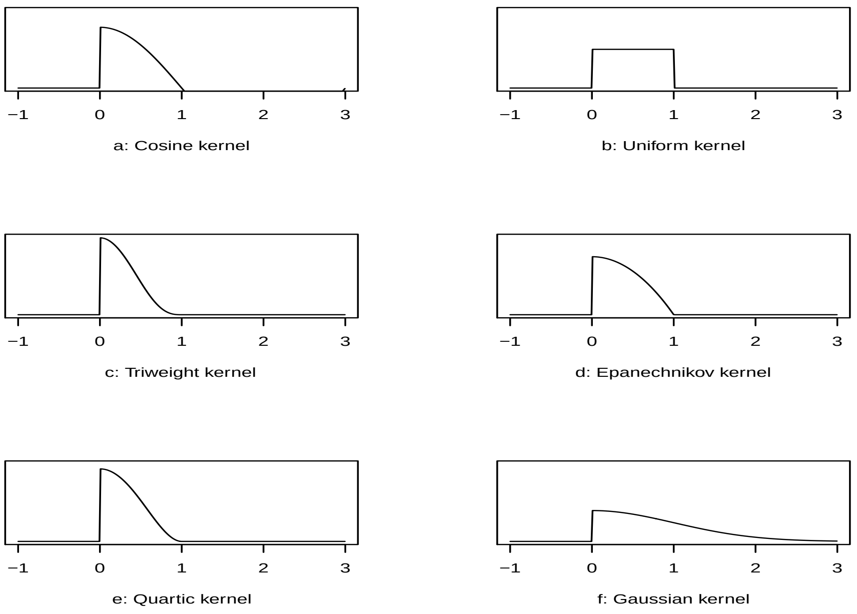

Figure 6.5: Asymmetric Kernels

Now the kernel is applied to a metric or semi-metric d that provides positive values. The figure 6.5 are displaying the asymmetrical kernels included in the package.

The functional non-parametric regression is implemented in the **fregre.np()** function and some of its arguments are:

- **h**: Smoothing parameter or bandwidth.
- **Ker**: Type of asymmetric kernel function, by default asymmetric normal kernel.
- **metric**: Type of metric or semi-metric, by default $\mathcal{L}_2$ (**metric.lp(...,p=2)**).
- **type.S**: Type of smoothing matrix S, by default Nadaraya-Watson (**S.NW()**).

The code for the Tecator example is:

```
R> fregre.np(X, y, metric = semimetric.deriv, nderiv = 1)
-Call: fregre.np(fdataobj = X, y = y, metric = semimetric.deriv, nderiv = 1)

-Bandwidth (h):  0.07296407
-R squared:  0.985607
-Residual variance:  3.810363
```

Again, it has also implemented the function **fregre.np.cv()** to estimate the smoothing parameter h by the validation criteria described in Section 3.3.

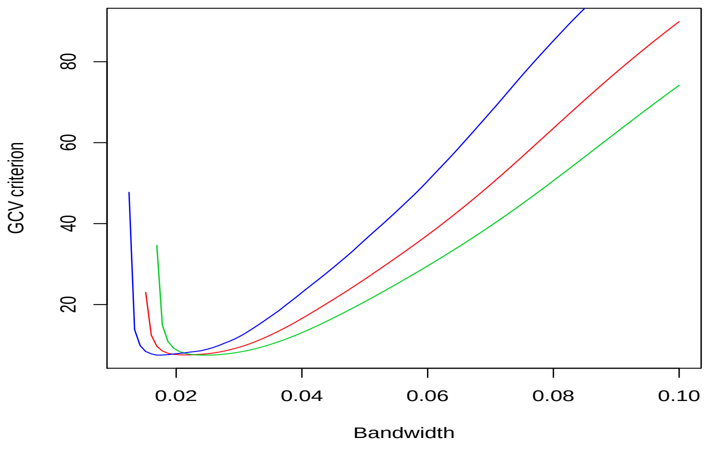

Figure 6.6: GCV criteria as a function of bandwidth choice: Quartic kernel (red line), Triweight kernel (green line) and Epanechnikov kernel (blue line).

- **type.CV**: Type of validation criteria, by default GCV criteria (**GCV.S()**).

```
R>  h<-seq(0.0125,0.1,len=100)
R>  np1=fregre.np.cv(X.d1,y,Ker=AKer.quar,h=h)
R>  plot(h,np1$gcv,col=2,type="l",ylab="GCV criterion",xlab="Bandwidth")
R>  np2=fregre.np.cv(X.d1,y,Ker=AKer.tri,h=h)
R>  lines(h,np2$gcv,col=3)
R>  np3=fregre.np.cv(X.d1,y,Ker=AKer.epa,h=h)
R>  lines(h,np3$gcv,col=4)
```

6.6 Semi-functional partially linear model (SFPLM)

An extension for the non-parametric functional regression models is the semi-functional partial linear model proposed in [Aneiros-Pérez and Vieu (2006)]. This model allows the estimation of the scalar response y for non functional covariates $\vec{Z}$ via linear terms and a functional covariate $X(t)$ via non-parametric kernel estimation described in Section 6.5.

$$y_i = f(\vec{Z}_i, X_i(t)) + \epsilon_i = f(Z_i^1, \ldots, Z_i^p, X(t)) + \epsilon_i = \sum_{j=1}^{p} Z_{ij}\beta_j + r(X_i(t)) + \epsilon_i \tag{6.7}$$

where r is an unknown smooth real function and the errors ϵ are independent, with zero mean, finite variance σ^2 and $E[\epsilon X(t)] = 0$.

The unknown parameters β for $\vec{Z}$ covariates are estimated by means of:

$$\hat{\beta}_j = (\tilde{X}_h^T \tilde{X}_h)^{-1} \tilde{X}_h^T \tilde{X}_h$$

where h is the smoothing parameter.

The unknown smooth real function r is estimated by means of:

$$\hat{r}_h(t) = \sum_{i=1}^{n} w_{n,h}(t, X_i)(y_i - \vec{Z}_i^T \hat{\beta}_h)$$

where W_h is a weight function: $w_{n,h}(t, X_i) = \frac{K(d(t,X_i)/h)}{\sum_{j=1}^{n} K(d(t,X_j)/h)}$ with an asymmetric kernel K and metric or semimetric d.

In **fregre.plm()** by default W_h is a functional version of the Nadaraya-Watson-type weights (**type.S=S.NW**) with asymmetric normal kernel (**Ker=AKer.norm**) in $\mathcal{L}_2$ space (**metric= metric.lp** with **p=2**).

Continuing with the example of Section 6.4, the fitted model to the case of a real variable **Z=Water** and the second derivative of the absorbance curves (**X.d2**) as functional covariate:

```
R> fregre.plm(f2, ldata, Ker = AKer.epa, type.S = S.KNN)
-Call: fregre.plm(formula = f2, data = ldata, Ker = AKer.epa, type.S = S.KNN)

-Coefficients:
       Estimate    Std. Error  t value     Pr(>|t|)
Water  -9.634e-01   3.748e-02  -2.571e+01   3.018e-24

-Bandwidth (h):  9
-R squared:  0.994232
-Residual variance:  1.256837
```

6.7 Predict methods for functional regression model fits

Once as the model is estimated we can obtain predictions from a fitted functional regression model object by means of:

- **predict.fregre.fd()** for fitted model from the function: **fregre.basis()**, **fregre.pc()**, **fregre.pls()** or **fregre.np()**.
- **predict.fregre.lm()** for fitted model from the function: **fregre.lm()**.
- **predict.fregre.plm()** for fitted model from the function: **fregre.plm()**.

A sample test of last 86 curves of absorbance (or one of the two first derivatives) are used to predict the **Fat** content, see next code:

```
R> newy = matrix(tecator$y$Fat[-ind], ncol = 1)
R> newX = absorp[-ind, ]
R> newX.d1 = fdata.deriv(absorp[-ind, ], nbasis = 19, nderiv = 1)
R> newX.d2 = fdata.deriv(absorp[-ind, ], nbasis = 19, nderiv = 2)
R> res.basis2 = fregre.basis.cv(X.d2, y, type.basis = "fourier")
R> pred.basis2 = predict.fregre.fd(res.basis2, newX.d2)
R> res.pc1 = fregre.pc.cv(X.d1, y, 8)$fregre.pc
R> pred.pc1 = predict.fregre.fd(res.pc1, newX.d1)
R> res.np2 = fregre.np.cv(X.d2, y, metric = semimetric.fourier)
R> pred.np2 = predict(res.np2, newX.d2)
```

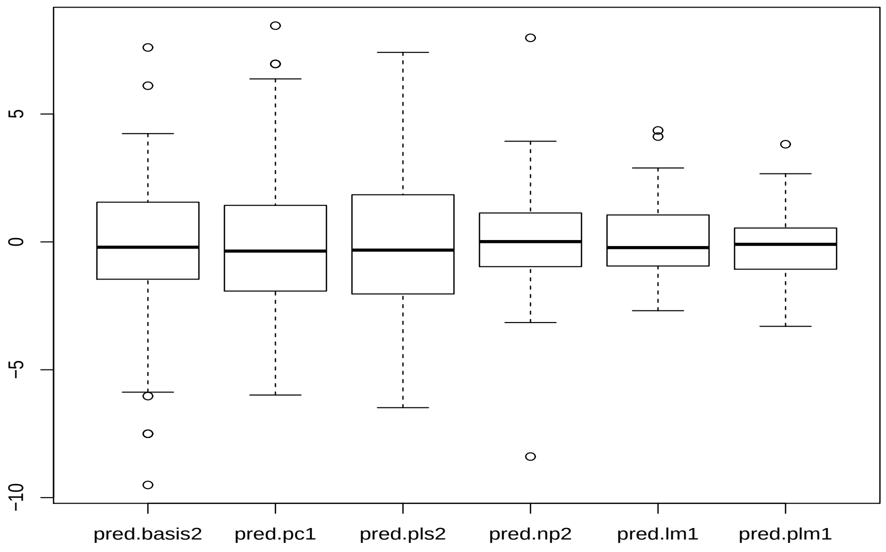

Figure 6.7: Boxplot of predicted residuals: **fregre.basis()**, **fregre.pc()**, **fregre.pls()**, **fregre.np()**, **fregre.lm()** and **fregre.plm()** (left to right).

