Robust Functional Linear Regression Models

Ufuk Beyaztas
Marmara University

Han Lin Shang
Macquarie University

Abstract

With advancements in technology and data storage, the availability of functional data whose sample observations are recorded over a continuum, such as time, wavelength, space grids, and depth, progressively increases in almost all scientific branches. Accordingly, the functional linear regression models, including scalar-on-function and function-on-function, have become popular tools for exploring the functional relationships between the scalar response-functional predictors and functional response-functional predictors. However, most of the existing estimation strategies are based on the non-robust estimators that are seriously hindered by outlying observations, which are common in empirical applications. In the case of outliers, the non-robust methods lead to undesirable estimation and prediction results. Using a readily-available R package **robflreg**, this paper presents several robust methods build upon the functional principal component analysis for modeling and predicting scalar-on-function and function-on-function regression models in the presence of outliers. The methods are demonstrated via simulated datasets.

Keywords: function-on-function linear regression; functional principal component analysis; robust estimation; scalar-on-function linear regression.

Introduction

Our aim with this paper is to present a hands-on tutorial for the implantation of readily-available R package **robflreg**. This package is designed for robustly modeling and predicting scalar-on-function and function-on-function linear regression models (abbreviated as SFLRM and FFLRM, respectively). This article is motivated by recent advances in data collection tools, causing (ultra) high dimensional and complex data structures, such as ultra-dense curves.

In the last few decades, the interest and need for developing statistical methods to analyze functional data has been tremendously increased. Consult Ramsay and Dalzell (1991), Ramsay and Silverman (2002, 2006), Ferraty and Vieu (2006), Horváth and Kokoszka (2012), Cuevas (2014), Hsing and Eubank (2015), and Kokoszka and Reimherr (2017) for many theoretical developments and applications in functional data analysis tools. Among many others, the SFLRM, where the response is scalar-valued and predictor(s) consist of random functions, and FFLRM, where both the response and predictor(s) consist of random curves, have received considerable attention among researchers to explore the functional relationship between a scalar response-functional predictors and a functional response-functional predictors, respectively. Consult Cardot, Ferraty, and Sarda (1991, 2003), James (2002), Reiss and Ogden (2007), Chen, Hall, and Müller (2011), Jiang and Wang (2011), Dou, Pollard, and Zhou (2012), and Beyaztas and Shang (2022) for the SFLRM and Yao, Müller, and Wang (2005), Harezlak, Coull, Laird, Magari, and Christiani (2007), Şentürk and Müller (2008),

Matsui, Kawano, and Konishi (2009), Ivanescu, Staicu, Scheipl, and Greven (2015), Chiou, Yang, and Chen (2016), and Beyaztas and Shang (2020) for the FFLRM.

Most of the existing methods developed to estimate the SFLRM and FFLRM are non-robust to outlying observations, which are generated by a stochastic process with a distribution different from that of the vast majority of the remaining observations (see, e.g., Raña, Aneiros, and Vilar 2015). In the case of outliers, the non-robust methods produce biased estimates; thus, predictions obtained from the fitted model become unreliable (see, e.g., Zhu, Brown, and Morris 2011; Maronna and Yohai 2013; Shin and Lee 2016; Kalogridis and Aelst 2019; Boente, Salibian-Barrera, and Vena 2020; Hullait, Leslie, Pavlidis, and King 2021; Beyaztas and Shang 2022). In this paper, we provide a hands-on tutorial for the implementation of several robust approaches, which are readily available in the R package **robflreg**, for robustly modeling and predicting the SFLRM and FFLRM in the presence of outliers.

The methods available in the **robflreg** package are centered on the robust functional principal component analysis (RFPCA) approach of Bali, Boente, Tyler, and Wang (2011). It uses the robust projection pursuit approach of Croux and Ruiz-Gazen (1996) combined with a robust scale estimator to produce functional principal components and the corresponding principal component scores. With this approach, the infinite-dimensional SFLRM and FFLRM are projected onto a finite-dimensional space of RFPCA bases. Then, for the SFLRM, the robust estimation methods, including the least trimmed squares (LTS) of Rousseeuw (1984), MM-type regression estimator (MM) of Yohai (1987) and Koller and Stahel (2011), S estimator, and the tau estimator of Salibian-Barrera, Willems, and Zamar (2008), are used to estimate the parameter vector of the regression model of the scalar-valued response on the robust principal component scores of functional predictors. For the FFLRM, on the other hand, the robust estimation methods, including the minimum covariance determinant estimator (MCD) of Rousseeuw, Driessen, Aelst, and Agullo (1984), multivariate least trimmed squares estimator (MLTS) of Bali, Boente, Tyler, and Wang (2008), MM estimator of Kudraszow and Moronna (2011), S estimator of Bilodeau and Duchesne (2000), and the tau estimator of Ben, Martinez, and Yohai (2006), are used to estimate the parameter matrix of the regression model between the robust principal component scores of the functional response and functional predictor variables. Besides the robust procedures, the package **robflreg** allows to obtain the non-robust estimation of the functional linear regression models using the classical functional principal component analysis (FPCA) of Ramsay and Silverman (2006) and the least-squares estimator.

