Journal of Statistical Software

October 2012, Volume 51, Issue 4. *http://www.jstatsoft.org/*

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

Manuel Febrero-Bande
Universidad de Santiago
de Compostela

Manuel Oviedo de la Fuente
Universidad de Santiago
de Compostela

Abstract

This paper is devoted to the R package **fda.usc** which includes some utilities for functional data analysis. 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, with a scalar response and a functional explanatory data via non-parametric functional regression, basis representation or functional principal components analysis. 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 incorporate the two main references of functional data analysis: The R package **fda** and the functions implemented by Ferraty and Vieu (2006).

Keywords: functional data regression, representation of functional data, non-parametric kernel estimation, depth measures, outlier.

1. Introduction

The technological progress has led to the development of new, quick and accurate measurement procedures. As a consequence, further possibilities for obtaining experimental data are now available and some classical paradigms must be revised. For example, it is now possible (and even frequent) to find problems where the number of data points is greater than the number of variables. In many areas it is common to work with large databases, which often increase these observations of a random variable taken over a continuous interval (or in increasingly larger discretizations of a continuous interval).

For example, in fields such as spectroscopy, the measurement result is a curve that, at least,

has been evaluated in 100 points. This type of data, which we call functional data arise naturally in many disciplines. In economics, one could consider curves intra-day stock quotes. In environmental studies, one could find continuous measurements of atmospheric monitoring networks. The importance of functional data in image recognition or spatio-temporal information is also well-known.

Undoubtedly, package **fda** (Ramsay, Wickham, Graves, and Hooker 2012) is a basic reference to work in R programming environment (R Development Core Team 2012) with functional data. The books by Ramsay and Silverman (2005) and Ramsay and Silverman (2002) are well-known references in this field. 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 book by Ferraty and Vieu (2006) is another important reference incorporating non-parametric approaches as well as the use of other theoretical tools such as semi-norms and small ball probabilities that allow us to deal with normed or metric spaces. These authors are part of the French group STAPH maintaining the page `http://www.lsp.ups-tlse.fr/staph` where R software can be downloaded.

Other R software provides implementation of functional data analysis (FDA) in different areas. The package **rainbow** (Shang and Hyndman 2012) for functional data representation, the package **ftsa** (Hyndman and Shang 2012) for functional time series analysis, the package **refund** (Crainiceanu, Reiss, Goldsmith, Huang, Huo, and Scheipl 2012) for functional penalized regression, the package **fpca** (Peng and Paul 2011) implements the restricted maximum likelihood estimation for functional principal components analysis (FPCA), the package **MFDF** (Dou 2009) for modeling functional data in finance via generalized linear models (GLM), the package **fdaMixed** (Markussen 2011), for FDA in a mixed model framework and the package `geofd` (Giraldo, Delicado, and Mateu 2010) for spatial prediction for function value data. In MATLAB **PACE** package (see `http://anson.ucdavis.edu/~ntyang/PACE/`) provides implementation of various methods using FPCA as generalized functional linear models (GFLM) for sparsely sampled functional data, this package is also in R (Liu 2011). Finally, from a Bayesian perspective, Crainiceanu and Goldsmith (2010) have developed tools for GFLM using **WinBUGS**.

The aim of the package **fda.usc** is to provide a broader, flexible tool for the analysis of functional data. Therefore, we propose an integration of the work on non-parametric functional methods implemented by Ferraty and Vieu (2006) with those from the group of the University of Santiago de Compostela (USC), thus complementing and extending some of the functions of package **fda**. The **fda.usc** package is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=fda.usc` and it has the greedy objective of being an integrate framework for all the functional data methods available.

In Section 2.1 we introduce a new R class for functional data: "`fdata`". Section 2.2 describes the functional representation of an object of class "`fdata`" by basis representation or kernel smoothing. Section 2.3 is focused in normed and semi-normed functional spaces and in how the metrics and semimetrics can be used as measures of proximity between functional data. Section 2.4 is concerned with the concept of statistical depth for functional data see Cuevas, Febrero-Bande, and Fraiman (2007). These depth measures are useful to define location and dispersion measures, for classification and for outlier detection (Febrero-Bande, Galeano, and González-Manteiga 2008) (Section 2.6). Section 3 is devoted to functional regression models with scalar response from different perspectives: Parametric, using basis representation (Section 3.1, 3.2 and 3.3), non-parametric, using kernel smoothing (Section 3.4) and semi-

parametric which combines both (Section 3.5). The Section 3.6 summarizes the functional regression models using a classical example to show the capabilities of the package **fda.usc**. Finally, Section 4 discusses the most important features of the package.

2. Functional data: Definition and descriptive analysis

After a brief introduction of the state of the art in functional data, we describe the most important features, procedures and utilities of the new package. This article is not intended to enter into many methodological details because most of the functions of the package are adaptations of original functions studied in previous publications.

2.1. Functional data definition: The new R class "fdata"

A functional variable $\mathcal{X}$ is just a random variable taking values in a functional space $\mathcal{E}$. Thus, a functional data set is just a sample $\{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$ (also denoted $X_1(t), \ldots, X_n(t)$ when convenient) drawn from a functional variable $\mathcal{X}$. Usually, $\mathcal{E}$ is assumed to be a normed or semi-normed metric space. If $\mathcal{E}$ is a Hilbert space, functional data can be represented using a basis. This is the approach of the **fda** package for its methodological developments and it is also used by **refund**, **geofd** or **MFDF** packages, among others. In practice, this representation projects an infinite dimensional space into a finite one and has the drawback of the approximation of the original data. However, the functional space $\mathcal{E}$ may not be Hilbertian and in this case, a more flexible representation of functional data is needed. For this reason, this package introduces the new class "`fdata`". This new class uses only the values evaluated on a grid of discretization points $\{t_1, \ldots, t_m\}$ that can be observed in a non-equispaced way. The drawback now is that every calculation (distance, norms,...) must be made through numerical approximations and the design or the density of this grid could affect the accuracy of the calculations. If functional data is sparsely sampled or there are many missing data points, probably the representation in a basis is mandatory and some other alternatives, as for example **PACE** package, should be considered.

The **fda.usc** package defines an object called `fdata` as a list of the following components:

- `data`: Typically a matrix of (n, 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.

All the methods in the package can work with the new class "`fdata`". Some basic operations to handle this new class are introduced. For example, generic methods for the *math* group `abs`, `sqrt`, `round`, `exp`, `log`, `cumsum`; for the *OPS* group "`+`", "`-`", "`*`", "`/`", "`==`", "`<`", '`>`", "`<=`", '`>=`"; the for *summary* group `all`, `any`, `sum`, `min`, `max`; and other operations `is.fdata()`, `c()`, `dim()`, `ncol()`, `nrow()`, `plot()`, `inprod.fdata()` and `norm.fdata()`. To calculate the derivative of functional data the `fdata.deriv()` function has been implemented, which computes the derivative by different methods, some purely numerical and others based on a

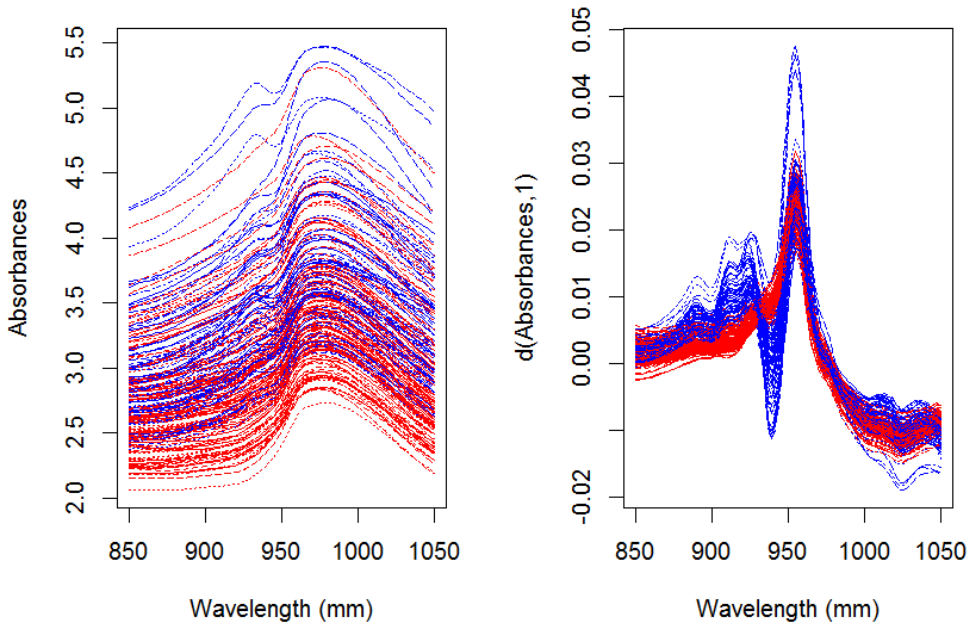

Figure 1: Spectrometric curves (left panel): curves with Fat $< 20\%$ (red lines) and curves with Fat $\geq 20\%$ (blue lines). Corresponding spectrometic curves after first order differencing (right panel).

basis representation. The choice of the right method depends on the nature of the data and on the grid of discretization.