More interestingly, it can be to predict the **Fat** content by means of a functional covariate and other non functional, as for example the **Water**. We provide below the complete code for the best prediction of each procedure developed: **fregre.lm()** and **fregre.plm()**.

```
R> newldata = list(df = newdataf, X = newX, X.d1 = newX.d1, X.d2 = newX.d2)
R> f1 = Fat ~ Water + X.d1
R> basis.x1 = list(X.d1 = basis1)
R> basis.b1 = list(X.d1 = basis2)
R> res.lm1 = fregre.lm(f1, ldata, basis.x = basis.x1, basis.b = basis.b1)
[1] "Non functional covariate: Water"
[1] "Functional covariate: X.d1"
R> pred.lm1 = predict.fregre.lm(res.lm1, newldata)
R> res.plm1 = fregre.plm(f1, ldata, Ker = AKer.tri, type.S = S.KNN)
R> pred.plm1 = predict.fregre.plm(res.plm1, newldata)
```

The boxplot of predicted residuals by each procedure are shown in Figure 6.7.

We make predictions for some of fitted models in this document (see all code in annex). Following the idea work in [Aneiros-Pérez and Vieu (2006)], we calculated the mean quadratic error of prediction (MEP): $MEP = \left(\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 /n\right) / (Var(y))$, which is used for comparing the predictions of different fitted models. Table 6.1 resumes the statistics of the fitted models and their predictions.

Function	*df*	R^2	S_R^2	MEP
lm(Fat ~ Water)	2	0.975	4.076	0.0213
fregre.basis(X,Fat)	8	0.937	10.726	0.0544
fregre.basis.cv(X.d2,Fat)	12	0.962	6.613	0.0485
fregre.pc(X.d1,Fat)	7	0.947	8.928	0.0502
fregre.pc(X.d2,Fat)	7	0.943	9.626	0.0521
fregre.pls(X.d1,Fat)	7	0.947	8.884	0.0502
fregre.pls(X.d2,Fat)	6	0.955	7.537	0.0422
fregre.lm(Fat ~ X.d1+Water)	9	0.987	2.149	0.0096
fregre.lm(Fat ~ X.d2+Water)	7	0.986	2.412	0.0119
fregre.np(X.d1,Fat)	39.4	0.981	4.239	0.0287
fregre.np(X.d2,Fat)	39.9	0.990	2.361	0.0243
fregre.plm(Fat ~ X+Water)	20.0	0.984	3.049	0.0178
fregre.plm(Fat ~ X.d1+Water)	38.5	0.994	1.314	0.0093
fregre.plm(Fat ~ X.d2+Water)	36.3	0.995	1.107	0.0114

Table 6.1: Results for functional regression models. *df* degrees of freedom, S_R^2 residual variance, R^2 R-squared and MEP mean quadratic error of prediction.

Regression methods with a functional covariate (fregre.basis, fregre.pc and fregre.pls) fit and predict correctly the fat content from absorbance curves (MEP=0.0485, MEP=0.0502 and MEP=0.0422 respectively), and has the advantage of being able to estimate the functional parameter β. However, if our goal is the only prediction of response gives a new curve, the method of functional non-parametric regression (fregre.np) has better prediction (a lower MEP, MEP=0.243) than the rest, but with more degrees of freedom. This last result is the closest to that obtained by fitting the linear model (using **lm()** function) with $Water$ as the only covariate, (MEP = 0.0213). If we take this linear model (not functional) as a reference, the prediction error with $Water$ and the curves X using partial linear model is reduced only to 84% (MEP = 0.0178). However if you include the first derivative $X.d1$, the gain is greater, reducing the error to 43% (MEP = 0.0093). It is clear that the prediction improves when a non-functional covariate is included in the model, but both models parametric (fregre.lm) and semi-parametric (fregre.plm) predict better using one of the first two derivatives of the curves of absorbance than the curves without deriving.

Chapter 7

Generalized Functional Linear Model (GFLM)

In several applications for instance when the response is binary the functional linear model (FLM) seen in section 6.4 may be too restrictive. One natural extension of this model is the generalized functional linear regression model (GFLM) [Müller and Stadtmüller (2005)] which allows various types of the response nd its expected value is related to this linear predictor via a link function. For example, with this approach in the case of a count data or binary variable would have the functional poisson or binomial regression, respectively.

In the GLM framework it is generally assumed that $y_i|x_i$ can be chosen within the set of distributions belonging to the exponential family with probability density function:

$$f(\theta,\phi,y) = exp\left\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi)\right\} \tag{7.1}$$

where ϕ represents a scale (or dispersion) parameter and θ is the canonical parameter of distribution. The functions $a()$, $b()$ and $c()$ are known and differ for the distinct Y distributions, e.g., the normal, binomial or poisson distribution.

The estimation of the model parameters should be carried out by maximizing the likelihood function. The log-likelihood is:

$$l(\theta,\phi,y) = logf(\theta,\phi,y) = \frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi) \tag{7.2}$$

The model is specified as follows:

$$E[y|X] = b'(\theta) = \mu$$

$$Var[y|X] = b''(\theta)a(\phi) = V(\mu)\phi$$

$$g(\mu) = \left(\int_T X\beta + dt\right) + Z\beta$$

where μ is the expected value of response, $g()$ is the link function that specified the dependence between μ and the regressors, $V[\mu]$ is the conditional variance.

A very flexible class of link functions can be defined. In **R** is specified some of the principal distribution together with the link function (see next table).

Distribution	ψ	$E(\mu)$	$V(\mu)$	Canonical link; in **R**
Binomial/n	1/n	μ	$\mu(1-\mu)$	$log(\mu(1-\mu))$; **logit**
Poisson	1	μ	μ	$log(\mu)$; **log**
Negative Binomial	1	$log\left(\frac{\mu}{1+1/\phi}\right)$	$\mu+\frac{\mu^2}{\phi}$	$log\left(\mu(\phi+\mu)\right)$; **log**
Normal	σ^2	μ	1	μ; **identity**
Gamma	$1/\upsilon$	$-1/\upsilon$	μ^2	μ^{-1}; **inverse**

Table 7.1: Principal distributions used in GLMs.

7.1 GFLM estimation

7.1.1 Inference for β.

The dependence of the scalar response Y is estimated by more than one functional covariate $X^j(t)$ and also more than one non functional covariate Z^j via link functions.

The model is given by:

$$y_i = g^{-1}\left(\alpha+\beta_1 Z_i^1+\cdots+\beta_p Z_i^p+\int_{T_1} X_i^1(t)\beta_1(t)dt+\cdots+\int_{T_f} X_i^q(t)\beta_q(t)dt\right)+\epsilon_i \tag{7.3}$$

where $Z = \left[Z^1,\cdots,Z^p\right]$ are the non functional covariates, $X(t) = \left[X^1(t_1),\cdots,X^q(t_q)\right]$ are the functional covariates and ϵ_i are random errors with mean zero and finite variance σ^2.

The functional generalized linear model 7.3 is estimated by the expression:

$$\hat{y} = g^{-1}\left(\tilde{\mathbf{X}}\beta\right) = g^{-1}\left(\tilde{\mathbf{X}}(\tilde{\mathbf{X}}^{\mathbf{T}}\tilde{\mathbf{X}})^{-\mathbf{1}}\tilde{\mathbf{X}}^{\mathbf{T}}\right)y = g^{-1}\left(\mathbf{H}\right)y$$

where the first columns of $\tilde{X}$ are the p non-functional covariates Z and the following columns are the q scores. This scores can be done by:

(1) basis expansion (of class "fd") analogous to equation 6.2:

$$\tilde{X} = \left[Z^1,\cdots,Z^p,(\mathbf{C^1})^{\mathbf{T}}\psi(\mathbf{t_1})\phi^{\mathbf{T}}(\mathbf{t_1}),\cdots,(\mathbf{C^q})^{\mathbf{T}}\psi(\mathbf{t_q})\phi^{\mathbf{T}}(\mathbf{t_q})\right]$$

(2) functional principal components basis as in equation 6.3:

$$\tilde{X} = \left[Z^1,\cdots,Z^p,\left\{f_{i1}^1,\ldots,f_{ik_1}^1\right\},\cdots,\left\{f_{i1}^q,\ldots,f_{ik_q}^q\right\}\right]$$

Maximum likelihood estimates of β can be obtained via iteratively weighted least squares (IWLS) algorithm, for a more complete description see [McCullagh and Nelder (1989)]. IWLS consists in:

1. Form the working responses z and the working weights W:
   $z_j^{(t)} = \nu_i^{(t)} + (y_i-\mu_i^{(t)})\left(\frac{d\nu}{d\mu}\right)$ where $\nu_i^{(t)} = x_i\hat{\beta}^{(t-1)}$ and $\mu_i^{(t)} = g^{-1}(\nu_i^{(t)})$.

   $W_j^{(t)} = \left[\left(\frac{d\nu}{d\mu}\right)\right]V_i^{(t)}]^{-1}$ where $V_i^{(t)} = V_i^{(t)}(\mu_i^{(t)})$ is referred to as the variance function.

   Initial values for z_0 and W_0 can be calculated from the initial linear predictor.

2. Run the weighted regression of the $z_j^{(t)}$ on the covariates $\tilde{x}_i$ with weights $W_j^{(t)}$; designate the coefficients from this weighted regression as $\hat{\beta}(t)$ and proceed to the next iteration.
3. Repeat step 1 and 2 until convergence in $\hat{\beta}(t)$ or the log-likelihood. The asymptotic variance of $V(\hat{\beta})$ is given by $V(\hat{\beta}) = \hat{\varphi}(\tilde{X}'W\tilde{X})^{-1}$ where $W = diag(W_1, \ldots, W_n)$ computed at the final iteration.

Asymptotically-valid standard errors for the coefficients are obtained by taking the square root of the leading diagonal of $V(\hat{\beta})$.