The remainder of this paper is organized as follows. The SFLRM and FFLRM, as well as the techniques used for modeling and predicting these regression models, are reviewed, and they are implemented using the **robflreg** package. Conclusions are given in the end.

Functional linear regression models

In this Section, we present the SFLRM and FFLRM, respectively.

The SFLRM

We consider a random sample $\{Y_i, \boldsymbol{\mathcal{X}}_i(s) : i = 1, \ldots, n\}$ from the pair $(Y, \boldsymbol{\mathcal{X}})$, where $Y \in \mathbb{R}$ is the scalar response and $\boldsymbol{\mathcal{X}} = [\mathcal{X}_1(s), \ldots, \mathcal{X}_P(s)]^\top$ with $\mathcal{X}_p(s) \in \mathcal{L}_2[0, \mathcal{I}]$, $\forall\ p = 1, \ldots, P$ is the vector of P set of functional predictors whose sample elements are denoted by curves belonging to $\mathcal{L}_2$ Hilbert

space, denoted by $\mathcal{H}$, with bounded and closed interval $s \in \mathcal{I}$. We assume that the functional predictors $\mathcal{X}_p(s)$, for $p = 1, \ldots, P$, have second-order finite moments, i.e., $\mathrm{E}[\|\mathcal{X}_p(s)\|] < \infty$. Without loss of generality, we also assume that Y and $\mathcal{X}_p(s)$, for $p = 1, \ldots, P$, are mean-zero processes, so that $\mathrm{E}[Y] = \mathrm{E}[\mathcal{X}_p(s)] = 0$ and $s \in [0, 1]$. Then, the SFRM is defined as follows:

$$Y_i = \int_0^1 \boldsymbol{\mathcal{X}}_i^\top(s)\boldsymbol{\beta}(s)ds + \epsilon_i, \tag{1}$$

where $\beta_p(s) \in \mathcal{L}_2[0, 1]$ is the regression coefficient function linking Y with $\mathcal{X}_p(s)$, and $\boldsymbol{\beta}(s) = [\beta_1(s), \ldots, \beta_P(s)]^\top \in \mathcal{L}_2^P[0, 1]$, and ϵ_i is the error term which is assumed to follow a Gaussian distribution with mean-zero and variance σ^2.

Simulation of a dataset for the SFLRM

The interface `generate.sf.data()` in the package **robflreg** allows to simulate a dataset for the SFRM (1) as follows:

```
generate.sf.data(n, n.pred, n.gp)
```

Here, the argument `n` denotes the number of observations for each variable to be generated, `n.pred` denotes the number of functional predictors to be generated, and `n.gp` denotes the number of grid points, i.e., a fine grid on the interval $[0, 1]$. In the data generation process, first, `generate.sf.data()` simulates the functional predictors based on the following process:

$$\mathcal{X}(s) = \sum_{j=1}^{5} \kappa_j \nu_j(s),$$

where κ_j is a vector generated from a Normal distribution with mean one and variance $\sqrt{a}j^{-3/2}$, where a is is a uniformly generated random number between 1 and 4, and

$$\nu_j(s) = \sin(j\pi s) - \cos(j\pi s).$$

The regression coefficient functions are generated from a coefficient space that includes ten different functions such as $b\sin(2\pi t)$ and $b\cos(2\pi t)$, where b is generated from a uniform distribution between 1 and 3. The error process is generated from the standard normal distribution. Finally, the scalar response is obtained using (1). A graphical display of the generated dataset with five functional predictors and `n = 400` observations at 101 equally spaced point in the interval $[0, 1]$ obtained by `generate.sf.data()` is presented in Figure 1. This Figure can be produced by the following code:

```
library(robflreg)
library(fda.usc)
set.seed(123)

# Generate a dataset with five functional predictors and 400
# observations at 101 equally spaced point in the interval [0, 1]
# for each variable for the scalar-on-function regression model
sim.data <- generate.sf.data(n = 400, n.pred = 5, n.gp = 101)
```

```
# Response variable
Y <- sim.data$Y
# Predictors
X <- sim.data$X
# Regression coefficient functions
coeffs <- sim.data$f.coef
# Plot the scalar response
plot(Y, type = "p", pch = 16, xlab = "Index", ylab = "", main = "Response")
# Plot the first functional predictor
fX1 <- fdata(X[[1]], argvals = seq(0, 1, length.out = 101))
plot(fX1, lty = 1, ylab = "", xlab = "Grid point",
main = expression(X[1](s)), mgp = c(2, 0.5, 0))
```