The function `fdata()` converts a data object of class: "`fd`", "`fds`", "`fts`", "`sfts`" or other formats ("`vector`", "`matrix`", "`data.frame`") to an object of class "`fdata`". We have included in the "`fdata`" format some popular datasets of functional data: `tecator`, `phoneme`, `poblenou` and `aemet`. As an illustration we applied some of above methods to the `tecator` dataset (`data("tecator")`) that has two components. The first component, `tecator$absorp.fdata`, includes 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`, is a dataframe including `Fat`, `Water` and `Protein` contents of each spectrometric curve obtained by an analytical chemical processing. The `tecator` dataset presents interesting features and it is a well known example in functional data analysis (available at `http://lib.stat.cmu.edu/datasets/tecator`). The following code displays the absorbance curves colored by the fat content, see Figure 1.

```
R> library("fda.usc")
R> data("tecator")
R> names(tecator)

[1] "absorp.fdata" "y"
```

```
R> absorp <- tecator$absorp.fdata
R> Fat20 <- ifelse(tecator$y$Fat < 20, 0, 1) * 2 + 2
R> plot(tecator$absorp.fdata, col = Fat20)
R> absorp.d1 <- fdata.deriv(absorp, nderiv = 1)
R> plot(absorp.d1, col = Fat20)
```

The representation of a functional dataset should be consistent with the physical interpretation of the phenomenon being described and so it is important to choose the space where the data must be considered. For example, in left panel of Figure 1, the spectrometric curves are plotted showing with different colors two levels of 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 shown, for example, in the right panel of Figure 1 with the first derivative of the spectrometric curves.

2.2. Smoothing

If we assume 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 from $X(t)$, we can get back the original signal $X(t)$ using a linear smoother,

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

where s_{ij} is the weight that the point t_j gives to the point t_i and $y_i = Y(t_i)$. In the package two procedures have been implemented for this purpose. The first one is the representation in a $\mathcal{L}_2$ basis (or penalized basis) and the second one is based on the smoothing kernel methods.

Basis representation

A curve can be represented by a basis when the data is assumed to 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 approximated by a linear combination of a sufficiently large number k_n of these functions, see Ramsay and Silverman (2005, p. 43 and 44).

The procedure recovers the function $X(t)$ by using a fixed truncated basis expansion in terms of k_n known basis elements,

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

The projection (or smoothing) matrix is given by: $S = \Phi(\Phi^\top\Phi)^{-1}\Phi^\top$, with degrees of freedom of the fit $df = \text{tr}(S_\nu) = k_n$.

If smoothing penalization is required, a parameter λ will also be provided and in this case the projection matrix S is: $S = \Phi(\Phi^\top\Phi + \lambda R)^{-1}\Phi^\top$, where R is the penalization matrix. Typically R penalizes the integral of the square of the derivative of order 2.

There are several different types of basis (Fourier, B-spline, Exponential, power, polynomial, ...) that can be created using the function `create.fdata.basis()` or directly through the `create.`*basis-name*`.basis` from the **fda** package. This package also includes the basis functions based on the data: `create.pc.basis()` and `create.pls.basis()` that provides a functional principal component and partial least square basis, respectively. The choice of the basis should be based on the objective of the analysis and on the data. For example, it is common to use the Fourier basis for periodic data, and B-spline basis for non-recurrent data. Also, the function `fdata2fd()` converts an object of class "`fdata`" into an object of class "`fd`" using the basis representation shown in (1).

```
R> class(absorp.fd <- fdata2fd(absorp, type.basis= "fourier", nbasis= 15))

[1] "fd"

R> class(absorp.fdata <- fdata(absorp.fd))

[1] "fdata"
```

Kernel smoothing

The non-parametric methodology and in particular, the kernel smoothing method, can also be used to represent functional data. Now, the non-parametric smoothing of functional data is given by the smoothing matrix $S = (s_{ij})$. For example, for the Nadaraya-Watson estimator (function `S.NW()`): $s_j(t_i) = K\left(\frac{t_i - t_j}{h}\right) / \sum_{k=1}^{m} K\left(\frac{t_i - t_k}{h}\right)$ where $K(\cdot)$ is the kernel function. Other possibilities for S are the k nearest neighbors estimator (function `S.KNN()`) or the local linear regression estimator (function `S.LLR()`), see Wasserman (2006), Ferraty and Vieu (2006) for futher details. Many different types of kernels are considered in the package: Gaussian, Epanechnikov, triweight, uniform, cosine or any user-defined.

Validation criterion

The choice of the smoothing parameter is crucial and, in principle, there is no universal rule that would enable an optimal choice. Among the different selection criterion to select the parameter ν, we have implemented two: Cross-validation (CV) and generalized cross-validation (GCV).

$$\text{Cross-validation}: CV(\nu) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_{-i}^{\nu}\right)^2 w_i \tag{2}$$

where $\hat{y}_{i(-i)}^{\nu}$ indicates the estimator based on leaving out the i pair (t_i, y_i) and w_i is the weight at point t_i. This criterion is implemented by the function `CV.S()`.

$$\text{Generalized cross-validation}: GCV(\nu) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_{i}^{\nu}\right)^2 w_i \Xi(\nu) \tag{3}$$

where $\Xi(\nu)$ denotes the type of penalizing function.

Generalized cross-validation criterion is implemented in `GCV.S()` function with the following types of penalized functions (see Härdle 1990, for futher details):

- Generalized cross-validation (GCV): $\Xi(\nu) = (1 - \text{tr}(S_\nu)n^{-1})^{-2}$.
- Akaike's information criterion (AIC): $\Xi(\nu) = \exp(2\text{tr}(S_\nu)n^{-1})$.
- Finite prediction error (FPE): $\Xi(\nu) = (1 + \text{tr}(S_\nu)n^{-1})/(1 - \text{tr}(S_\nu)n^{-1})$.
- Shibata's model selector (Shibata): $\Xi(\nu) = (1 + 2\text{tr}(S_\nu)n^{-1})$.
- Rice's bandwidth selector (Rice): $\Xi(\nu) = (1 - 2\text{tr}(S_\nu)n^{-1})^{-1}$.

Smoothing FDA with ***fda.usc*** *package*

Let us now consider, for illustration purposes, the Phoneme dataset `data("phoneme")`[1] see, e.g., Ferraty and Vieu (2006). The `phoneme$classlearn` object contains 250 speech frames with class membership: "sh" (1), "iy" (2), "dcl" (3), "aa" (4) and "ao" (5). From each speech frame, a log-periodogram of length 150 has been stored in `phoneme$learn` which is used as learning sample. The goal is to predict the class membership `phoneme$classtest` using the test sample `phoneme$test`.

The purpose of the function `min.basis()` is to represent the functional data in terms of a (truncated) expansion with respect to a given basis of functions in the corresponding space. Such expansions depend on a parameter $\nu = (k_n, \lambda)$, where k_n is the number of basis elements used in the truncated expansion and λ is the penalization parameter. In the following example (see below R code), `phoneme$learn` is smoothed using `min.basis()` function.

```
R> data("phoneme")
R> learn <- phoneme$learn
R> l <- c(0, 2^seq(-2, 9, length.out = 30))
R> nb <- seq(7, 31, by = 2)
R> out0 <- min.basis(learn, lambda = l, numbasis = nb)

 The minimum GCV (GCV.OPT=2.3815) is achieved with
 the number of basis (numbasis.opt=27)
 and lambda value    (lambda.opt=81.27995)
```

In the left panel of Figure 2, the GCV criterion is drawn in function of the number of B-spline basis elements $\nu_1 =$ `nbasis` and the penalizing parameter $\nu_2 = \lambda$ using `min.basis()` function and the smoothed functional data $\hat{x}(t)$ (in `out0$fdata.est` object) can be used as the signal we have tried to recover. Figure 3 shows the 11th curve of speech frame x_{11} (`phoneme$learn[11]`) and the smooth representation of the curve $\hat{x}_{11}$ (`out0$fdata.est[11]`). The `min.np()` function returns the "optimal" value (in `out1$h.opt`) of the smoothing parameter $\nu = h$ that best represents the functional data for a range of values of bandwidth h using the validation criterion (2) and (3) shown above. As an illustration, in the right panel of Figure 2, the GCV criterion is drawn in function of the bandwidth h using the function `min.np()` and Figure 3 shows smooth representation of 11th curve of `phoneme$learn` by Nadaraya-Watson (`S.NW()`), local linear regression (`S.LLR()`) method and K nearest neighbors (`S.KNN()`) method.