7.1.2 Deviance

The deviance is often used to validate the model that is calculated as:

$$D = 2(l(y;y) - l(y;\mu))$$

where l(y;y) is the likelihood under the full model and $l(y;\mu)$ is the likelihood under the fitted model.

When the parameter of dispersion is known, the scaled deviance is define as:
$D^* = D/\psi$ which is approximately distributed as a χ^2_{n-p}.

7.1.3 Overdispersion

The dispersion parameter is usually estimated after computing residuals with $\hat{\beta}$ from:

$$\hat{\phi} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - \hat{\mu}_i}{V(\hat{\mu}_i)}\right)$$

We focus on the case of discrete-response model, the data may be overdispersed if the estimated dispersion parameter is greater than expected, $V(y) = \psi E(y)$, ie, the variance is larger than the mean. If Y_i is over-dispersed the standard errors of the parameter estimates can be underestimating and we could lead to incorrect conclusions. Three solutions are often used in practice, (1) adjust the standard errors by the dispersion parameter, (2) leave the dispersion parameter not fixed at one $\phi \neq 1$ (see **quasi()** families in **R** as **quasipoisson** family) and (3) model mean and dispersion jointly using a negative binomial model (**family = negative.binomial()**).

7.1.4 Model Validation

different types of residuals may be selected in **R** [Venables and Ripley (2002)]: **"deviance"** (by default), **"Pearson"**, **"working"** and **"response"**.

- Pearson residuals: $(y_i - \hat{\mu}_i)\,/\sqrt{\hat{V}_i}$.
- Working residuals: $(y_i - \hat{\mu}_i)\,/\,(d\mu_i/d\nu_i)$.
- Response residuals: $(y_i - \hat{\mu}_i)$.
- Deviance residuals: $sign(y_i - \hat{\mu}_i)\sqrt{d_i}$ where d_i is the observation-specific contributions to the deviance statistic D.

7.2 Example of Functional Logistic Regression Model

In this section we focus on the particular case in which the response is binary, this model is also called functional logistic regression (FLR). [Escabias *et al.* (2005)] models the relationship between the binary response Y and the functional covariate $X(t)$ by basis representation of the $X(t)$ and $\beta(t)$. The functional logistic regression model the probability, π_i , of the occurrence of an event, $Y_i = 1$, rather than the event $Yi = 0$, conditional on a vector of covariates $X_i(t)$ is expressed as:

$$y_i = \pi_i + \epsilon_i \ , i = 1, \ldots, n$$

where π_i is the expectation of Y given $X_i(t)$ that will be modeled as:

$$\pi_i = P[Y = 1|x_i(t) : t \in T] = \frac{exp\left\{\int_T X_i(t)\beta(t)dt\right\}}{1 + exp\left\{\int_T X_i(t)\beta(t)dt\right\}} \quad , i = 1, \ldots, n$$

with ϵ are the independent errors with zero mean.

Equivalently the logit transformations can be expressed as:

$$l_i = ln[\frac{\pi_i}{1 - \pi_i}] = \int_T X_i(t)\beta(t)dt$$

where, $E(e|X(t), t \in T) = 0$ and $Var(e|X(t), t \in T) = \sigma^2(g(\eta))$

As functional linear model seen in section 6.3, the main idea is to reduce the dimension of functional corvariates to few basis. Thee functional logistic regression (FLR) is estimated by **fregre.glm** that has the same arguments of the fregre.lm() function plus the family parameter (family=binomial()).

$$\hat{l}_i = \int_T X_i(t)\beta(t)dt \approx \mathbf{\tilde{X}\hat{b}} \qquad (7.4)$$

where $\mathbf{\tilde{X}\hat{b}} = \mathbf{C_i^T}\psi(\mathbf{t})\phi^{\mathbf{T}}(\mathbf{t})\mathbf{\hat{b}}$ for basis expansion (of class "fd") or $\mathbf{\tilde{X}\hat{b}} = \{f_{i1}, \ldots, f_{ik}\}$ for functional principal components basis.

Below, we show how to apply the FLR model with binary response (dichotomized fat content, 1 for $fat > 15$, 0 otherwise) in the Tecator dataset. The following code we uses a training sample (first 165 curves) of the second derivative of absorbance curves **X.d2** to estimate the response.

```
R> ind<-1:165
R> Fat.bin<-ifelse(tecator$y$Fat<15,0,1)
R> dataf=data.frame(tecator$y[ind,],Fat.bin[ind])
R> names(dataf)[4]<-"Fat.bin"
R> X.d2=fdata.deriv(absorp,nderiv=2)
R> ldata=list("df"=dataf,"X.d2"=X.d2[ind])
R> basis.pc2=create.pc.basis(X.d2[ind],c(1))
R> basis.x=list("X.d2"=basis.pc2)
R> res.pc=fregre.glm(Fat.bin ~ X.d2,ldata,family=binomial,basis.x=basis.x)
```

For illustration, the fitted object returned (**res.glm2**) can be used in other functions of the *"glm"* class such as: **summary()**.

```
R> summary.glm(res.pc)
Call: glm(formula = pf)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.49009  -0.07842  -0.01143   0.00304   2.01818

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    2.7477     0.7581   3.624  0.00029 ***
X.d2.PC1    4069.3956   909.8110   4.473 7.72e-06 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 227.713  on 164  degrees of freedom
Residual deviance:  32.889  on 163  degrees of freedom
AIC: 36.889
Number of Fisher Scoring iterations: 9
```

For logistic regression, two link functions are commonly used in **R** : (1) the canonical link, **logit**: $g(\pi) = log(\pi/(1-\pi))$ and (2) **probit**: $g(\pi) = \phi^{-1}(\mu_i)$, where ϕ is the normal cumulative distribution function. Other link function are also used: (3) the complementary log-log, **cloglog**: $g(\pi) = log(log(1-\pi))$ and (4) the **cauchit**, $g(u) = tan(\pi(u-1/2))$, see [Morgan and Smith (1992)].

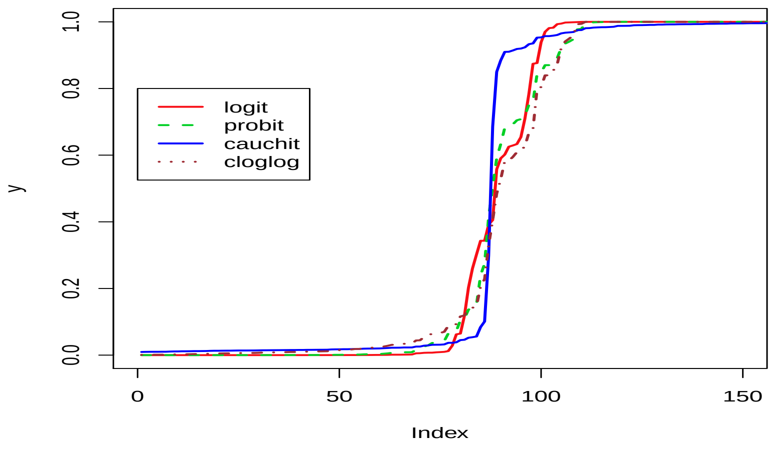

Figure 7.1: Form of the logistic function by logit (red), probit (green), cauchit (blue) and cloglog (brown) link.

In figure 7.1 shows the fitted values obtained by transforming the linear predictors by different link functions; Logit (red line) and probit (green line) link have similar shapes and different from others. The table 7.2 compares the models in terms of their deviance (the best model is the one with minimum deviance). The deviance is lower for the **logit** link (best fitted model) and higher for the **cloglog** link. Another criterion often used is the Akaike information criterion (AIC), in this case the best fit is for **probit** link and the worst for **cloglog** link.

Link	Deviance	AIC
logit	26.075	38.075
probit	32.823	36.823
cauchit	33.924	37.924
cloglog	37.72	41.72

Table 7.2: Deviance and AIC for fitted models

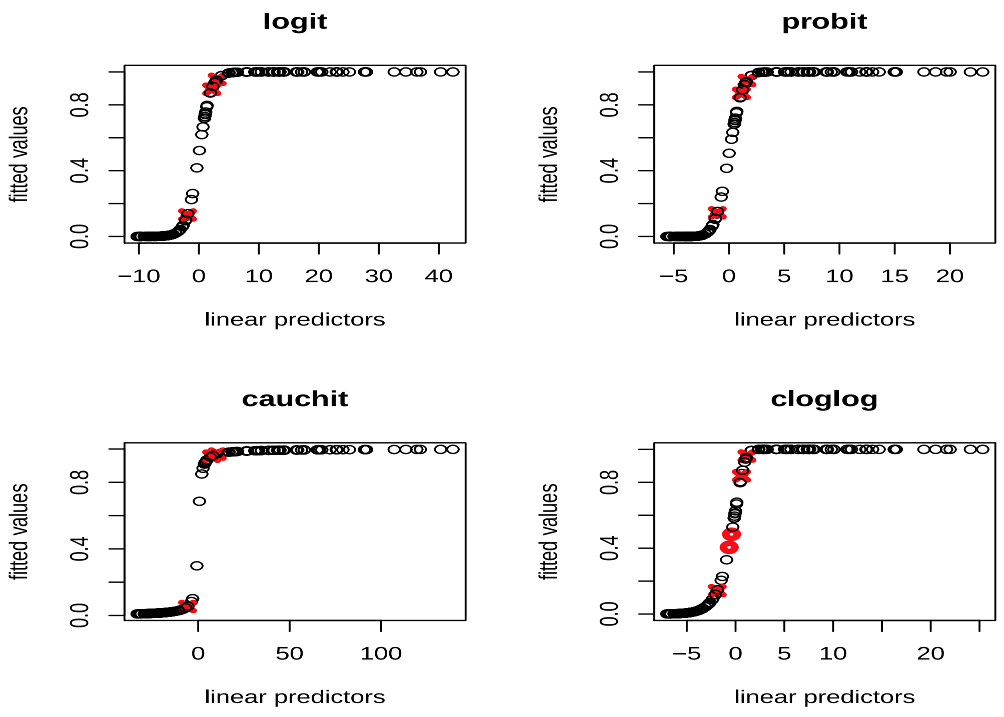

Figure 7.2: Linear predictors vs fitted values; curves correctly classifieds (black points) and incorrectly (red crosses) by link function: logit (top left), probit (top right), cauchit (bottom left) and cloglog (bottom right).