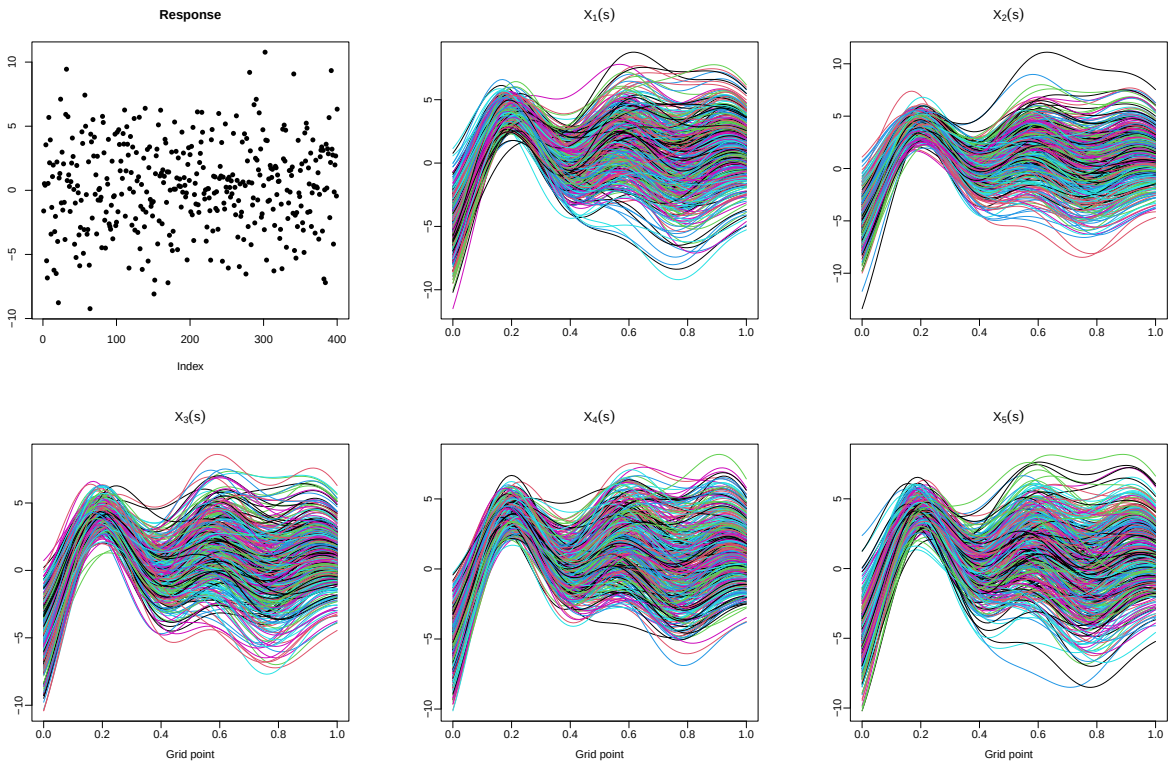

Figure 1: Plots of the simulated scalar response and functional predictor variables.

The FFLRM

Let us consider a random sample $\{\mathcal{Y}_i(t), \boldsymbol{\mathcal{X}}_i(s)\colon\ i = 1, 2, \ldots n\}$ from the pair $(\mathcal{Y}, \boldsymbol{\mathcal{X}})$, where $\mathcal{Y} \in \mathcal{L}_2[0,1]$ is the functional response and $\boldsymbol{\mathcal{X}} = [\mathcal{X}_1(s), \ldots, \mathcal{X}_P(s)]^\top$ with $\mathcal{X}_p(s) \in \mathcal{L}_2[0,1]$, $\forall\ p = 1, \ldots, P$ is the vector of P set of functional predictors. We assume that the functional response and functional predictors have second-order finite moments, i.e., $\mathrm{E}[\|\mathcal{Y}(t)\|] = \mathrm{E}[\|\mathcal{X}_p(s)\|] < \infty$, for $p = 1, \ldots, P$. Without loss of generality, we also assume that both $\mathcal{Y}(t)$ and $\mathcal{X}_p(s)$, for $p = 1, \ldots, P$, are mean-zero

processes, so that $\mathrm{E}[Y(t)] = \mathrm{E}[\mathcal{X}_p(s)] = 0$. Then, the FFRM is defined as follows:

$$Y_i(t) = \int_0^1 \boldsymbol{\mathcal{X}}_i^\top(s)\boldsymbol{\beta}(s,t)dsdt + \epsilon_i(t), \tag{2}$$

where $\beta_p(s,t) \in \mathcal{L}_2[0,1]$ is the bivariate regression coefficient function linking $\mathcal{Y}(t)$ with $\mathcal{X}_p(s)$, and $\boldsymbol{\beta}(s,t) = [\beta_1(s,t), \ldots, \beta_P(s,t)]^\top \in \mathcal{L}_2^P[0,1]$, and $\epsilon_i(t) \in \mathcal{L}_2[0,1]$ is the error term which is assumed to be independent of $\mathcal{X}_p(s)$, for $p = 1, \ldots, P$ and $\mathrm{E}[\epsilon_i(t)] = 0$.