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

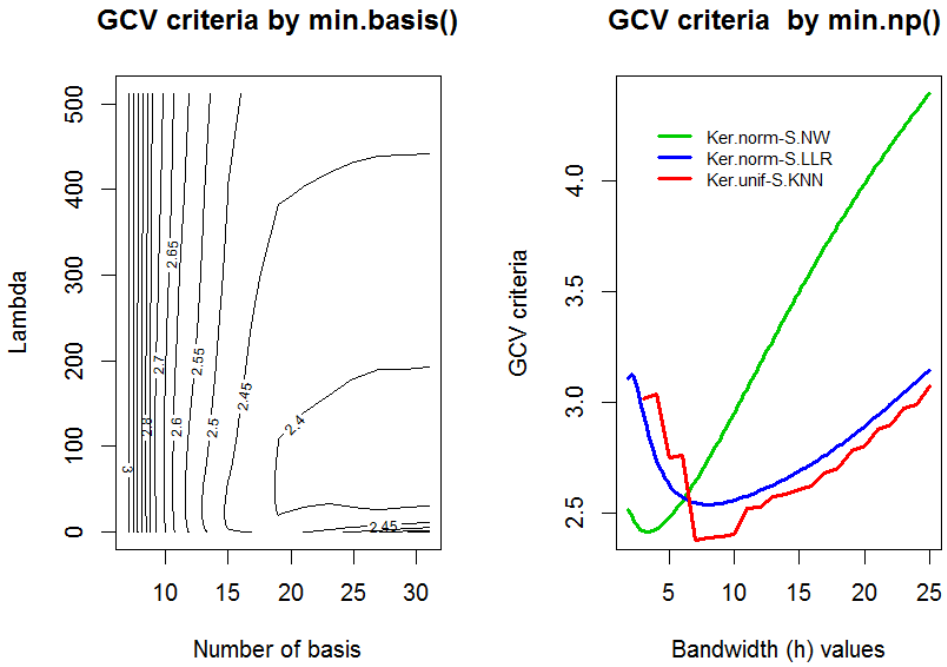

Figure 2: GCV criterion as a function of the number of B-spline basis elements and the penalizing parameter λ (left panel). GCV criterion as a function of bandwidth parameter (right panel): Normal kernel and local linear smoothing matrix (blue line); Normal kernel and Nadaraya-Watson smoothing matrix (green line) and uniform kernel and K nearest neighbors smoothing matrix (red line).

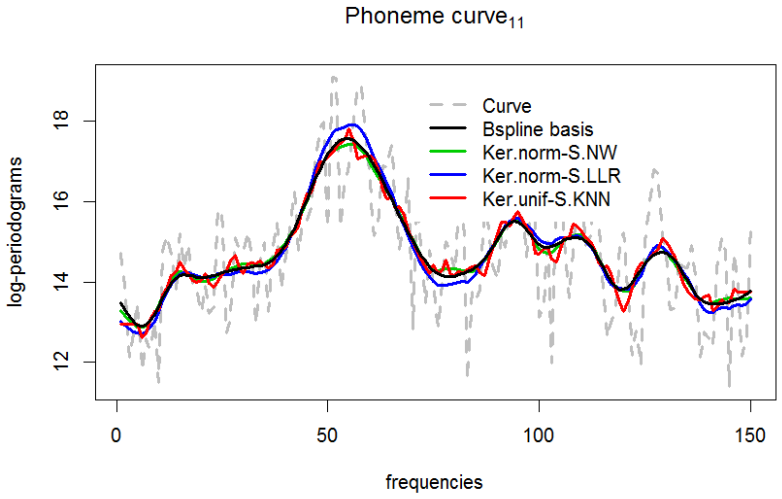

Figure 3: Eleventh phoneme curve: Observed (grey dashed line), smoothed by 27 B-spline basis elements and $\lambda = 81.28$ (black line), smoothed by normal kernel and Nadaraya-Watson smoothing matrix with bandwidth $h = 3.28$ (green line), smoothed by normal kernel and local linear smoothing matrix with bandwidth $h = 7.85$ (blue line) and smoothed by uniform kernel and K nearest neighbors smoothing matrix with bandwidth $h = 7$ (red line).

```
R> out1 <- min.np(learn, type.S = S.NW, par.CV = list(criteria = "GCV"))

 The minimum GCV (GCV.OPT=2.4119) is achieved with
 the h value (h.opt=3.2765)

R> out2 <- min.np(learn, type.S = S.LLR, par.CV = list(criteria = "GCV"))

 The minimum GCV (GCV.OPT=2.5363) is achieved with
 the h value (h.opt=7.9482)

R> out3 <- min.np(learn, type.S = S.KNN, h = 3:25, Ker = Ker.unif)

 The minimum GCV (GCV.OPT=2.3746) is achieved with
 the h value (h.opt=7)
```

2.3. Measuring distances

Sometimes, it is difficult to find the best plot to summarize a particular functional dataset because the shape of the graphics depends strongly on the chosen proximity measure. As shown in Figure 1, the plot of $X(t)$ against t is not necessarily the most informative and perhaps, considering another distance between curves we can obtain better results. This package collects several metric and semi-metric functions which allow us to extract as much information possible from the functional variable.

The most general 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 $d(\cdot, \cdot)$ is associated with a norm (so that $d(X(t), Y(t)) = \|X(t) - Y(t)\|$) we have a normed space (or a Banach space). In some important cases the norm $\|.\|$ is associated with an inner product $\langle ., . \rangle$ in the sense that $\|X\| = \langle X, X \rangle^{1/2}$. A complete normed space (Banach space) whose norm derives from an inner product is called a Hilbert space. The best known example is the space $\mathcal{L}_2[a, b]$ of real square-integrable functions defined on $[a, b]$ with $\langle f, g \rangle = \int_a^b fg$.

Utilities for computing distances, norms and inner products are included in the package. If we focused on $\mathcal{L}_p$ spaces (the set of functions whose absolute value raised to the p-th power has finite integral), the function `metric.lp()` uses Simpson's rule to compute distances between elements: $\|f(t) - g(t)\|^p = \left(\frac{1}{\int_a^b w(t)dt} \int_a^b |f(t) - g(t)|^p \, w(t)dt \right)^{1/p}$ where w are the weight and the observed points on each curve t are equally spaced (by default) or not. `norm.fdata()` computes the norm and, specifically for $\mathcal{L}_2$, `inprod.fdata()` calculates the inner product between elements of the space.

In the next example, the distances between some training sample curves of phoneme data (`phoneme$learn`) are calculated by the `metric.lp()` function. Figure 4 shows the dendrogram for a selection of 11 curves of class `(3)` (corresponding to the index from 110 to 120 curves) and 11 curves of class `(5)` (from 220 to 230). This example can be understood as a classification problem in which the goal is to classify the curves in 2 classes.

```
R> mdist <- metric.lp(learn)
```

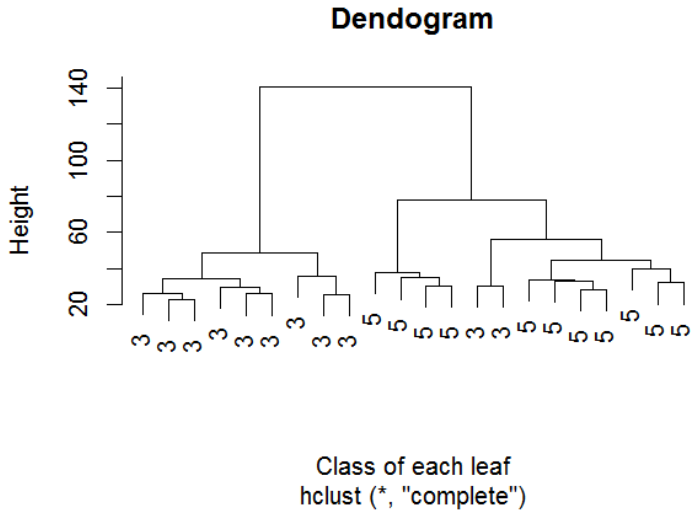

Figure 4: Dendogram for 22 phoneme curves divided into 2 classes using $\mathcal{L}_2$ metric.

Other semi-metric functions are included in the **fda.usc** package to compute distances between "`fdata`" objects under the assumption that these objects belongs to a metric o semi-metric space. `semimetric.basis()` function calculates the semi-metric $\mathcal{L}_2$ of the derivatives of order q (by default $q = 0$), $d_q(f,g) = \sqrt{\frac{1}{\sqrt{T}}\int_T (f^{(q)}(t) - g^{(q)}(t))^2 dt}$ for functional data of two "`fdata`" (or "`fd`") class objects. An alternative, without using the basis function of **fda** package, is to use the semi-metrics proposed by Ferraty and Vieu (2006); `semimetric.deriv()` and `semimetric.fourier()` functions calculate distances between the derivative of order `nderiv` using B-spline approximation and Fourier approximation, respectively. Other semi-metric functions have been included in the package, see `help(semimetric.NPFDA)`. The functions `semimetric.pca()` and `semimetric.mplsr()` compute proximities between curves based on the functional principal components analysis (FPCA) method and functional partial least-squared (FPLS) method, respectively. The FPCA and FPLS semi-metric can be used only if the curves are observed at the same discretized points and in a fine enough grid. Finally, `semimetric.hshift()` computes proximities between curves taking into account a horizontal shift effect of two functional objects.

The procedures of the package **fda.usc** which include the argument `metric` allow us the use of metric or semi-metrics functions implemented or other user defined metric with the only restriction that the first two arguments belong to the class "`fdata`".

2.4. Exploring functional data

Exploratory functional data analysis can help the user to analyze datasets by summarizing their main characteristics (identify features), detect possible errors or unexpected variability and resume the data.