If new data is observed, the response can be predicted by the **predict.fregre.glm()** function.

```
R> newx.d2<-fdata.deriv(newx,nderiv=2,method="fmm")
R> newdataf=as.data.frame(tecator$y[-ind,])
R> newldata=list("df"=newdataf,"x.d2"=newx.d2)
R> pred.pc<-predict.fregre.glm(res.pc,newldata)
```

Logistic regression method can be used for a binary classification variable. Figure 7.2 displays linear predictors vs fitted values of the estimated fat content (dichotomized). All links classifieds bad the same curves (4 red crosses), but also the cloglog link classifieds erroneously other two curves. The prediction of dichotomized fat content is correct in 97.6% of cases in training sample (first 165 data) and 98% in test sample (last 50 data). Another way to summarize the classification is to use of confusion matrix between the theoretical response values (usually not observed) and the predicted values.

```
R>  y.pred1.pc1<-ifelse(res.pc$fitted.values>0.5,1,0)  # estimation
R>  table(y.pred1.pc1,Fat.bin[ind])
y.pred1.pc1  0  1
          0 86  2
          1  3 74
R> y.pred1<-ifelse (pred.pc>0.5,1,0)    # prediction
R> table(y.pred1,Fat.bin[-ind])
y.pred1  0  1
      0 23  1
      1  0 26
```

In order to test the results we have repeated 200 times changing the data in the training sample (length 165) and summarized into the table 7.4 the percentage of good classification of the binary response. We repeat the estimation process by different link and basis functions, see table 7.4.

Basis	Link	Min	Mean	Median	Max
1st PC	**logit**	96	98	98	99
1st PC	**probit**	100	100	100	100
1st PC	**cloglog**	92	95	95	100
1st PC	**cauchit**	96	98	98	100
1st 3 PC	**logit**	96	98	98	99
1st 3 PC	**probit**	100	100	100	100
1st 3 PC	**cloglog**	92	95	95	100
1st 3 PC	**cauchit**	93	97	97	100
5 fourier	**logit**	95	97	97	99
5 fourier	**probit**	100	100	100	100
5 fourier	**cloglog**	92	95	95	100
5 fourier	**cauchit**	72	97	96	100
5 bspline	**logit**	97	98	98	99
5 bspline	**probit**	100	100	100	100
5 bspline	**cloglog**	95	97	98	100
5 bspline	**cauchit**	93	99	99	100

Table 7.3: Percentage of good classification for training sample.

In GFLM model estimation, the percentage of good classification has been 100% for the **logit** link regardless of the type of basis used. The model with a single functional principal component basis works properly.

Finally, we repeat the process for predict new response values. We uses the 200 fitted models (training sample of length 165) and new curves (test sample of length 50) to predict the binary response, see table 7.4.

In prediction, the percentage of good classification still the best for the **logit** link, but now the **logit** link are also getting higher percentage of good classification. The predictions have been worse

Basis	Link	Min	Mean	Median	Max
1st PC	**logit**	92	97	98	100
1st PC	**probit**	92	98	98	100
1st PC	**cloglog**	84	94	94	100
1st PC	**cauchit**	86	96	96	100
1st 3 PC	**logit**	92	97	98	100
1st 3 PC	**probit**	92	98	98	100
1st 3 PC	**cloglog**	84	94	94	100
1st 3 PC	**cauchit**	86	96	96	100
5 fourier	**logit**	90	97	97	100
5 fourier	**probit**	90	98	98	100
5 fourier	**cloglog**	82	94	94	100
5 fourier	**cauchit**	72	95	96	100
5 bspline	**logit**	92	97	98	100
5 bspline	**probit**	92	98	98	100
5 bspline	**cloglog**	84	94	94	100
5 bspline	**cauchit**	86	96	96	100

Table 7.4: Percentage of good classification for test sample.

using 5 Fourier basis with the **cloglog** and **cauchit** links.

Chapter 8

Extensions

8.1 Also implemented in fda.usc package

The **fda.usc** package has implemented other utilities for statistical computing within the field of Functional Data Analysis (FDA). Some useful additions to the package include are:

- Functional ANOVA with random project, **anova.RPm()** [Cuesta *et al.* (2010)]: The procedure is based on the analysis of randomly chosen one-dimensional projections. The function tests ANOVA models for functional data with continuous covariates and perform special contrasts for the factors in the formula.

- Functional Supervised classification, [Ferraty and Vieu (2006)]: three non parametric methods for discriminating functional data; (1) **classif.knn.fd()** k-Nearest Neighbor method, (2) Non parametric kernel method **classif.kernel.fd()** and (3) Non parametric kernel method with basis representation **classif.kernel.fb()**. The **summary.classif.fd()** function displays information from classification methods (object of class 'classif.fd" and **predict.classif.fd()** function returns the classes predicted for new functional data using a previously trained model of class "classif.fd".

- Functional non-supervised classification **kmeans.fd()**: perform k-means clustering on functional data. The method searches the locations around which are grouped data (for a predetermined number of groups).

- Conditional distribution function, [Ferraty and Vieu (2006)]: (1) **cond.F()** function, calculates the conditional distribution function of a scalar response with functional data, (2) **cond.mode()** function, computes the mode for conditional distribution function maximizing the argument of the derivative of the conditional distribution function and (3) **cond.quantile()** that computes the quantile for conditional distribution function.

- Other utilities and auxiliary functions as for example; univariate ANOVA for heteroscedastic data **anova.hetero()** (used in **anova.RPm()**), **Kernel.integrate()** that computes the integrate kernels (used in **cond.F()**), **kmeans.center.ini()** that computes the k curves with greater distance are the initial centers (used in **kmeans.fd()**) and **h.default()** that calculates the smoothing parameter h for a functional data (used in **fregre.np()** and **fregre.plm()**).

Further documentation are also available at the websites `http://eio.usc.es/pub/MAESFE` and `http://eio.usc.es/pub/gi1914/` for projects *Metodologías y aplicaciones en estadística semi-paramétrica, funcional y espacio-temporal (MTM2008-03010)* and *Modelización estadística y apli-*

caciones (GI-1914) respectively. The file "script.zip file" contains the following example code:

i) aemet.raw.R script file: contains the code for download a selection of complete series of daily summaries of about 110 weather stations in Spain from:
ftp://ftpdatos.aemet.es/series_climatologicas/valores_diarios/estacion/, where there are daily data files grouped by station.

This script does:

1. Download files from Meteorological State Agency of Spain (AEMET).
2. Select the information to processed (station, latitude, period,,...) related with the functional variables selected (temperature, precipitation,...).
3. Return to list *aemet.raw* with variables selected and geographic information related with each curve on the object *aemet.raw$df*. 1 curve contain the daily functional variable by station and year.

In the script the user can make selections through geographical variables as: **altitude**, **longitude**, **latitude** or temporal variable as **year.ini** and **year.end**.

A list of possible functional variables to select is:

1. **tmax**: maximum daily temperature (in degrees Celsius).
2. **hour.tmax**: Time of the maximum temperature (in hh:mm) .
3. **tmin**: minimum daily temperature (in degrees Celsius).
4. **hour.tmin**: Time of the minimum temperature (in hh:mm).
5. **temp**: average daily temperature (in degrees Celsius).
6. **gust**: maximum wind gust (in m/sg).
7. **wind.dir**: Wind direction in tens of degree.
8. **hour.gust**: Time of the maximum gust (in hh:mm).
9. **wind.speed**: average wind speed (in m/sg).
10. **prec**: Daily precipitation (in millimeters mm).
11. **sun**: Duration of sunshine (in hours).
12. **presmax**: daily maximum pressure (in hPa).
13. **hour.presmax**: Time of the maximum pressure, rounded to the nearest whole hour.
14. **presmin**: daily minimum pressure (in hPa).
15. **hour.presmin**: Time of the minimum pressure, rounded to the nearest whole hour.

ii) Outliers_fdata.R script file: Functional outlier detection procedure for detecting outliers via smoothed bootstrap procedure based on trimming **outliers.depth.trim()** or based on weighed **outliers.depth.pond()**. Three examples are shown in this script code using the datasets: (1) poblenou dataset, see [Febrero-Bande *et al.* (2008)] (2) Canadian Weather dataset and (3) aemet dataset.

iii) flm_beta_estimation_brownian_data.R script file. This scripts estimates the functional beta parameter using simulated data where the theoretical beta is known. For more details see [Febrero-Bande *et al.* (2010)].

iv) Classif_phoneme.R script file: This script uses three non parametric methods for discriminating functional data [Ferraty and Vieu (2006)]: (1) k-Nearest Neighbor method, (2) Non parametric kernel method and (3) Non parametric kernel method with basis representation, see .

8.2 Not implemented in fda.usc package yet

We are working in other utilities for statistical computing within the field of Functional Data Analysis (FDA). Some useful additions, not implemented in the package yet, are:

- Wavelet basis for functional data.
- Functional (Generalized) Additive Models (FAM and FGAM).
- Penalties for (Generalized) Functional Linear Model, as Ridge or LASSO models.
- Functional Mixed Models
- Functional Regression Model with functional response.
- Extension of Zero-inflated Poisson (ZIP) regression model and Zero-inflated Negative Binomial (ZINB) regression model
- Supervised classification via: depth measures and Boosting algorithm.
- Functional spatio-temporal data analysis.

Chapter 9

Conclusion

The package **fda.usc** presented in this document is the result of the integration of our code with procedures from other authors, the package **fda** or the functions from **STAPH** group.

One major advantage of the package procedures is that avoids the need of basis representation of functional data. Using the new class **fdata**, the proposed methods can represent the functional data using the evaluations at the discretization points without having to define them in a basis and also, all procedures are adapted to this new class.

This package contains most of the methods recently developed for exploratory functional data analysis and for functional linear regression with scalar response. In addition, there is a gap in the practical application of generalized functional linear models in **R** that the procedures implemented in the package tried to cover.