Simulation of a dataset for the FFLRM

The **robflreg** package with the interface `generate.ff.data()` allows for simulation a dataset for the FFLRM as follows:

```
generate.ff.data(n.pred, n.curve, n.gp)
```

In this interface, `n.pred` denotes the number of functional predictors to be generated, `n.curve` denotes the number of functions for each functional variable to be generated, and `n.gp` denotes the number of grid points, i.e., a fine grid on the interval $[0,1]$. When generating a dataset, first, the interface `generate.ff.data()` first simulates the functional predictors via the following process:

$$\mathcal{X}(s) = \sum_{j=1}^{5} \kappa_j \nu_j(s),$$

where κ_j is a vector generated from a Normal distribution with mean one and variance $\sqrt{a}j^{-1/2}$, where a is is a uniformly generated random number between 1 and 4, and

$$\nu_j(s) = \sin(j\pi s) - \cos(j\pi s).$$

The bivariate regression coefficient functions are generated from a coefficient space that includes ten different functions such as $b\sin(2\pi s)\sin(\pi t)$ and $be^{-3(s-0.5)^2}e^{-4(t-1)^2}$, where b is generated from a uniform distribution between 1 and 3. The error process $\epsilon(t)$, on the other hand, is generated from the Ornstein-Uhlenbeck process:

$$\epsilon(t) = l + [\epsilon_0(t) - l]e^{-\theta t}\sigma \int_0^t e^{-\theta(t-u)}dW_u,$$

where $l, \theta > 0, \sigma > 0$ are real constants, $\epsilon_0(t)$ is the initial value of $\epsilon(t)$ taken from W_u, and W_u is the Wiener process. A graphical display of the generated dataset with five functional predictors and `n = 200` observations at 101 equally spaced point in the interval $[0,1]$ obtained by `generate.ff.data()` is presented in Figure 2. This Figure can be produced by the following code:

```
library(robflreg)
library(fda.usc)
set.seed(123)

# Generate a dataset with five functional predictors and 400
# observations at 101 equally spaced point in the interval [0, 1]
```

```
# for each variable for the function-on-function regression model
sim.data <- generate.ff.data(n.pred = 5, n.curve = 200, n.gp = 101)

# Response variable
Y <- sim.data$Y
# Predictors
X <- sim.data$X
# Regression coefficient functions
coeffs <- sim.data$f.coef
# Plot the scalar response
fY <- fdata(Y, argvals = seq(0, 1, length.out = 101))
plot(fY, lty = 1, ylab = "", xlab = "Grid point",
main = "Response", mgp = c(2, 0.5, 0))
# Plot the first functional predictor
fX1 <- fdata(X[[1]], argvals = seq(0, 1, length.out = 101))
plot(fX1, lty = 1, ylab = "", xlab = "Grid point",
main = expression(X[1](s)), mgp = c(2, 0.5, 0))
```