In **fda.usc**, the usual tools for summarizing functional data are included: `func.mean()`, `func.var()`, `fdata2pc()` and `fdata2pls()` for computing the mean, the marginal variance, the principal eigenfunctions and the partial least squared, respectively. The output of these

tools is always an object of class "fdata". Different depth notions have been proposed in the literature, with the aim of measuring how deep a data point is in the sample. In univariate data, the median would typically be the deepest point of a cloud of points. In functional data, although there are more depth measures, this package includes those that are contained in the work of Cuevas *et al.* (2007) as `depth.FM()` that is based on the median (proposed by Fraiman and Muniz 2001), `depth.RP()` that is calculated through random projections (RP) based on the Tukey depth or `depth.RPD()` that is calculated through random projections of the curves and their derivatives and `depth.mode()` that is based on how surrounded the curves are with respect to a metric or a semi-metric distance, selecting the trajectory most densely surrounded by other trajectories of the process. The population h-depth of a datum z is given by the function:

$$D(z) = E\left(K_h(d(z, X))\right)$$

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

$$\hat{D}(z) = \frac{1}{n} \sum_{i=1}^{n} K_h(d(z, X_i))$$

All depth functions return: The deepest curve `median`, the index of the deepest curve `lmed`, the mean of $(1-\alpha)\%$ deepest curves `mtrim`, the index of $(1-\alpha)\%$ deepest curves `ltrim` and the depth of each curve `dep`.

An interesting application of the proposed depth measures is their use as measures of location and dispersion. In the following example we use `data("poblenou")` where each curve represents the evolution of NO_x levels in 1 day had measured every hour $\{t_i\}_{i=0}^{23}$ by a control station in Poblenou in Barcelona (Spain). We split the whole sample into working and non-working days.

```
R> data("poblenou")
R> nox <- poblenou$nox
R> dd <- as.integer(poblenou$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]
```

The top row of Figure 5 displays the α-trimmed means and the medians for *h-modal* and RP depth for working days (top left) and for non-working days (top right). The bottom row of Figure 5 shows the marginal variance using trimmed subsets for working days (bottom left), and for non-working days (bottom right).

```
R> plot(func.mean(working), ylim = c(10, 170),
+    main = "Centrality measures in working days")
R> legend(x = 11,y = 185, cex = 1, box.col = "white", lty = 1:5, col = 1:5,
+    legend = c("mean", "trim.mode", "trim.RP", "median.mode", "median.RP"))
R> lines(func.trim.mode(working, trim = 0.15), col = 2, lty =2)
R> lines(func.trim.RP(working, trim = 0.15),col = 3,lty = 3)
R> lines(func.med.mode(working, trim = 0.15),col = 4,lty = 4)
R> lines(func.med.RP(working, trim = 0.15),col = 5, lty = 5)
```

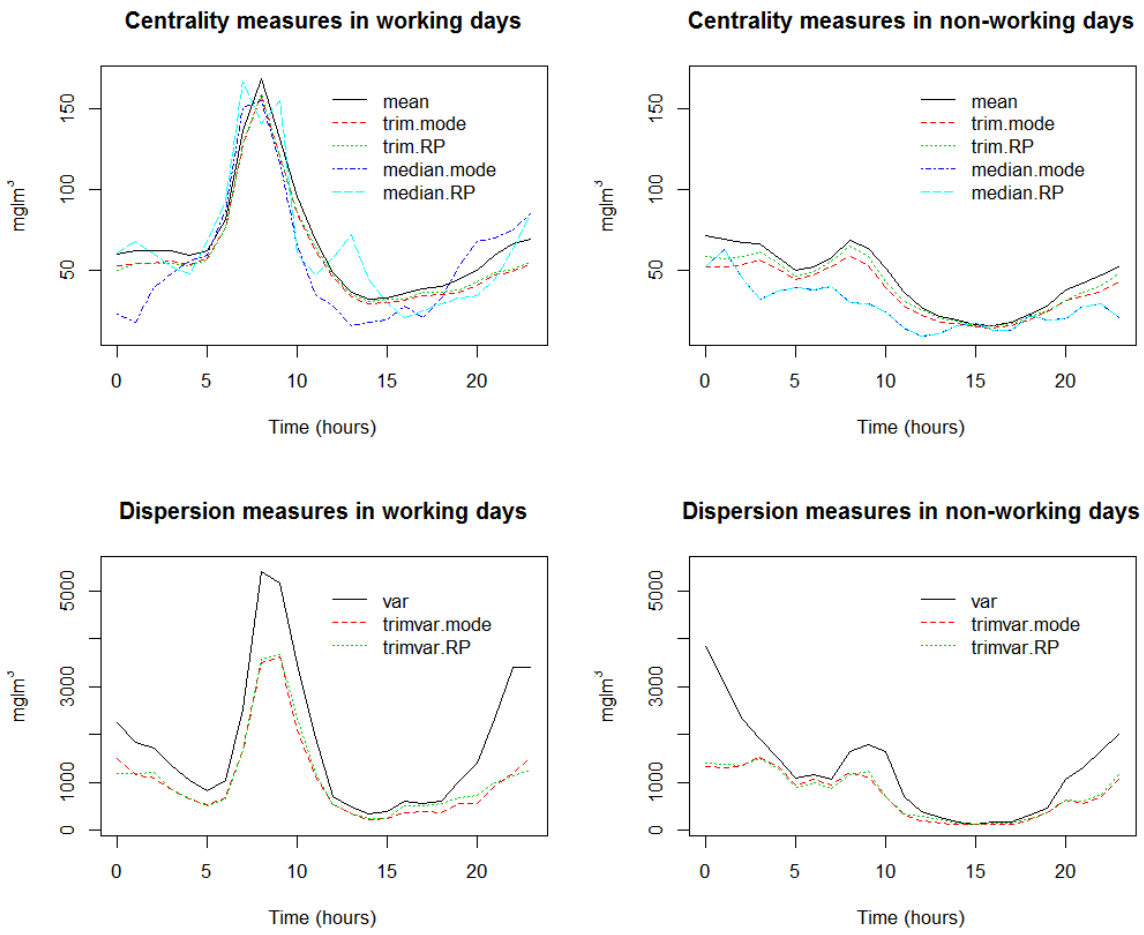

Figure 5: Descriptive statistics for poblenou dataset based on depth: trend measures for working days (top left) and non-working days (top right), dispersion measures for working days (bottom left) and for non-working days (bottom right).

In the previous code (top left of Figure 5) we have employed some shortcut functions as follows:

- For central trend: `func.`*centr*`.`*depth*
- For dispersion measures (or marginal variability measures): `func.trimvar.`*depth*

where *centr* = {`med`, `trim`} indicates whether it has used the median or trimmed mean and *depth* = {`FM`, `mode`, `RP`, `RPD`} indicates the type of depth used. This implementation allows the incorporation of other measures of depth that could be used easily and quickly as measures of location and dispersion.

2.5. Bootstrap replications as dispersion measures

The dispersion of a location statistic for functional data can be estimated by smoothed

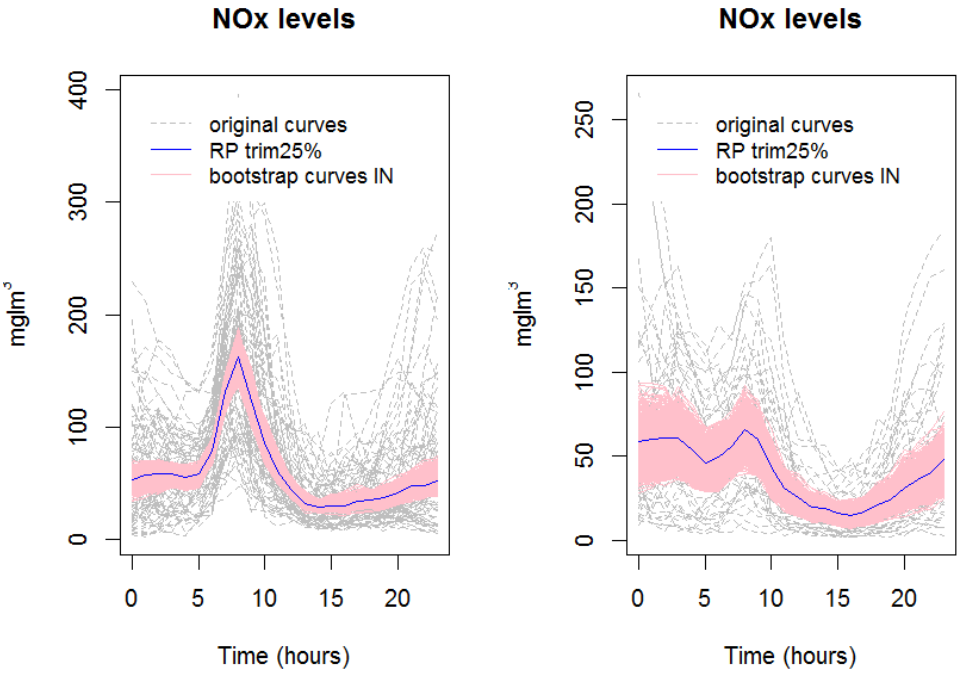

Figure 6: Bootstrap replications of poblenou curves: Using the 25%-trimmed mean statistic for working days curves (left) and for non-working days curves (right).

bootstrap (see Cuevas, Febrero-Bande, and Fraiman 2006, for theoretical details). The `fdata.bootstrap()` computes a confidence ball using bootstrap in the following way:

- Let $X_1(t), \ldots, X_n(t)$ the original data and $T = T(X_1(t), \ldots, X_n(t))$ the sample statistic.
- Calculate the `nb` bootstrap resamples $\{X_1^*(t), \ldots, X_n^*(t)\}$, using the following scheme $X_i^*(t) = X_i(t) + Z(t)$ 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.
- Let $T^{*j} = T(X_1^{*j}(t), ..., X_n^{*j}(t))$ the estimate using the j resample.
- Compute $d(T, T^{*j})$, $j = 1, \ldots, nb$. Define the bootstrap confidence ball of level $1-\alpha$ as $CB(\alpha) = X \in E$ such that $d(T, X) \leq d_\alpha$ being d_α the quantile $(1-\alpha)$ of the distances between the bootstrap resamples and the sample estimate.