Finally, the **fda.usc** package tries to be an integrated framework for FDA and is continually being developed, therefore updates will be available in CRAN (see the NEWS file.) Further documentation are also available at the project website whose URL is given in the DESCRIPTION file.

Acknowledgments

This work was supported by grants MTM2008-03010 from the Ministerio de Ciencia e Innovación, 10MDS207015PR from the Xunta de Galicia and GI-1914 MODESTYA-Modelización estadística y aplicaciones.

Bibliography

[Aneiros-Pérez and Vieu (2006)] Aneiros-Pérez G, Vieu P (2006). Semi-Functional Partial Linear Regression. *Statist. Probab. Lett.*, **76**(11), 1102–1110. ISSN 0167-7152.

[Cardot *et al.* (1999)] Cardot H, Ferraty F, Sarda P (1999). Functional Linear Model. *Statist. Probab. Lett.*, **45**(1), 11–22.

[Cardot *et al.* (2003)] Cardot H, Ferraty F, Sarda P (2003). Spline Estimators for the Functional Linear Model. *Statistica Sinica,*, **13**, 571–591.

[Crainiceanu and Goldsmith (2010)] Crainiceanu CM, Goldsmith AJ (2010). Bayesian functional data analysis using winbugs *Journal of Statistical Soft*, **32**(11).

[Cuesta *et al.* (2010)] Cuesta-Albertos, J.A., Febrero-Bande, M. (2010) A simple multiway ANOVA for functional data. *Test.*, **19**(3), 537–557.

[Cuevas *et al.* (2006)] Cuevas A, Febrero M, Fraiman R (2006). On the Use of the Bootstrap for Estimating Functions with Functional Data. *Comput. Statist. Data Anal.*, **51**(2), 1063–1074.

[Cuevas *et al.* (2007)] Cuevas A, Febrero-Bande M, Fraiman R (2007). Robust Estimation and Classification for Functional Data via Projection-Based Depth Notions. *Comput. Statist.*, **22**(3), 481–496.

[Escabias *et al.* (2005)] Escabias M, Aguilera AM, Valderrama, MJ(2005). Modeling environmental data by functional principal component logistic regression. *CEnvironmetrics*, **16**(1), 95–107.

[Escabias *et al.* (2007)] Escabias M, Aguilera AM, Valderrama MJ (2007). Functional PLS logit regression. *Computational Statistics and Data Analysis*, **51**, 4891–4902.

[Febrero-Bande *et al.* (2010)] Febrero-Bande M, Galeano P, González-Manteiga W (2010). Measures of Influence for the Functional Linear Model with Scalar Response. *J. Multivariate Anal.*, **101**(2), 327–339.

[Febrero-Bande *et al.* (2008)] Febrero-Bande M, Galeano P, González-Manteiga W (2008). Outlier Detection in Functional Data by Depth Measures, with Application to Identify Abnormal NO$_x$ Levels. *Environmetrics*, **19**(4), 331–345.

[Ferraty and Vieu (2006)] Ferraty F, Vieu P (2006). *Nonparametric Functional Data Analysis.* Springer Series in Statistics. Springer-Velag, New York. Theory and practice.

[Fraiman and Muniz (2001)] Fraiman R, Muniz G (2001). Trimmed Means for Functional Data. *Test*, **10**(2), 419–440.

[Goldsmith *et al.* (2011)] Goldsmith J, Bobb J, Crainiceanu C, Caffo B, Reich D(2011) Penalized Functional Regression. *Journal of Computational and Graphical Statistics.*

[Härdle (1990)] Härdle W (1990). *Applied Nonparametric Regression*, volume 19 of *Econometric Society Monographs*. Cambridge University Press, Cambridge.

[Hyndman and Shang (2010a)] Hyndman RJ, Shang HL (2010a). ***fds**: Functional Data Sets.* **R** package version 1.6., http://cran.r-project.org/package=fds.

[Hyndman and Shang (2010b)] Hyndman RJ, Shang HL (2010b). ***ftsa**: Functional Time Series Analysis.* **R** package version 2.6., http://cran.r-project.org/package=ftsa.

[Martens and Naes (1989)] Martens H, Naes T. (1989). Multivariate calibration. New York: Wiley

[McCullagh and Nelder (1989)] McCullagh P and Nelder JA (1989) Generalized Linear Models. Second ed. London: Chapman and Hall

[Morgan and Smith (1992)] Morgan JT, Smith DM (1992) P A note on Wadleys problem with overdispersion. *Applied Statistics*, **41**, 349–354.

[Müller and Stadtmüller (2005)] Müller HG and Stadtmüller U. (2005). Generalized functional linear models. *Ann. Statist.*., **33**, 774–805.

[Mevik and Wehrens (2007)] Mevik RW, Wehrens R (2007). *The **pls** Package: Principal Component and Partial Least Squares Regression in **R**.* **R** package version 2.1.0., http://cran.r-project.org/package=pls. *Journal of Statistical Software*, **18**(2), 1–24.

[Preda et al. (2007)] Preda C, Saporta G, Lévéder CL. (2007). PLS classification of functional data. *Comput. Stat*, **22**(2), 223–235.

[**R** Development Core Team(2011)] **R** Development Core Team (2011). ***R**: A Language and Environment for Statistical Computing.* **R** Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3. http://www.R-project.org/.

[Ramsay and Silverman (2002)] Ramsay JO, Silverman BW (2002). *Applied Functional Data Analysis: Methods and case studies.* Springer Series in Statistics. Springer-Verlag, New York.

[Ramsay and Silverman (2005)] *Functional Data Analysis.* Springer Series in Statistics, second edition. Springer-Velag, New York.

[Ramsay *et al.* (2010)] Ramsay JO, Wickham H GS, Hooker G (2010). ***fda**: Functional Data Analysis.* **R** package version 2.2.6., http://cran.r-project.org/package=fda.

[Reiss *et al.* (2010)] ***refund**: Regression with Functional Data.* **R** package version 0.1.3., http://cran.r-project.org/package=refund.

[Shang and Hyndman (2010)] Shang HL, Hyndman RJ (2010). ***Rainbow**: Rainbow Plots, Bagplots and Boxplots for Functional Data.* **R** package version 2.3.4., http://cran.r-project.org/package=rainbow.

[Venables and Ripley (2002)] Venables WN, Ripley BD (2002) Modern applied statistics with S. New York: Springer-Verlag.

[Wasserman (2006)] Wasserman L (2006). *All of Nonparametric Statistics.* Springer Texts in Statistics. Springer-Velag, New York.