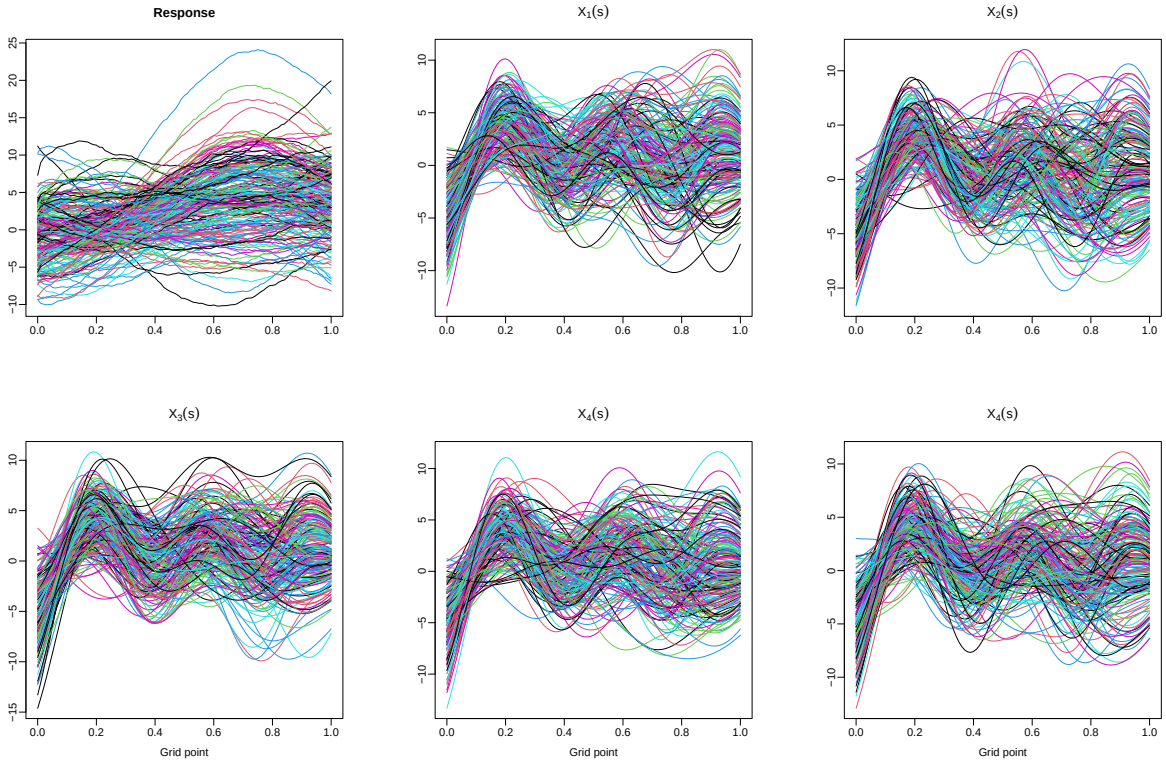

Figure 2: Plots of the simulated functional response and functional predictor variables.

Estimation

We first review the classical and robust FPCA methods. Then, we will focus on the robust estimation of the SFLRM and FFLRM.

Functional principal component analysis (FPCA)

For a functional random variable $\mathcal{X}(s)$, let us denote its covariance function by $\mathcal{C}(s_1, s_2) = \text{Cov}[\mathcal{X}(s_1),\ \mathcal{X}(s_2)]$ satisfying $\int_0^1 \int_0^1 \mathcal{C}(s_1, s_2) ds_1 ds_2 < \infty$. Then, by Mercer's Theorem, the following representation holds:

$$\mathcal{C} = \sum_{k=1}^{\infty} \kappa_k \psi_k(s_1)\psi_k(s_2), \quad \forall s_1, s_2 \in [0, 1],$$

where $\{\psi_k(s) : k = 1, 2, \ldots\}$ are orthonormal bases of eigenfunctions in $\mathcal{L}_2[0, 1]$ corresponding to the non-negative eigenvalues $\{\kappa_k : k = 1, 2, \ldots\}$ with $\kappa_k \geq \kappa_{k+1}$. In practice, most of the variability in functional variables can be captured via a finite number of the first K eigenfunctions, and thus, the covariance function of a functional variable is estimated using a pre-determined truncation constant K. In addition, the orthonormal bases of eigenfunctions are unknown in practice, and thus, they are approximated via a suitable basis expansion method like B-spline, which is used in the **robflreg** package.

The RFPCA of Bali *et al.* (2011) follows a similar structure as the classical FPCA but it uses a robust scale functional instead of variance. Now let $\|\alpha\|^2 = \langle \alpha, \alpha \rangle$ denote the norm generated by the inner product $\langle \cdot, \cdot \rangle$. Also, let $\mathcal{F}[\alpha]$ denote the distribution of $\langle \alpha, \mathcal{X} \rangle$ where $\mathcal{F}$ is the distribution of $\mathcal{X}$. Then, for a given M-scale functional $\sigma_M(\mathcal{F})$, the orthonormal bases of eigenfunctions defined by Bali *et al.* (2011) are as follows:

$$\begin{cases} \psi_k(\mathcal{F}) = \underset{\|\alpha\|^2=1}{\arg\max}\ \sigma_M(\mathcal{F}[\alpha]), & k = 1, \\ \psi_k(\mathcal{F}) = \underset{\|\alpha\|^2=1, \alpha \in \mathcal{B}_k}{\arg\max}\ \sigma_M(\mathcal{F}[\alpha]), & k \geq 2, \end{cases}$$

where $\mathcal{B}_k = \{\alpha \in \mathcal{L}_2[0, 1] : \langle \alpha, \psi_k(\mathcal{F}) \rangle = 0,\ 1 \leq k \leq\ K - 1\}$. The k-th largest eigenvalue is given by:

$$\kappa_k(\mathcal{F}) = \sigma_M^2(\mathcal{F}[\psi_k]) = \max_{\|\alpha\|^2=1, \alpha \in \mathcal{B}_k} \sigma_M^2(\mathcal{F}[\alpha]).$$