The `fdata.bootstrap()` function allows us to define a statistic calculated on the `nb` resamples, control the degree of smoothing by `smo` argument and represent the confidence ball with level $1-\alpha$ as those resamples that fulfill the condition of belonging to $CB(\alpha)$. The `statistic` used by default is the mean (`func.mean()`) but also other depth-based functions can be used (see `help(Descriptive)`). As an example, in next two code lines the confidence ball for location statistic 25%-trimmed mean is computed and plotted in Figure 6 using the random project depth.

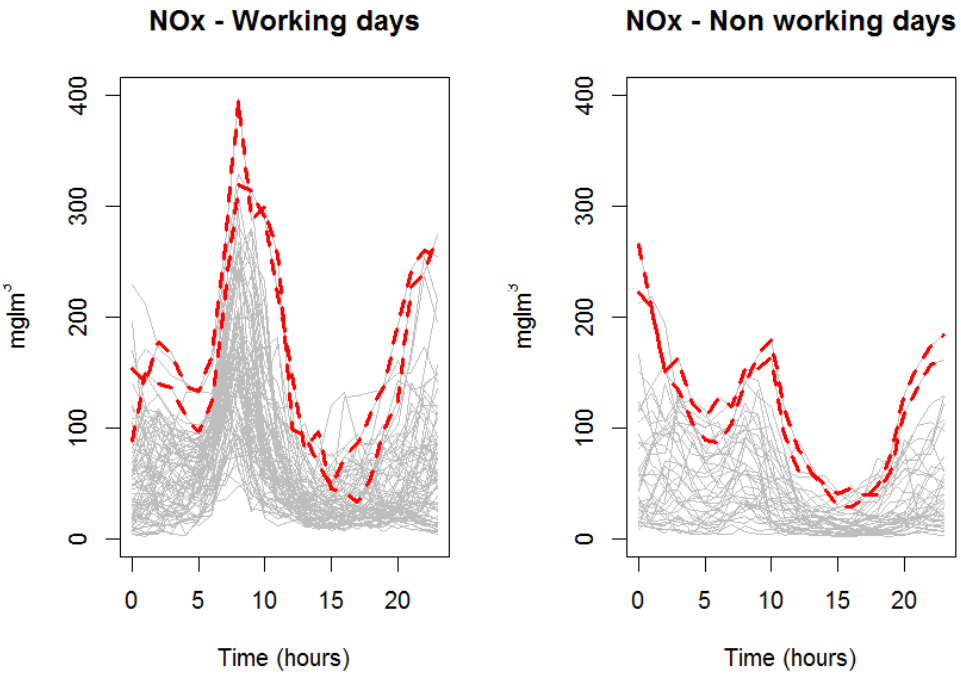

Figure 7: NO_x levels measured by a control station split into two groups (gray lines): Working days (left) and non-working days (right). The red lines correspond to days detected as outliers.

```
R> out.boot1 <- fdata.bootstrap(working, statistic = func.trim.RP, nb = 1000)
R> out.boot2 <- fdata.bootstrap(nonworking, statistic = func.trim.RP,
+    nb = 1000)
```

2.6. Functional outlier detection

In order to identify outliers in functional datasets, Febrero-Bande *et al.* (2008) make use of the fact that depth and outlyingness are inverse notions, so that if an outlier is in the dataset, the corresponding curve will have a significantly low depth. Therefore, a way to detect the presence of functional outliers is to look for curves with lower depths. Two procedures for detecting outliers are implemented: The first one is based on weighting `outliers.depth.pond()` and the second one is based on trimming `outliers.depth.trim()`. Both methods are based on bootstrap and therefore the CPU time can be high.

As an illustration of the outliers detection procedures we replicate the example used by Febrero-Bande *et al.* (2008). Figure 7 shows the result of applying the outliers detection method based on trimming with FM depth for the case of working days and non-working days. The curves in red correspond to those identified as outliers in the original paper.

```
R> out <- outliers.depth.trim(working, dfunc = depth.FM, nb = 1000,
+    smo = 0.1, trim = 0.06)
R> out2 <- outliers.depth.trim(nonworking, dfunc = depth.FM, nb = 1000,
+    smo = 0.1, trim = 0.06)
```

3. Functional regression models

A regression model is said to be "functional" when at least one of the involved variables (either a predictor variable or response variable) is functional. 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 as a functional covariate $X(t) =$ `X` and the water contents as a non-functional covariate (`Z = Water`). We will use the first 165 curves to fit the model. The last 50 records will be used to check the predictions. 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 water content (`Water`) as non-functional variable.

```
R> ind <- 1:165
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 and illustrated with examples for estimating the `Fat` content of the `tecator` dataset. Finally, we show in Section 3.6 a summary focused in prediction of the presented methods.

3.1. Functional linear model with basis representation: `fregre.basis()`

In this section we will assume a functional linear model of type:

$$y = \langle X, \beta \rangle + \epsilon = \frac{1}{\sqrt{T}} \int_T X(t)\beta(t)dt + \epsilon \tag{4}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product on $\mathcal{L}_2$ and ϵ 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 $\beta(t) = \sum b_k \phi_k(t)$. The functional linear model in Equation 4 is estimated by the expression:

$$\hat{y} = \left\langle X, \hat{\beta} \right\rangle \approx \mathbf{C}^\top \psi(\mathbf{t}) \phi^\top(\mathbf{t}) \hat{\mathbf{b}} = \tilde{\mathbf{X}} \hat{\mathbf{b}} \tag{5}$$

where $\tilde{\mathbf{X}}(\mathbf{t}) = \mathbf{C}^\top \psi(\mathbf{t}) \phi^\top(\mathbf{t})$, and $\hat{\mathbf{b}} = (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}})^{-\mathbf{1}} \tilde{\mathbf{X}}^\top y$ and so, $\hat{y} = \tilde{\mathbf{X}} \hat{\mathbf{b}} = \tilde{\mathbf{X}} (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}})^{-\mathbf{1}} \tilde{\mathbf{X}}^\top y = \mathbf{H} y$ where $\mathbf{H}$ is the hat matrix with degrees of freedom: $df = \mathrm{tr}(\mathbf{H})$.

If we want to incorporate a roughness penalty $\lambda > 0$ then the above expression is now: $\hat{y} = \tilde{\mathbf{X}} \hat{\mathbf{b}} = \tilde{\mathbf{X}} (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}} + \lambda \mathbf{R_0})^{-\mathbf{1}} \tilde{\mathbf{X}}^\top y = \mathbf{H}_\lambda y$ 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 function allows covariates of class "`fdata`", "`matrix`", "`data.frame`" or directly covariates of class "`fd`". 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. In addition, the function

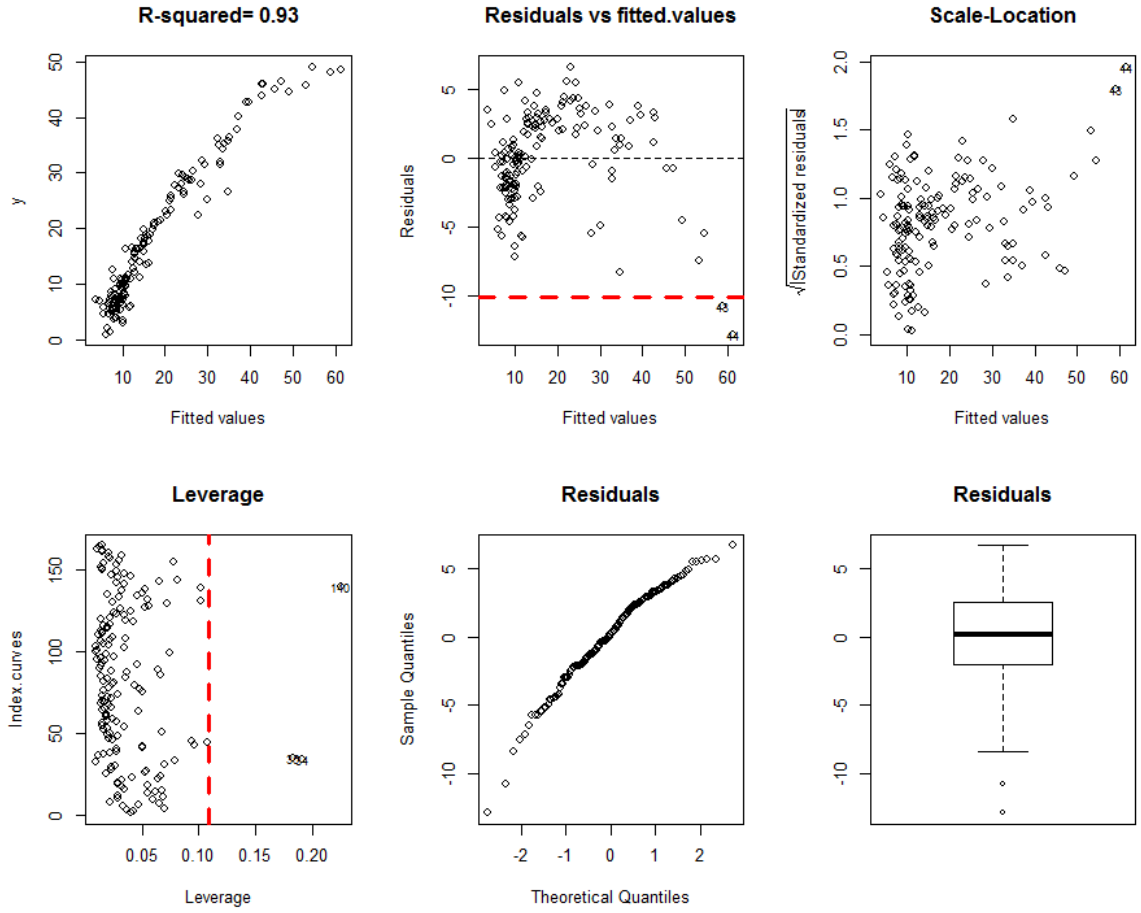

Figure 8: Summary plots for fitted object (`res.basis1`) from a functional linear model with basis representation (`fregre.basis()`).

`fregre.basis.cv()` uses validation criterion defined in Section 2.2 by argument `type.CV` to estimate the number of basis elements and/or the penalized parameter (λ) that best predicts the response.

Going on with the example, the next code illustrates how to estimate the fat contents (`y = Fat`) using a training sample of 1st derivative absorbance curves `X.d1`.