Appendices

```
###################################################
#                 R code of:
# Utilities for Statistical Computing in
# Functional Data Analysis:  The R Package fda.usc
###################################################

###################################################
# Chapter 2:   Functional Data Definition:
# The new {R} class fdata
###################################################

###################################################
# 2.2 Displaying functional data using semi-metrics:
# example tecator dataset
###################################################
library(fda.usc)
data(tecator)
names(tecator);names(tecator$y)
absorp<-tecator$absorp.fdat
dim(absorp);nrow(absorp);ncol(absorp)
is.fdata(absorp)
sum1<-absorp[1]+absorp[2]
res1<-sum1-absorp[2]
res1==absorp[1]
Fat20<-ifelse(tecator$y$Fat<20,0,1)*2+2
names(absorp);par(mfrow=c(1,2))
plot(absorp,col=Fat20)
absorp.d1 = fdata.deriv(absorp, nderiv = 1 ,
method = "bspline" )
# Figure 1 (right panel)
plot(absorp.d1, col = Fat20)

###################################################
# 2.3 Displaying functional data using fd class:
# expample aemet dataset
###################################################
data(aemet)
names(aemet);head(aemet$df)
names(aemet$temp);temp<-aemet$temp
wind.speed<-aemet$wind.speed
logprec<-aemet$logprec
Z<-aemet$df
peninsular<-aemet$df$latitude>33
cl<-ifelse(peninsular,"gray","red")
library(mapproj);dev.off()

mapproject(map("world",c("Spain","Canary Island")),"mercator")
points(aemet$df$longitude,aemet$df$latitude,col=cl,pch=4)
```

```
###################################################
###  Convert fdata class to fd class
###################################################
dev.off();
temp.fd=fdata2fd(temp,type.basis="fourier",
nbasis=15)
class(temp.fd)
temp.fdata=fdata(temp.fd) #back to fdata
class(temp.fdata)
wind.speed.fd=fdata2fd(wind.speed,type.basis="fourier",nbasis=15)
logprec.fd=fdata2fd(aemet$logprec,type.basis="fourier",nbasis=15)
ans<-Z$altitude
par(mfrow=c(3,2))
plot(temp,col=cl,ylim=c(-1,29))
plot(temp.fd,col=cl,main="Temperature fd",ylim=c(-1,29))
plot(wind.speed,col=cl,ylim=c(0,12))
plot(wind.speed.fd,col=cl,main="Wind speed fd",ylim=c(0,12))
plot(logprec,col=cl)
plot(logprec.fd,col=cl,main="Log precipitation fd")

###################################################
# Chapter 3:  Functional data representation:
#             smoothing
###################################################
###################################################
###  3.1 Basis representation
###################################################
par(mfrow=c(1,2))
plot(create.fourier.basis(rangeval=absorp$rangeval,nbasis=7))
plot(create.fdata.basis(tecator$absorp.fdata,
l=c(1,3,5),maxl=7,type.basis="fourier"));dev.new()
plot(create.bspline.basis(rangeval=c(0,1),nbasis=7))

###################################################
###  3.3 Kernel smoothing
###################################################
a<-sort(runif(1000,-3,3))
par(mfrow=c(3,2))
plot(a,Ker.cos(a),type="l",ylim=c(0,1),axes=F,frame=T,ylab="",
xlab="a: Cosine kernel");axis(1,-3:3)
plot(a,Ker.unif(a),type="l",ylim=c(0,1),axes=F,frame=T,ylab="",
xlab="b: Uniform kernel");axis(1,-3:3)
plot(a,Ker.tri(a),type="l",ylim=c(0,1.2),axes=F,frame=T,ylab="",
xlab="c: Triweight kernel");axis(1,-3:3)
plot(a,Ker.epa(a),type="l",ylim=c(0,1),axes=F,frame=T,ylab="",
xlab="d: Epanechnikov kernel");axis(1,-3:3)
plot(a,Ker.quar(a),type="l",ylim=c(0,1),axes=F,frame=T,ylab="",
xlab="e: Quartic kernel");axis(1,-3:3)
plot(a,Ker.norm(a),type="l",ylim=c(0,1),axes=F,frame=T,ylab="",
xlab="f: Gaussian kernel");axis(1,-3:3)
integrate(Ker.cos,-1,1)
###  phoneme data and smoothing
data(phoneme);learn<-phoneme$learn
l<-c(0,2^seq(-2,9,len=30));nb<-seq(7,31,by=2)
out0<-min.basis(learn,lambda=l,numbasis=nb)
out1<-min.np(learn,Ker=Ker.epa)
out2<-min.np(learn,type.S=S.LLR)

####################################################
# 3.2 Validation criterion:
####################################################
par(mfrow=c(1,2))
contour(nb,l,out0$gcv,ylab="Lambda",xlab="Number of basis",
main="GCV criteria by min.basis()")
plot(out1$h,out1$gcv,type="l",main="GCV criteria  by min.np() ",
ylim=c(600,1200),xlab="Bandwidth (h) values",ylab="GCV criteria",
col=3,lwd=2)
lines(out2$h,out2$gcv,col=4,lwd=2)
legend(x=4.5,y=1125,legend=c("Ker.epa-S.NW","Ker.norm-S.LLR"),
lwd=c(2,2),col=c(3,4),cex=0.75)

dev.new();ind<-11
plot(learn[ind,],main="Phoneme curve[11]")
lines(out0$fdata.est[ind,],col=2,lty=2,lwd=2)
lines(out1$fdata.est[ind,],col=3,lty=3,lwd=2)
lines(out2$fdata.est[ind,],col=4,lty=4,lwd=2)
legend(x=70,y=19,legend=c("Bspline basis",
"Ker.epa-S.NW","Ker.norm-S.LLR"),lty=2:4,lwd=2,col=2:4)

####################################################
# 3.4 Functional Principal Components Analysis (FPCA)
####################################################
pc.svd<-fdata2pc(tecator$absorp,ncomp=3)
norm.fdata(pc.svd$rotation[1:2])
plot(pc.svd$rotation,main="",lwd=2)
fat<-tecator$y$Fat;summary(pc.svd)
X<-absorp
X.d1<-fdata.deriv(X,nderiv=1)
X.d2<-fdata.deriv(X,nderiv=2)
pc.svd<-fdata2pc(X,ncomp=3)
pc.svd1<-fdata2pc(X.d1,ncomp=3)
pc.svd2<-fdata2pc(X.d2,ncomp=3)
summary(pc.svd,fat,biplot=FALSE,corplot=TRUE)
summary(pc.svd1,fat,biplot=FALSE,corplot=TRUE)
summary(pc.svd2,fat,biplot=FALSE,corplot=TRUE)
```

```
###################################################
# 3.5 Functional Partial Least Squares (FPLS)
###################################################
X<-tecator$absorp
pls1<-fdata2pls(X,fat,ncomp=3)
plot(pls1$rotation,main="",lwd=2)
summary(pls1,corplot=TRUE)
a<-fdata2pc(X);b<-fdata2pls(X,fat);dev.off()
plot(a$rotation[1:2],lty=c(1,1),lwd=c(2,2))
lines(b$rotation,col=3:4,lty=2:3,lwd=c(2,2))

###################################################
# Chapter 4: Measuring distances of Functional Data
###################################################
r<-rnorm(1001,sd=.001);x<-seq(0,2*pi,length=1001)
fx<-fdata(sin(x)/sqrt(pi),x)
fx0<-fdata(rep(0,length=length(x))+r,x)
plot(c(fx,fx0))
metric.lp(fx,fx0)
integrate(function(x){(sin(x)/sqrt(pi))^2},0,2*pi)
1-dis.cos.cor(fx,fx0);semimetric.basis(fx,fx0);
semimetric.deriv(fx,fx0)/sqrt(2*pi)
semimetric.fourier(fx,fx0)*sqrt(2*pi)

###################################################
# 4.5 Semi-metric as classification rule:
#     example phoneme dataset
###################################################
glearn=phoneme$classlearn
mdist1<-metric.lp(learn)
mdist2<-semimetric.basis(learn,type.basis1 = "fourier")
mdist4<-semimetric.pca(learn,learn)
mdist5<-semimetric.mplsr(learn,learn,q=3,class1=glearn)

# figure dendogram
ind=c(120:130,220:230)
dev.new();par(mfrow=c(2,2))
a=mdist1[ind,ind];b=as.dist(a)
c2=hclust(b);c2$labels=glearn[ind]
plot(c2,main="metric.lp using L2",xlab="Class of each leaf")
a=mdist2[ind,ind];b=as.dist(a)
c2=hclust(b);c2$labels=glearn[ind]
plot(c2,main="semimetric.basis with fourier basis",
xlab="Class of each leaf")
a=mdist4[ind,ind];b=as.dist(a)
c2=hclust(b);c2$labels=glearn[ind]
```

```
plot(c2,main="semimetric.pca with 2 pc")
a=mdist5[ind,ind];b=as.dist(a)
c2=hclust(b);c2$labels=glearn[ind]
plot(c2,main="semimetric.mplsr",xlab="Class of each leaf")

###################################################
# Chapter 5: Descriptive analysis of Functional Data
###################################################
###################################################
# 5.1 Functional Depth
###################################################
# Centrality measures (trimmed)
par(mfrow=c(2,2))
plot(func.mean(absorp),ylim=c(2.6,3.6),
main="Centrality measures (15% trimmed mean)")
legend(x=835,y=3.65, cex=1,box.col="white",lty=1:4,
col=c(1:4),legend=c("mean","trim.FM","trim.mode","trim.RP"))
lines(func.trim.FM(absorp,trim=0.15),col=2)
lines(func.trim.mode(absorp,trim=0.15),col=3,lty=3)
lines(func.trim.RP(absorp,trim=0.15),col=4,lty=4)
# Centrality measures (ponderation)
plot(func.mean(absorp),main="Centrality measures (median)",
ylim=c(2.6,3.6))
legend(x=835,y=3.65,cex=1,box.col="white",lty=1:4,col=c(1:4),
legend=c("mean","med.FM","med.mode","med.RP"))
lines(func.med.FM(absorp),col=2)
lines(func.med.mode(absorp),col=3,lty=3)
lines(func.med.RP(absorp),col=4,lty=4)
# Measures of dispersion
plot(func.var(absorp),main="Dispersion measures",
ylim=c(0.07,.30))
legend(x=950,y=.295,cex=1,box.col="white",lty=1:4,col=1:4,
legend=c("var","trimvar.FM","trimvar.mode","trimvar.RP"))
lines(func.trimvar.FM(absorp,trim=0.15),col=2)