Denote by $\sigma_M(\mathcal{F}_n[\alpha])$ the functional for σ_M. Let $s_n^2 : \mathcal{L}_2[0, 1] \to \mathbb{R}$ denote the function of empirical M-scale functional such that $s^2(\alpha) = \sigma_M^2(\mathcal{F}[\alpha])$. Then, the RFPCA estimates of the orthonormal bases of eigenfunctions for $\mathcal{X}(s)$ are given by

$$\begin{cases} \widehat{\psi}_k(s) = \underset{\|\alpha\|^2=1}{\arg\max}\ s_n(\alpha), & k = 1, \\ \widehat{\psi}_k(s) = \underset{\alpha \in \widehat{\mathcal{B}}_k}{\arg\max}\ s_n(\alpha), & k \geq 2, \end{cases}$$

where $\widehat{\mathcal{B}}_k = \left\{\alpha \in \mathcal{L}_2[0, 1] : \|\alpha\| = 1, \langle \alpha, \widehat{\psi}_k \rangle = 0,\ \forall\ 1 \leq k \leq\ K - 1\right\}$. The corresponding eigenvalues, on the other hand, are given by

$$\widehat{\kappa}_k = s_n^2(\widehat{\psi}_k), \quad k \geq 1.$$

Main RFPCA function and its arguments

The main function to obtain the robust estimates of functional principal components and the corresponding principal component scores is called `getPCA()`:

```
getPCA(data, nbasis, ncomp, gp, emodel = c("classical", "robust"))
```

In the `getPCA()` interface, the data is provided in the `data` argument as a matrix. `nbasis` denotes the number of B-spline basis expansion functions used to approximate the robust functional principal components. `ncomp` specifies the number of functional principal components to be computed. The grid points of the functional data is provided in the `gp` argument as a vector. The argument `emodel` denotes the method to be used for functional principal component decomposition. If `emodel = "classical"`, then, the classical functional principal component decomposition is performed. On the other hand, if `emodel = "robust"`, then, the RFPCA method of Bali *et al.* (2011) is used to obtain the functional principal components and the corresponding principal component scores. In Figure 3, the plot of five functional principal components computed from a simulated functional data using RFPCA and `nbasis = 20` B-spline basis expansion functions is presented. This Figure can be produced by the following code:

```
library(robflreg)
# Generate a dataset with five functional predictors and 200
# observations at 101 equally spaced point in the interval [0, 1]
# for each variable for the function-on-function regression model
set.seed(123)
sim.data <- generate.ff.data(n.pred = 5, n.curve = 200, n.gp = 101)
# Response variable
Y <- sim.data$Y
gpY <- seq(0, 1, length.out = 101) # grid points

# Perform robust functional principal component analysis on the response variable Y
rob.fpca <- getPCA(data = Y, nbasis = 20, ncomp = 4, gp = gpY, emodel = "robust")

# Principal components
PCs <- rob.fpca$PCAcoef

plot(PCs, xlab = "Grid point", ylab = "Values")


[1] "done"
```

Robust estimation of the SFLRM

In the robust estimation of the SFRM, we first consider the principal component decomposition of the functional predictors as follows:

$$\mathcal{X}_p(s) = \sum_{k=1}^{K_p} \xi_{pk}\psi_{pk}(s),$$

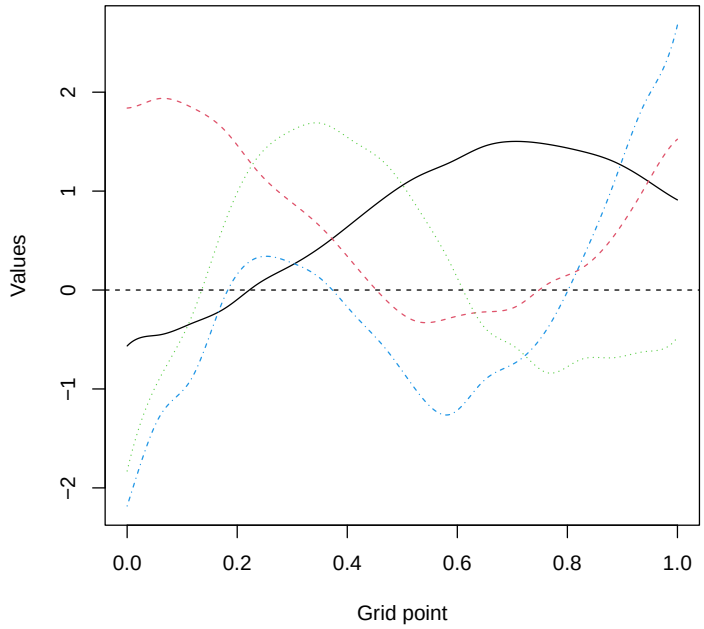

Figure 3: Plot of the first five functional principal components of a simulated functional data using RFPCA.