```
R> rangett <- absorp$rangeval
R> basis1 <- create.fourier.basis(rangeval = rangett, nbasis = 5)
R> res.basis1 <- fregre.basis(X.d1, y, basis.x = basis1)
```

The fitted object (`res.basis1`) contains useful information as the smoothing parameter (ν), the degrees of freedom (df), the residual variance (S^2_R): $S^2_R = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 / (n - df)$ (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 8.

3.2. Functional linear model with functional PC basis: `fregre.pc()`

Similarly, Cardot, Ferraty, and Sarda (1999) used a $\{\nu_k\}_{k=1}^{\infty}$ orthonormal basis of functional principal components to represent the functional data as $X_i(t) = \sum_{k=1}^{\infty} \gamma_{ik}\nu_k$ and the functional parameter as $\beta(t) = \sum_{k=1}^{\infty} \beta_k\nu_k$, where $\gamma_{ik} = \langle X_i(t), \nu_k\rangle$ and $\beta_k = \langle \beta, \nu_k\rangle$. Now, the estimation of $\beta(t)$ can be made by a few principal components (PCs) under the assumption that $\beta_k = 0$, for $k > k_n$ and the integral can be approximated by:

$$\hat{y} = \left\langle X, \hat{\beta} \right\rangle \approx \sum_{k=1}^{k_n} \gamma_{ik}\hat{\beta}_k \tag{6}$$

where, $\hat{\beta}_{(1:k_n)} = \left(\frac{\gamma_{.1}^{\top} y}{n\lambda_1}, \ldots, \frac{\gamma_{.k_n}^{\top} y}{n\lambda_{k_n}}\right)$ and $\gamma_{(1:k_n)}$ is the (n,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 is expressed as: $\hat{y} = \mathbf{H}y$ where $\mathbf{H} = \left(\frac{\gamma_{.1}\gamma_{.1}^{\top} y}{n\lambda_1}, \ldots, \frac{\gamma_{.k_n}\gamma_{.k_n}^{\top} y}{n\lambda_{k_n}}\right)$ with degrees of freedom: $df = \mathrm{tr}(\mathbf{H}) = k_n$.

We have implemented the functional principal regression in the function `fregre.pc()`. The call for `tecator` example is shown below:

```
R> res.pc1 <- fregre.pc(X.d1, y, l = 1:6)
```

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 component.

How to select k_n: `fregre.pc.cv()`

The novelty of the procedure used in `fregre.pc.cv()` is that the algorithm selects the principal components that best estimated the response. The selection is done by cross-validation (CV) or model selection criteria (MSC).

- 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: $MSC(k_n) = \log\left[\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \left\langle X_i, \hat{\beta}_{(i,k_n)}\right\rangle\right)^2\right] + p_n\frac{k_n}{n}$ with
  $p_n = \frac{log(n)}{n}$ for `criteria = "SIC"` (by default),
  $p_n = \frac{log(n)}{n-k_n-2}$ for `criteria = "SICc"`,
  $p_n = 2$ for `criteria = "AIC"`,
  $p_n = \frac{2n}{n-k_n-2}$ for `criteria = "AICc"`,

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

In the following example the regression model is fitted using the best selection of first 7 functional principal components (FPCs). The minimum MSC ($MSC = 2.128167$) is achieved with the FPC 1, 2, 7, 3 and 6 using the SIC criterion.

```
R> res.pc2 <- fregre.pc.cv(X.d2, y, kmax = 7)
R> res.pc2$pc.opt

PC1 PC2 PC7 PC3 PC6
  1   2   7   3   6

R> res.pc2$MSC

        PC(1)    PC(2)    PC(3)   PC(4)    PC(5)    PC(6)    PC(7)
rn=0 2.852951 2.258292 2.144443 2.13227 2.128167 2.159107 2.190052
```

Functional linear model with functional PLS basis: `fregre.pls()`

In the previous section, the dimension reduction by FPCs is a good solution for the estimation of the functional linear model. Another good alternative is the use of the criterion that maximizes the covariance between $X(t)$ and the scalar response Y via the partial least squares (PLS) components. Let $\{\tilde{\nu}_k\}_{k=1}^{\infty}$ the functional PLS components (Preda and Saporta 2005) and as before $X_i(t) = \sum_{k=1}^{\infty} \tilde{\gamma}_{ik}\tilde{\nu}_k$ and $\beta(t) = \sum_{k=1}^{\infty} \tilde{\beta}_k\tilde{\nu}_k$. The functional linear model is estimated by:

$$\hat{y} = \left\langle X, \hat{\beta} \right\rangle \approx \sum_{k=1}^{k_n} \tilde{\gamma}_{ik}\tilde{\beta}_k \tag{7}$$

We have implemented the FPLS regression in the function `fregre.pls()` where the PLS factors have been calculated using an alternative formulation of the NIPALS algorithm proposed by Krämer and Sugiyama (2011). The call for `tecator` example and the print of the fitted object are shown below:

```
R> fregre.pls(X.d1, y, l = 1:5)

-Call: fregre.pls(fdataobj = X.d1, y = y, l = 1:5)

-Coefficients:
(Intercept)         PLS1         PLS2         PLS3         PLS4         PLS5
      17.39       491.92       142.44       230.95       134.00       186.78