lines(func.trimvar.mode(absorp,trim=0.15),col=3,lty=3)
lines(func.trimvar.RP(absorp,trim=0.15),col=4,lty=4)
lines(func.var(absorp))
# Fraiman-Muniz Depth
out.FM=depth.FM(absorp,trim=0.1,draw=FALSE)
#Modal Depth
out.mode=depth.mode(absorp,trim=0.1,draw=FALSE)
tt= tecator$absorp[["argvals"]]
plot(out.mode$dep,out.FM$dep,main="FM depth vs mode depth",
xlab="mode depth",ylab="FM depth")

###################################################
###  5.2 Bootstrap for functional data
###################################################
```

```
# This take a lot
par(mfrow=c(1,2))
out.boot1=fdata.bootstrap(absorp,statistic=func.mean,
nb=1000,draw=TRUE)
out.boot2=fdata.bootstrap(absorp,statistic=func.trim.FM,
nb=1000,draw=TRUE)

###################################################
###  5.3 Functional Outlier Detection
###################################################
# This take a lot
data(poblenou);nox<-poblenou$nox
working=poblenou$nox[poblenou$df$day.festive==0 &
as.integer(poblenou$df$day.week)<6]
nonworking=poblenou$nox[poblenou$df$day.festive==1 |
as.integer(poblenou$df$day.week)>5]
out=outliers.depth.trim(nonworking,dfunc=depth.RP,
nb=1000,smo=0.1,trim=0.06)
out2= outliers.depth.trim(working,dfunc=depth.FM,
nb=1000,smo=0.1,trim=0.06)
par(mfrow=c(2,1))#  Figure outlier detection
plot(working,ylim=c(0,400),col="gray",lty=1,
main="NOx - Working days")
lines(working[out2[[1]]],col=2,lty=2,lwd=2)
plot(nonworking,ylim=c(0,400),col="gray",lty=1,
main="NOx - Non working days")
lines(nonworking[out[[1]]],col=2,lty=2,lwd=2)

###################################################
### Chapter 6: Functional regression models (FRM)
###################################################
ind=1:129   # training data
tt=argvals(absorp);y=tecator$y$Fat[ind]
X=absorp[ind,]
X.d1=fdata.deriv(X,nbasis=19,nderiv=1)
X.d2=fdata.deriv(X,nbasis=19,nderiv=2)

###################################################
###  6.1 Functional linear model with basis representation
###################################################
rangett<-absorp[ind,]$rangeval
basis1=create.bspline.basis(rangeval=rangett,nbasis=17)
basis2=create.bspline.basis(rangeval=rangett,nbasis=7)
res.basis0=fregre.basis(X,y,basis.x=basis1,basis.b=basis2)
summary(res.basis0);
bb<-seq(11,31,by=5);l<-2^(-15:5)
res.basis0=fregre.basis.cv(X,y,basis.x=51,basis.b=bb,lambda=l)
```

```
bb<-seq(9,53,by=4)
bsp.cv=fregre.basis.cv(X,y,basis.x=71,basis.b=bb,type.basis="bspline")
fou.cv=fregre.basis.cv(X,y,basis.x=71,basis.b=bb,type.basis="fourier")
plot(bb,bsp.cv$gcv,type="l",lwd=2,xlab="number of basis",ylab="GCV")
lines(bb,fou.cv$gcv,type="l",col=2,lwd=2)
bsp.cv$basis.b.opt$nbasis;fou.cv$basis.b.opt$nbasis

###################################################
# 6.2 Functional linear model with functional PCA basis
###################################################
res.pc0=fregre.pc(X,y,l=c(1,3,6))
summary(res.pc0)

###################################################
# 6.3 Functional linear model with functional PLS basis
###################################################
res.pls0=fregre.pls(X,y,l=1:6)
res.pls1=fregre.pls(X.d1,y,l=1:6)
res.pls2=fregre.pls(X.d2,y,l=1:5)

###################################################
# 6.3.1 How to select the components?
###################################################
fregre.pc.cv(X.d1,y,kmax=8)
fregre.pls.cv(X.d1,y,kmax=8,criteria="SICc")

###################################################
# 6.3.2 Functional influence measures
###################################################
# This take a lot
res.infl=influence.fdata(res.basis0)
mat=cbind(res.infl$DCP,res.infl$DCE,res.infl$DP)
colnames(mat)=c("CPi","CEi","Pi")
pairs(mat)
#res.quan<-influence.quan(res.basis0,res.infl,mue.boot=500,kmax.fix=TRUE)

###################################################
#6.3.3 Bootstrap for Functional linear model
###################################################
# This take a lot
par(mfrow=c(1,3))
fregre.bootstrap(res.basis0,nb=1000,kmax.fix=TRUE,alpha=.999)
fregre.bootstrap(res.pc0,nb=1000,kmax.fix=TRUE,alpha=.999)
fregre.bootstrap(res.pls0,nb=1000,kmax.fix=TRUE,alpha=.999)


###################################################
#6.4 Functional linear models with functional
```

```
#    and non functional covariate
####################################################
ind<-1:129
dataf=as.data.frame(tecator$y[ind,])
newdataf=as.data.frame(tecator$y[-ind,])
ldata=list("df"=dataf,"X"=X,"X.d1"=X.d1,"X.d2"=X.d2)
basis.x=list("X"=basis1);basis.b=list("X"=basis2)
f2=Fat~Water+X.d2
basis.x1=list("X.d2"=basis1)
basis.b1=list("X.d2"=basis2)
res.lm2=fregre.lm(f2,ldata,basis.x=basis.x1,basis.b=basis.b1)
coefficients(res.lm2)

####################################################
# 6.5 Nonparametric functional regression model
####################################################
 h<-seq(0.0125,0.1,len=100)
 np1=fregre.np.cv(X.d1,y,Ker=AKer.quar,h=h)
 plot(h,np1$gcv,col=2,type="l",ylab="GCV criterion",xlab="Bandwidth")
 np2=fregre.np.cv(X.d1,y,Ker=AKer.tri,h=h)
 lines(h,np2$gcv,col=3)
 np3=fregre.np.cv(X.d1,y,Ker=AKer.epa,h=h)
 lines(h,np3$gcv,col=4)

a<-sort(runif(1000,-3,3));par(mfrow=c(3,2))
plot(a,AKer.cos(a),type="l",ylim=c(0,2),axes=F,frame=T,ylab="",
xlab="a: Cosine kernel");axis(1,-3:3)
plot(a,AKer.unif(a),type="l",ylim=c(0,2),axes=F,frame=T,ylab="",
xlab="b: Uniform kernel");axis(1,-3:3)
plot(a,AKer.tri(a),type="l",ylim=c(0,2.2),axes=F,frame=T,ylab="",
xlab="c: Triweight kernel");axis(1,-3:3)
plot(a,AKer.epa(a),type="l",ylim=c(0,2),axes=F,frame=T,ylab="",
xlab="d: Epanechnikov kernel");axis(1,-3:3)
plot(a,AKer.quar(a),type="l",ylim=c(0,2),axes=F,frame=T,ylab="",
xlab="e: Quartic kernel");axis(1,-3:3)
plot(a,AKer.norm(a),type="l",ylim=c(0,2),axes=F,frame=T,ylab="",
xlab="f: Gaussian kernel");axis(1,-3:3)

####################################################
# 6.6 Semi-functional partially linear model (SFPLM)
####################################################
# Functional semi-parametric regression
res.plm2=fregre.plm(f2,ldata,Ker=AKer.tri,type.S=S.KNN)

####################################################
## 6.7 Predict methods for FLR fits
####################################################
newy=matrix(tecator$y$Fa[-ind],ncol=1)
```

```
newX=absorp[-ind,]
newX.d1=fdata.deriv(absorp[-ind,],nbasis=19,nderiv=1)
newX.d2=fdata.deriv(absorp[-ind,],nbasis=19,nderiv=2)
# Functional and nonfunctional covariate
newldata=list("df"=newdataf,"X"=newX,"X.d1"=newX.d1,
"X.d2"=newX.d2)
res.basis2=fregre.basis.cv(X.d2,y,type.basis="fourier")
pred.basis2=predict.fregre.fd(res.basis2,newX.d2)
res.pc1=fregre.pc(X.d1,y,l=1:6)
pred.pc1=predict.fregre.fd(res.pc1,newX.d1)
res.np2=fregre.np.cv(X.d2,y,metric=semimetric.fourier)
pred.np2=predict.fregre.fd(res.np2,newX.d2)

###################################################
# prediction II
###################################################
newldata = list(df = newdataf, X = newX, X.d1 =
newX.d1, X.d2 = newX.d2)
f = Fat ~ Water + X;f1 = Fat ~ Water + X.d1
basis.x1 = list(X.d1 = basis1)
basis.b1 = list(X.d1 = basis2)
res.lm1=fregre.lm(f1,ldata,basis.x=basis.x1,basis.b=basis.b1)
pred.lm1=predict.fregre.lm(res.lm1,newldata)
res.plm1=fregre.plm(f,ldata,metric=semimetric.deriv,nderiv=1)
pred.plm1=predict.fregre.plm(res.plm1,newldata)

###################################################
# Figure Boxplot residual
###################################################
pred.basis0=predict.fregre.fd(res.basis0,newX)
res.pc2=fregre.pc(X.d2,y,l=1:6)
pred.pc2=predict.fregre.fd(res.pc2,newX.d2)
pred.pc2=predict.fregre.fd(res.pc2,newX.d2)
pred.pls0=predict.fregre.fd(res.pls0,newX)
pred.pls1=predict.fregre.fd(res.pls1,newX.d1)
pred.pls2=predict.fregre.fd(res.pls2,newX.d2)
pred.lm2=predict.fregre.lm(res.lm2,newldata)
res.np1=fregre.np.cv(X.d1,y)
pred.np1=predict.fregre.fd(res.np1,newX.d1)
res.plm0=fregre.plm(f,ldata,Ker=AKer.tri,type.S=S.KNN)
pred.plm0=predict.fregre.plm(res.plm0,newldata)
pred.plm2=predict.fregre.plm(res.plm2,newldata)
res<-cbind(pred.basis2-newy,pred.pc1-newy,pred.pls2-newy,
pred.np2-newy,pred.lm2-newy,pred.plm2-newy)
colnames(res)<-c("basis","pc","pls","np","lm","plm")
colnames(res)<-c("pred.basis2","pred.pc1","pred.pls2",
"pred.np2","pred.lm1","pred.plm1")
boxplot(res,xlab="")
```

```
####################################################
#  Goodness of fit fucntion
####################################################
GOF.predict<-function(model,pred,newy){
 if (class(model)[1]=="fregre.fd") {