where K_p is the truncation constant for the p-th functional predictor $\mathcal{X}_p(s)$, $\psi_{pk}(s)$ is the k-th eigenfunction obtained by the RFPCA of Bali *et al.* (2011), and ξ_{pk} is the corresponding principal components score, given by:

$$\xi_{pk} = \int_0^1 \mathcal{X}_p(s)\psi_{pk}(s)ds.$$

In practice, the eigenfunctions are approximated via a basis expansion function such as B-spline. Let $\varphi_p(s)$ denote the B-spline basis expansion function and $A_p = (a_{pl})$ being an $n \times L$-dimensional matrix of basis expansion coefficients for the p-th functional predictor variable. In addition, let $\boldsymbol{\varphi} = \int_0^1 \varphi_p(s)\varphi_p^\top(s)ds$ and $\boldsymbol{\varphi}^{1/2}$ denote the $L \times L$ dimensional matrix of inner products between the basis expansion functions and its square root, respectively. Then, the infinite-dimensional RFPCA of $\mathcal{X}_p(s)$ is equivalent to the multivariate principal component analysis of $A_p\boldsymbol{\varphi}^{1/2}$ and the k-th eigenfunction is given by $\psi_{pk}(s) = \boldsymbol{\varphi}^{-1/2}v_{pk}$, where v_{pk} denotes the p-th eigenvector of the sample covariance matrix of $A_p\boldsymbol{\varphi}^{1/2}$ (see, e.g., Ocana, Aguilera, and Escabias 2007, for more information).

If we assume that the p-th regression coefficient function $\beta_p(s)$ admits the similar functional principal decomposition as the functional predictors as follows:

$$\beta_p(s) = \sum_{k=1}^{K_p} b_{pk}\psi_{pk}(s),$$

where $b_{pk} = \int_0^1 \beta_p(s)\psi_{pk}(s)ds$. Then, the infinite-dimensional SFRM in (1) is approximated by the finite-dimensional regression model of scalar response on all the functional principal components scores as follows:

$$Y = \sum_{p=1}^{P}\sum_{k=1}^{K_p} b_{pk}\xi_{pk}.$$

Main functions for the robust estimation of a SFRM and their arguments

The main function to robustly estimate a SFRM is called `rob.sf.reg()`:

```
rob.sf.reg(Y, X, emodel = c("classical", "robust"), fmodel = c("LTS", "MM", "S", "tau"),
nbasis = NULL, gp = NULL, ncomp = NULL)
```

In the `rob.sf.reg()` interface, the scalar response is provided in the `Y` argument as an $n \times 1$-dimensional column vector, where n is the sample size. The functional predictors, on the other hand, are provided in the `X` argument as a list object. Each element of `X` is an $n \times L_p$-dimensional matrix containing the observations of p-th functional predictor $\mathcal{X}_p(s)$, where L_p is the number of grid points for $\mathcal{X}_p(s)$. The method to be used for functional principal component decomposition is provided in the `emodel` argument. If `emodel = "classical"`, then the classical functional principal component decomposition is performed to obtain principal components and the corresponding principal components scores. The coefficient vector of the regression problem of scalar response on the principal components scores is estimated via the least-squares method. If `emodel = "robust"`, then, the RFPCA of Bali *et al.* (2011) is performed to obtain the principal components and the corresponding principal components scores. In this case, the method used to estimate the coefficient vector of the regression problem constructed by the scalar response and principal components scores is provided in the `fmodel` argument. Here, one of the methods among LTS, MM, S, and tau can be chosen to estimate the parameter vector. The number of B-spline basis expansion functions used to approximate the functional principal components are provided in the `nbasis` argument as a vector with length p. If `nbasis = NULL`, then 20 B-spline basis expansion functions are used to approximate the functional principal components of all functional predictors. The grid points for the functional predictors are provided in the `gp` argument as a list object. The p-th element of `gp` is a vector containing the grid points of the p-th functional predictor $\mathcal{X}_p(s)$. If `gp = NULL`, then L_p equally spaced time points in the interval $[0, 1]$ are used for the p-th functional predictor. The number of functional predictors to be computed for the functional predictors are provided in the `ncomp` argument as a vector with length P. If `ncomp = NULL`, then 4 principal components are used for each functional predictor.

The interface `get.sf.coeffs()` can be used to obtain the estimated regression coefficient functions from a fitted SFRM:

```
get.sf.coeffs(object)
```

In this interface, the argument `object` is the output object obtained using the interface `rob.sf.reg()`. The interface `get.sf.coeffs()` produces a list object whose p-th element is a vector with length L_p containing the p-th regression coefficient function $\beta_p(s)$.