-R squared:  0.9508922
-Residual variance:  7.927896
```

Functional influence measures: `influence.fdata()`

This section focuses on how to identify influential observations in the FLM discussed in the previous sections. Febrero-Bande, Galeano, and González-Manteiga (2010) studied three statistics for measuring the influence: Cook prediction distance (CP_i), Cook estimation distance (CE_i) and Peña's distance (P_i), respectively.

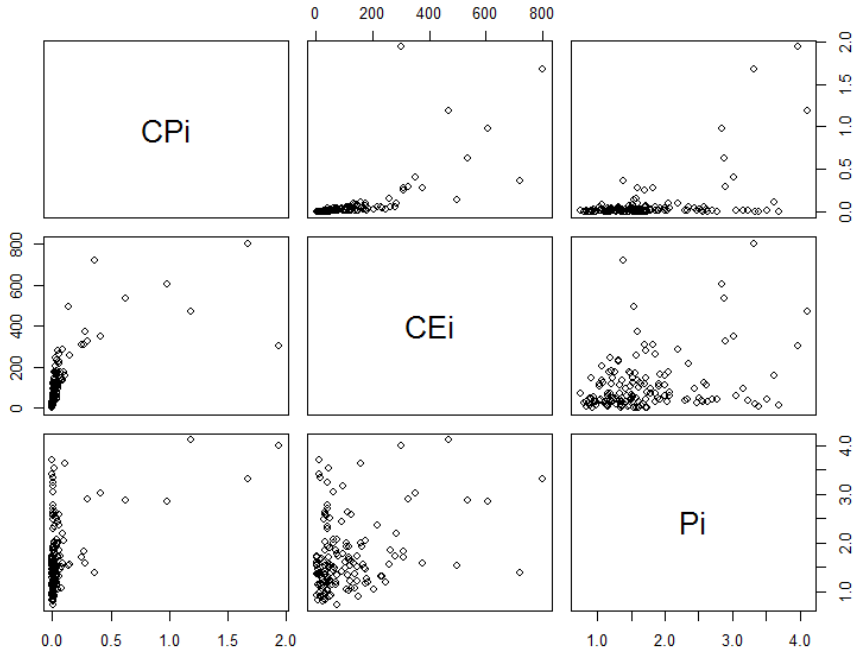

Figure 9: Influence measures for `tecator` dataset calculated from fitted model (`res.basis0`): Matrix of scatter plots for Cook prediction distance (CP_i), Cook estimation distance (CE_i) and Peña's distance (P_i).

1. The functional Cook's measure for prediction (CP_i) detects observations whose deletion may entail important changes in the prediction of the rest of the data. It is defined as

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

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

2. The functional Cook's measure for estimation (CE_i) detects observations whose deletion may entail important changes in the estimation.

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

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

3. The functional Peña's measure for prediction (P_i) detects observations whose prediction is most affected by the deletion of other data.

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

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.basis()`, `fregre.pc()` or `fregre.pls()`), the `influence.fdata()` function is used to obtain the influence measures, see Figure 9.

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 advantage derived from this procedure is that the calculation takes into account the `mue.boot` curves of the functional parameter $\beta(t)$ in the functional Cook's measure for estimation. However, this procedure has a very high computational cost. In order to reduce substantially the computational time we have created the `fregre.bootstrap()` function which is explained below.

Bootstrap for functional linear model

We can study the variability of functional beta parameter β, which is estimated by the previously functional regression models (`fregre.basis()`, `fregre.pc()` and `fregre.pls()`), using the confidence ball (CB) defined as follows,

$$CB(\hat{\beta}_\nu) = \left\{ \beta \in \mathcal{L}_2 : \left\| \beta - \hat{\beta}_\nu \right\| < D_\alpha \right\} \tag{8}$$

where D_α such that $Pr\left(\beta \in CB(\hat{\beta}_\nu)\right) = 1 - \alpha$. To do this, we have followed the smoothed bootstrap procedure proposed by Febrero-Bande *et al.* (2010). Given n scalar responses Y and n functional data X.

1. Fit the functional linear model in Equation 4 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 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 of β obtained in step 1.
5. Fit the model 4 to the smoothed bootstrap $\tilde{Y}^b$ and $\tilde{X}^b$.
6. D_α in Equation 8 is estimated by the $(1-\alpha)$ quantile $\hat{D}_\alpha$ of $\left\| \hat{\beta}_\nu - \hat{\beta}_\nu^i \right\|_{i=1}^B$.
7. Plot the bootstrap replicates $\left\{ \hat{\beta}_\nu^i \right\}_{i=1}^B$ such that $\left\| \hat{\beta}_\nu - \hat{\beta}_\nu^i \right\| < \hat{D}_\alpha$.

In Figure 10, the bootstrap confidence ball is plotted for the three fitted models (`res.basis1`, `res.pc1` and `res.pls1`), with $\left\{ \hat{\beta}_\nu^i \right\}_{i=1}^{1000}$ and $(1-\alpha) = 0.999$. Those replicates belonging to

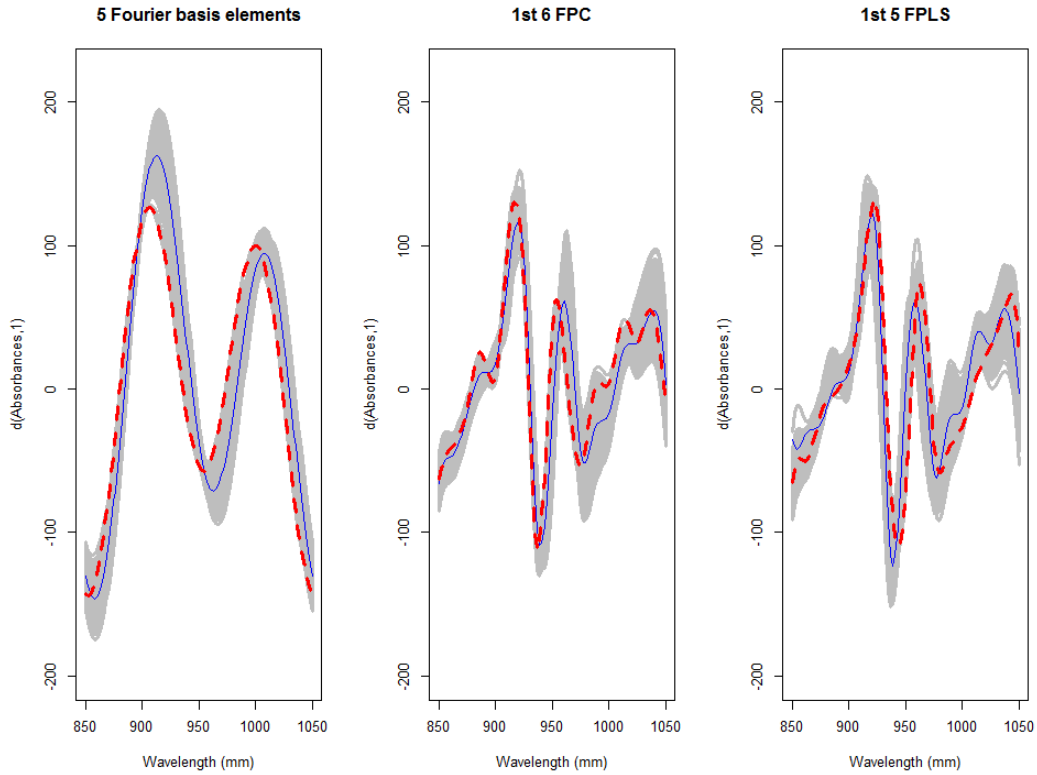

Figure 10: Estimated regression function $\hat{\beta}_\nu$ joint with a 99.9% bootstrap confidence ball by: `fregre.basis()` (left), `fregre.pc()` (center) and `fregre.pls()` (right). Red curve correspond to that resample not in the bootstrap confidence ball.

the confidence ball are drawn in gray whereas the curve falling outside is drawn in red. The parameter $\hat{\beta}_\nu$ (blue line) is clearly significant (different from 0) showing the maximum difference in the first half of wavelength interval.

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

In the example, we have used a small resample `nb = 1000` and we have fixed the parameter `kmax.fix` (5 Fourier basis, 6 FPC and 5 FPLS) to have a reasonable computing time. Still, the calculation procedure is computationally high demanding.

```
R> fregre.bootstrap(res.basis1, nb = 1000, kmax.fix = TRUE, alpha = 0.999)
R> fregre.bootstrap(res.pc1, nb = 1000, kmax.fix = TRUE, alpha = 0.999)
R> fregre.bootstrap(res.pls1, nb = 1000, kmax.fix = TRUE, alpha = 0.999)
```

3.3. Functional linear model with functional and non-functional covariate: `fregre.pls()`

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:

$$y_i = \alpha + \beta_1 Z_i^1 + \cdots + \beta_p Z_i^p + \frac{1}{\sqrt{T_1}} \int_{T_1} X_i^1(t)\beta_1(t)dt + \cdots + \frac{1}{\sqrt{T_q}} \int_{T_q} X_i^q(t)\beta_q(t)dt + \epsilon_i \quad (9)$$

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 ones.

The functional linear model (9) is estimated by the expression:

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

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

The arguments are as follows:

- `formula`: A symbolic description of the model to be fitted.
- `data`: List 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 estimating the functional parameter $\beta(t)$.

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

```
R> ind <- 1:165
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> basis2 <- create.bspline.basis(rangeval = rangett, nbasis = 5)
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"
```

3.4. Non-parametric functional regression model: `fregre.np()`

An alternative to model of Equation 4 is the non-parametric functional regression studied by Ferraty and Vieu (2006). In this case, the regression model is written as

$$y_i = r(X_i(t)) + \epsilon_i, \quad (10)$$

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 $K(\cdot)$ is an kernel function, h is the smoothing parameter and $d(\cdot, \cdot)$ is a metric or a semi-metric. Now, the kernel is applied to a metric or semi-metrics that provides non-negative values, so it is common to use asymmetric kernels.

This procedure is implemented in the `fregre.np()` function and some of its arguments are:

- `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(...,lp = 2)`).
- `type.S`: Type of smoothing matrix S, by default Nadaraya Watson (`S.NW()`).

Again, the function `fregre.np.cv()` is used to choose the smoothing parameter h by the validation criterion described in Section 2.2.

- `type.CV`: Type of validation criterion, by default GCV criterion (`GCV.S()`).

The code for the `tecator` example is:

```
R> fregre.np(X.d1,y)

-Call: fregre.np(fdataobj = X.d1, y = y)

-Bandwidth (h):  0.006498587
-R squared:  0.9873025
-Residual variance:  3.036685
```

3.5. Semi-functional partially linear model: `fregre.plm()`

An extension of the non-parametric functional regression models is the semi-functional partial linear model proposed in Aneiros-Pérez and Vieu (2006). This model uses a non-parametric kernel procedure as that described in Section 3.4. The output y is scalar. A functional covariate X and a multivariate non-functional covariate Z are considered.

$$y = r(X) + \sum_{j=1}^{p} Z_j \beta_j + \epsilon \tag{11}$$

The unknown smooth real function r is estimated by means of

$$\hat{r}_h(X) = \sum_{i=1}^{n} w_{n,h}(X, X_i)(Y_i - Z_i^{\top}\hat{\beta}_h)$$

where W_h is the weight function: $w_{n,h}(X, X_i) = \frac{K(d(X,X_i)/h)}{\sum_{j=1}^{n} K(d(X,X_j)/h)}$ with smoothing parameter h, an asymmetric kernel $K(\cdot)$ and a metric or semi-metric $d(\cdot, \cdot)$. 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 $\mathcal{L}_2$ (`metric = metric.lp` with `p = 2`). The unknown parameters β_j for the multivariate non-functional covariates are estimated by means of $\hat{\beta}_j = (\tilde{Z}_h^\top \tilde{Z}_h)^{-1} \tilde{Z}_h^\top \tilde{Z}_h$ where $\tilde{Z}_h = (I - W_h)Z$ with the smoothing parameter h. The errors ϵ are independent, with zero mean, finite variance σ^2 and $E[\epsilon|Z_1, \ldots, Z_p, X(t)] = 0$.

Coming back to the example of Section 3.3, the fitted model for the case of a real variable `Z = Water` and the second derivative of the absorbance curves (`X.d2`) as functional covariate can be obtained by:

```
R> res.plm2 <- fregre.plm( Fat ~ Water + X.d2, ldata)

-Call: fregre.plm(formula = Fat ~ Water + X.d2, data = ldata)

-Coefficients:
          Estimate  Std. Error      t value    Pr(>|t|)
Water   -5.934e-01   6.463e-02   -9.181e+00   2.085e-13

-Bandwidth (h):  0.0003614096
-R squared:  0.9972631
-Residual variance:  0.6986455
```

3.6. Prediction methods for functional regression model fits

Once the model is estimated we can obtain predictions by means of:

- `predict.fregre.fd()` corresponds to the model fitted from `fregre.basis()`, `fregre.pc()`, `fregre.pls()` or `fregre.np()` functions.
- `predict.fregre.lm()` corresponds to the model fitted from `fregre.lm()` function.
- `predict.fregre.plm()` corresponds to the model fitted from `fregre.plm()` function.

A sample test of the last 50 curves of absorbance (or one of the two first derivatives) and `Water` content can be used to predict the `Fat` content. We provide below the complete code for the prediction process.