 df<-model$df; r2<-model$r2; sr2<-model$sr2
 }
else {
 if (class(model)[1]=="lm") {
 smr<-summary(model)
 df<-smr$df[1]; r2<-smr$r.squared
 sr2<-  sum(model$residuals^2)/(length(model$residuals) - df)
 }
}
MEP=((1/length(newy))*sum((newy-pred)^2))/var(newy)
out<-cbind(round(df,1),round(r2,3),round(sr2,3),round(MEP,4))
colnames(out)<-c("df","R-squared","Sr2","MEP")
 return(out)
}

####################################################
#  Table 1,  Goodness of fit
####################################################
tabl<-matrix(NA,nrow=13,ncol=4)
tabl[1,]<-GOF.predict(res.basis0,pred.basis0,newy)
tabl[2,]<-GOF.predict(res.basis2,pred.basis2,newy)
tabl[3,]<-GOF.predict(res.pc1,pred.pc1,newy)
tabl[4,]<-GOF.predict(res.pc2,pred.pc2,newy)
tabl[5,]<-GOF.predict(res.pls1,pred.pls1,newy)
tabl[6,]<-GOF.predict(res.pls2,pred.pls2,newy)
tabl[7,]<-GOF.predict(res.lm1,pred.lm1,newy)
tabl[8,]<-GOF.predict(res.lm2,pred.lm2,newy)
tabl[9,]<-GOF.predict(res.np1,pred.np1,newy)
tabl[10,]<-GOF.predict(res.np2,pred.np2,newy)
tabl[11,]<-GOF.predict(res.plm0,pred.plm0,newy)
tabl[12,]<-GOF.predict(res.plm1,pred.plm1,newy)
tabl[13,]<-GOF.predict(res.plm2,pred.plm2,newy)
colnames(tabl)<-c("df","R-squared","Sr2","MEP")
tabl

res.lm<-lm(Fat~Water,tecator$y[1:129,])
pred.lm<-predict(res.lm,tecator$y[-c(1:129),"Water"])
newy<-tecator$y[-c(1:129),"Fat"]
GOF.predict(res.lm,pred.lm,newy)

####################################################
# Chapter 7: Generalized Functional Linear Model (GFLM)
```

```
####################################################
####################################################
# 7.2 Example of Functional Logistic Regression Model
####################################################
ind<-1:165
Fat.bin<-ifelse(tecator$y$Fat<15,0,1)
tt=absorp[["argvals"]]
dataf=data.frame(tecator$y[ind,],Fat.bin[ind])
names(dataf)[4]<-"Fat.bin"
X.d2=fdata.deriv(absorp,nderiv=2)
ldata=list("df"=dataf,"X.d2"=X.d2[ind])
basis.pc2=create.pc.basis(X.d2[ind],c(1))
basis.x=list("X.d2"=basis.pc2)
res.pc=fregre.glm(Fat.bin ~ X.d2,ldata,family=binomial,basis.x=basis.x)
summary.glm(res.pc)
y.pred1.pc1<-ifelse(res.pc$fitted.values>0.5,1,0)
table(y.pred1.pc1,Fat.bin[ind])

####################################################
#plot(sort(Fat.bin[ind]))
f2<-Fat.bin ~ X.d2
res.pc2=fregre.glm(f2,ldata,family=binomial("probit"),basis.x=basis.x)
res.pc3=fregre.glm(f2,ldata,family=binomial("cauchit"),basis.x=basis.x)
res.pc4=fregre.glm(f2,ldata,family=binomial("cloglog"),basis.x=basis.x)
summary.glm(res.pc2)
cbind(res.pc$deviance,res.pc2$deviance,res.pc3$deviance,res.pc4$deviance)
cbind(summary(res.pc)$aic,summary(res.pc2)$aic,
summary(res.pc3)$aic,summary(res.pc4)$aic)

####################################################
plot(sort(res.pc$fitted.values),col=2,lwd=2,type="l",xlim=c(0,150),ylab="y")
lines(sort(res.pc2$fitted.values),col=3,lwd=2,lty=2)
lines(sort(res.pc3$fitted.values),col=4,lwd=2)
lines(sort(res.pc4$fitted.values),col="brown",lwd=2,lty=4)
legend(x=0,.8,legend=c("logit","probit","cauchit","cloglog"),
lwd=c(2,2,2,2),col=c(2:4,"brown"),lty=c(1,2,1,3))
nsa<-length(ind)
fit<-ifelse(res.pc$fitted.values>0.5,1,0)
mis1<-sum(fit==Fat.bin[ind])/nsa
fit2<-ifelse(res.pc2$fitted.values>0.5,1,0)
mis2<-sum(fit2==Fat.bin[ind])/nsa
fit3<-ifelse(res.pc3$fitted.values>0.5,1,0)
mis3<-sum(fit3==Fat.bin[ind])/nsa
fit4<-ifelse(res.pc4$fitted.values>0.5,1,0)
mis4<-sum(fit4==Fat.bin[ind])/nsa
round(cbind(mis1,mis2,mis3,mis4)*100,2)
####################################################
```

```
 newx.d2<-fdata.deriv(absorp,nderiv=2)[-ind]
 nt<-nrow(newx.d2)
 newdataf=data.frame(tecator$y[-ind,],Fat.bin[-ind])
 newldata=list("df"=newdataf,"X.d2"=newx.d2)
 pred.pc<-predict.fregre.glm(res.pc,newldata)
 y.pred1<-ifelse(pred.pc>0.5,1,0)
 table(y.pred1,Fat.bin[-ind])
 mis11<-sum(y.pred1==Fat.bin[-ind])/nt
 pred.pc2<-predict.fregre.glm(res.pc2,newldata)
 y.pred2<-ifelse(pred.pc2>0.5,1,0)
 mis22<-sum(y.pred2==Fat.bin[-ind])/nt
 pred.pc3<-predict.fregre.glm(res.pc3,newldata)
 y.pred3<-ifelse(pred.pc3>0.5,1,0)
 mis33<-sum(y.pred3==Fat.bin[-ind])/nt
 pred.pc4<-predict.fregre.glm(res.pc4,newldata)
 y.pred4<-ifelse(pred.pc4>0.5,1,0)
 mis44<-sum(y.pred4==Fat.bin[-ind])/nt
table(y.pred4,Fat.bin[-ind])
round(cbind(mis11,mis22,mis33,mis44)*100,2)
res.bs2=fregre.glm(Fat.bin ~ X.d2,ldata,family=binomial("cloglog"))
summary(res.bs2)
pred.bs2<-predict.fregre.glm(res.bs2,newldata)
y.pred1<-ifelse(pred.bs2>0.5,1,0)
 table(y.pred1,Fat.bin[-ind])

par(mfrow=c(2,2))
correct<-ifelse(fit==Fat.bin[ind],1,2);correct2<-ifelse(fit2==Fat.bin[ind],1,2);
correct3<-ifelse(fit3==Fat.bin[ind],1,2);correct4<-ifelse(fit4==Fat.bin[ind],1,2);
plot(res.pc$linear.predictors,res.pc$fitted.values,
col=correct,lwd=correct^2,pch=correct^2,main="logit",ylab="fitted values")
plot(res.pc2$linear.predictors,res.pc2$fitted.values,col=correct2,
lwd=correct2^2,pch=correct^2,main="probit",ylab="fitted values")
plot(res.pc3$linear.predictors,res.pc3$fitted.values,col=correct3,
lwd=correct3^2,pch=correct^2,main="cauchit",ylab="fitted values")
plot(res.pc4$linear.predictors,res.pc4$fitted.values,col=correct4,
lwd=correct4^2,pch=correct^2,main="cloglog",ylab="fitted values")
###################################################
# Percentage of good classification for training sample
###################################################
ind<-1:165;lenin<-165
lenout<-215-lenin;nrep=200
n<-nrow(absorp);tt=argvals(absorp)
Fat.bin<-ifelse(tecator$y$Fat<15,0,1)
prob.y<-sum(Fat.bin)/n      #
X<-absorp;X.d2=fdata.deriv(tecator$absorp,nderiv=2)
link=c("logit","probit","cloglog","cauchit")
lenlink<-length(link)
miss1clas=dev<-aic<-missclas<-array(NA,dim=c(4,nrep,4))
```

```
dimnames(miss1clas)[1]=dimnames(missclas)[1]=list(link)
dimnames(miss1clas)[3]=list(c("1 PC","3 PC","5 fou","5 bsp"))
dimnames(missclas)[3]==list(c("1 PC","3 PC","5 fou","5 bsp"))
f<-Fat.bin ~X.d2
b1<-create.fdata.basis(X,1:7,type.basis="fourier")
b2<-create.fdata.basis(X,1:5,type.basis="fourier")
basis.x<-list("X"=b1,"X.d2"=b1)
basis.b<-list("X"=b2,"X.d2"=b2)
PC1<-1;PC2<-1:3
for (i in 1:nrep) {
 set.seed(i)
 ind<-sample(n,lenin)
 dataf=data.frame(tecator$y[ind,],Fat.bin[ind])
 names(dataf)[4]<-"Fat.bin"
 x2<-X.d2[ind];x<-X[ind]
 ldata=list("df"=dataf,"X"=x,"X.d2"=x2)
 basis.pc=create.pc.basis(x,PC1)
 basis.pc2=create.pc.basis(x2,PC1)
 basis.pc11=list("X"=basis.pc,"X.d2"=basis.pc2)
 basis.pc=create.pc.basis(x,PC2)
 basis.pc2=create.pc.basis(x2,PC2)
 basis.pc22=list("X"=basis.pc,"X.d2"=basis.pc2)
 for (j in 1:lenlink) {
  res.pc1=fregre.glm(f,ldata,family=binomial(link=link[j]),
  basis.x=basis.pc11)
  res.pc2=fregre.glm(f,ldata,family=binomial(link=link[j]),
  basis.x=basis.pc22)
  res.basis1=fregre.glm(f,ldata,family=binomial(link=link[j]),
  basis.x=basis.x,basis.b=basis.b)
  res.basis2=fregre.glm(f,ldata,family=binomial(link=link[j]))
  y.pred1.pc1<-ifelse(res.pc1$fitted.values>0.5,1,0)
  y.pred1.pc2<-ifelse(res.pc2$fitted.values>0.5,1,0)
  y.pred1.basis1<-ifelse(res.basis1$fitted.values>0.5,1,0)
  y.pred1.basis2<-ifelse(res.basis2$fitted.values>0.5,1,0)
  miss1.pc1<-sum(y.pred1.pc1==Fat.bin[ind])/lenin
  miss1.pc2<-sum(y.pred1.pc2==Fat.bin[ind])/lenin
  miss1.basis1<-sum(y.pred1.basis1==Fat.bin[ind])/lenin
  miss1.basis2<-sum(y.pred1.basis2==Fat.bin[ind])/lenin
  miss1clas[1,i,j]<-c(miss1.pc1);
  miss1clas[2,i,j]<-c(miss1.pc2)
  miss1clas[3,i,j]<-c(miss1.basis1)
  miss1clas[4,i,j]<-c(miss1.basis2)
  newdataf=data.frame(tecator$y[-ind,],Fat.bin[-ind])
  newx.d2<-X.d2[-ind];newx<-X[-ind]
  newldata=list("df"=newdataf,"X"=newx,"X.d2"=newx.d2)
  pred.pc1<-predict.fregre.glm(res.pc1,newldata)
  pred.pc2<-predict.fregre.glm(res.pc2,newldata)
  pred.basis1<-predict.fregre.glm(res.basis1,newldata)
  pred.basis2<-predict.fregre.glm(res.basis2,newldata)
  y.pred.pc1<-ifelse(pred.pc1>0.5,1,0)
  y.pred.pc2<-ifelse(pred.pc2>0.5,1,0)
  y.pred.basis1<-ifelse(pred.basis1>0.5,1,0)
  y.pred.basis2<-ifelse(pred.basis2>0.5,1,0)
  miss.pc1<-sum(y.pred.pc1==Fat.bin[-ind])/lenout
  miss.pc2<-sum(y.pred.pc2==Fat.bin[-ind])/lenout
  miss.basis1<-sum(y.pred.basis1==Fat.bin[-ind])/lenout
  miss.basis2<-sum(y.pred.basis2==Fat.bin[-ind])/lenout
  missclas[1,i,j]<-c(miss.pc1);missclas[2,i,j]<-c(miss.pc2)
  missclas[3,i,j]<-c(miss.basis1);missclas[4,i,j]<-c(miss.basis2)
 }
}
round(apply(miss1clas,c(1,3),summary)*100)

####################################################
####################################################
```