```
R> newy <- matrix(tecator$y$Fa[-ind], ncol = 1)
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 = "bspline")
R> pred.basis2 <- predict.fregre.fd(res.basis2, newX.d2)
R> res.np2 <- fregre.np.cv(X.d1, y, metric = semimetric.deriv)
R> pred.np2 <- predict.fregre.fd(res.np2, newX.d1)
R> newldata <- list("df" = newdataf, "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)
```

Function	*df*	R^2	S_R^2	MEP
`fregre.basis(X.d1, Fat)`	6.0	0.928	11.470	0.0626
`fregre.basis.cv(X.d2, Fat)`	12.0	0.965	5.817	0.0566
`fregre.pc(X.d1, Fat)`	7.0	0.950	7.975	0.0580
`fregre.pc(X.d2, Fat)`	6.0	0.954	7.239	0.0556
`fregre.pls(X.d1, Fat)`	8.4	0.951	7.928	0.0567
`fregre.pls(X.d2, Fat)`	6.5	0.962	6.015	0.0487
`fregre.lm(Fat ~ X.d1 + Water)`	7.0	0.987	2.140	0.0097
`fregre.lm(Fat ~ X.d2 + Water)`	7.0	0.986	2.251	0.0119
`fregre.np(X.d1, Fat)`	59.3	0.987	3.037	0.0220
`fregre.np(X.d2, Fat)`	68.1	0.996	1.128	0.0144
`fregre.plm(Fat ~ X.d1 + Water)`	61.0	0.996	0.914	0.0090
`fregre.plm(Fat ~ X.d2 + Water)`	66.0	0.997	0.699	0.0115

Table 1: Results for functional regression models. *df* degrees of freedom, R^2, S_R^2 residual variance, and mean square error of prediction (MEP).

```
R> pred.lm1 <- predict.fregre.lm(res.lm1, newldata)
R> res.plm1 <- fregre.plm(f1, ldata)
R> pred.plm1 <- predict.fregre.plm(res.plm1, newldata)
```

We make predictions for the rest of models fitted in this paper (code not shown). Following the ideas by Aneiros-Pérez and Vieu (2006), we calculated the mean square 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 1 resumes the statistics of the fitted models and their predictions.

4. Conclusion

The package **fda.usc** presented in this paper is the result of the integration of our codes with other procedures from different authors, such as the package **fda** or the functions from the STAPH group. One major advantage of this software is that it avoids the need of a basis representation of functional data. Using the new class "`fdata`", the proposed methods can represent the functional data using discretized versions in a given grid of points.

The **fda.usc** package includes most of the methods recently developed for exploratory functional data analysis and for functional regression with scalar response. This package also incorporates other utilities for statistical computing within the field of functional data analysis (FDA). Some useful additions are:

- Generalized functional linear models (FGLM): `fregre.glm()`.
- Functional analysis of variance: `anova.RPm()`.
- Functional supervised classification: `classif.knn()`, `classif.kernel()`, and `classif.glm()`.
- Functional non-supervised classification: `kmeans.fd()`.

- Other utilities and auxiliary functions, as the `cond.F()` function that calculates the conditional distribution function of a scalar response with functional data.

The **fda.usc** package is an attempt to obtain an integrated framework for FDA. It is under continuous development therefore updates will be available in CRAN. We are working to incorporate new depth measures and methods for functional regression and classification. Further information, as script files and a beta version of package, is also available at the project website whose URL http://eio.usc.es/pub/gi1914/, also given in the `DESCRIPTION` file.

Acknowledgments

We would like to thank Antonio Cuevas and two anonymous referees for their helpful comments and suggestions, which have substantially improved the original manuscript. 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 – Modelos de optimización, decisión, estadística y aplicaciones.

References

Aneiros-Pérez G, Vieu P (2006). "Semi-Functional Partial Linear Regression." *Statistics & Probability Letters*, **76**(11), 1102–1110.

Cardot H, Ferraty F, Sarda P (1999). "Functional Linear Model." *Statistics & Probability Letters*, **45**(1), 11–22.

Crainiceanu C, Reiss P, Goldsmith J, Huang L, Huo L, Scheipl F (2012). ***refund**: Regression with Functional Data*. R package version 0.1-6, URL http://CRAN.R-project.org/package=refund.

Crainiceanu CM, Goldsmith AJ (2010). "Bayesian Functional Data Analysis Using **WinBUGS**." *Journal of Statistical Software*, **32**(11), 1–33. URL http://www.jstatsoft.org/v32/i11/.

Cuevas A, Febrero-Bande M, Fraiman R (2006). "On the Use of the Bootstrap for Estimating Functions with Functional Data." *Computational Statistics & Data Analysis*, **51**(2), 1063–1074.

Cuevas A, Febrero-Bande M, Fraiman R (2007). "Robust Estimation and Classification for Functional Data via Projection-Based Depth Notions." *Computational Statistics*, **22**(3), 481–496.

Dou W (2009). ***MFDF**: Modeling Functional Data in Finance*. R package version 0.0-2, URL http://CRAN.R-project.org/package=MFDF.

Febrero-Bande M, Galeano P, González-Manteiga W (2010). "Measures of Influence for the Functional Linear Model with Scalar Response." *Journal of Multivariate Analysis*, **101**(2), 327–339.

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 F, Vieu P (2006). *Nonparametric Functional Data Analysis*. Springer-Verlag, New York.

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

Giraldo R, Delicado P, Mateu J (2010). ***geofd**: Spatial Prediction for Function Value Data*. R package version 0.4.6, URL http://CRAN.R-project.org/src/contrib/Archive/geofd/.

Härdle W (1990). *Applied Nonparametric Regression*. Cambridge University Press, Cambridge.

Hyndman RJ, Shang HL (2012). ***ftsa**: Functional Time Series Analysis*. R package version 3.5, URL http://CRAN.R-project.org/package=ftsa.

Krämer N, Sugiyama M (2011). "The Degrees of Freedom of Partial Least Squares Regression." *Journal of the American Statistical Association*, **106**(494), 697–705.

Liu B (2011). ***PACE** Package*. R package version 0.1-0, URL http://CRAN.R-project.org/src/contrib/Archive/PACE/.

Markussen B (2011). ***fdaMixed**: Functional Data Analysis in a Mixed Model Framework*. R package version 0.1, URL http://CRAN.R-project.org/package=fdaMixed.

Peng J, Paul D (2011). ***fpca**: Restricted MLE for Functional Principal Components Analysis*. R package version 0.2-1, URL http://CRAN.R-project.org/package=fpca.

Preda C, Saporta G (2005). "PLS Regression on a Stochastic Process." *Computational Statistics & Data Analysis*, **48**(1), 149–158.

Ramsay JO, Silverman BW (2002). *Applied Functional Data Analysis: Methods and Case studies*. Springer-Verlag, New York.

Ramsay JO, Silverman BW (2005). *Functional Data Analysis*. 2nd edition. Springer-Verlag, New York.

Ramsay JO, Wickham H, Graves S, Hooker G (2012). ***fda**: Functional Data Analysis*. R package version 2.2.8, URL http://CRAN.R-project.org/package=fda.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Shang HL, Hyndman RJ (2012). ***rainbow**: Rainbow Plots, Bagplots and Boxplots for Functional Data*. R package version 3.0, URL http://CRAN.R-project.org/package=rainbow.

Wasserman L (2006). *All of Nonparametric Statistics*. Springer-Verlag, New York.

Affiliation:

Manuel Febrero-Bande, Manuel Oviedo de la Fuente
Departamento de Estadística e Investigación Operativa
Facultad de Matemáticas
Rúa Lope Gómez de Marzoa s/n
15782 Santiago de Compostela, Spain
E-mail: manuel.febrero@usc.es, manuel.oviedo@usc.es
URL: http://eio.usc.es/pub/MAESFE/, http://eio.usc.es/pub/gi1914/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 51, Issue 4 *Submitted:* 2011-04-27
October 2012 *Accepted:* 2012-09-21