The plots of the estimated regression coefficient functions can be obtained using the interface `plot_sf_coeffs()`:

```
plot_sf_coeffs(object, b)
```

In this interface, the argument `object` is the output object obtained by the interface `get.sf.coeffs()`. The argument `b`, on the other hand, is an integer value indicating which regression parameter function to be plotted. In Figure 4, the plots of the regression coefficient functions obtained from simulated data using `RFPCA` and tau estimator are presented. The following code can produce this Figure and the results:

```
library(robflreg)
# Generate a dataset with three functional predictors and 400
# observations at 101 equally spaced point in the interval [0, 1]
# for each variable for the scalar-on-function regression model
set.seed(123)
sim.data <- generate.sf.data(n = 400, n.pred = 3, n.gp = 101)
# Response variable
Y <- sim.data$Y
# Predictors
X <- <- sim.data$X

gp <- rep(list(seq(0, 1, length.out = 101)), 3) # grid points of Xs

# Fit a scalar-on-function regression model for the generated data
# using the classical functional principal component analysis method:
model.fit <- rob.sf.reg(Y, X, emodel = "classical", gp = gp)

# Estimated regression coefficient functions
coefs <- get.sf.coeffs(model.fit)
# Plot the first regression coefficient function
plot_sf_coeffs(object = coefs, b = 1)
```

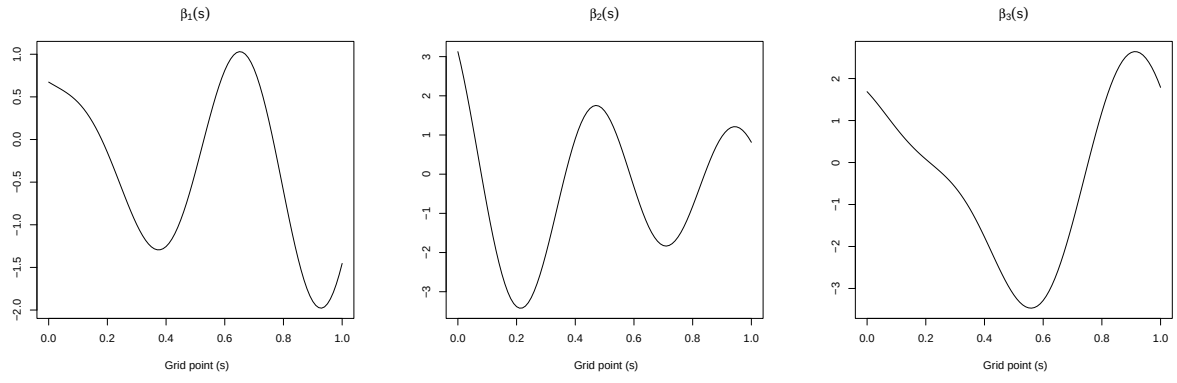

Figure 4: A plot of the estimated regression coefficient functions obtained from a simulated data using RFPCA and tau estimator.

Robust estimation of the FFLRM

Let us consider the functional principal decompositions of both the functional response and functional predictor variables as follows:

$$\mathcal{Y}(t) = \sum_{k=1}^{K} \zeta_k \phi_k(t), \quad , \mathcal{X}_p(s) = \sum_{j=1}^{K_p} \xi_{pj} \psi_{pj}(s),$$

where $\phi_k(t)$ and $\psi_{pj}(s)$ respectively are the k-th and j-th eigenfunctions of $\mathcal{Y}(t)$ and $\mathcal{X}_p(s)$ obtained by the RFPCA and ζ_k and ξ_{pj} are the corresponding principal components scores given by

$$\zeta_k = \int_0^1 \mathcal{Y}(t)\phi_k(t)dt, \quad \xi_{pj} = \int_0^1 \mathcal{X}_p(s)\psi_{pj}(s)ds.$$

If we assume that the p-th bivariate regression coefficient function $\beta_p(s,t)$ admits the principal component decomposition with the eigenfunctions $\phi_k(t)$ and $\psi_{pj}(s)$ as follows:

$$\beta_p(s,t) = \sum_{k=1}^{K} \sum_{j=1}^{K_p} b_{pkj} \phi_k(t) \psi_{pj}(s),$$

where $b_{pkj} = \int_0^1 \int_0^1 \beta_p(s,t)\phi_k(t)\psi_{pj}(s)dtds$. Then, the infinite-dimensional FFRM in (2) is approximated by the finite-dimensional regression model of principal component scores of the functional response on all the functional principal components scores as follows:

$$\zeta_k = \sum_{p=1}^{P} \sum_{j=1}^{K_p} b_{pkj} \xi_{pj}.$$

Finally, the following regression model is obtained for the functional response

$$\mathcal{Y}(t) = \sum_{k=1}^{K} \left(\sum_{p=1}^{P} \sum_{j=1}^{K_p} b_{pkj} \xi_{pj} \right) \phi_k(t).$$

Main functions for the robust estimation of a FFRM and their arguments

The main function to estimate the FFRM robustly is called `rob.ff.reg()`:

```
rob.ff.reg(Y, X, model = c("full", "selected"), emodel = c("classical", "robust"),
fmodel = c("MCD", "MLTS", "MM", "S", "tau"), nbasisY = NULL, nbasisX = NULL,
gpY = NULL, gpX = NULL, ncompY = NULL, ncompX = NULL)
```

In the `rob.ff.reg()` interface, the functional response is provided in the `Y` argument as a matrix. The functional predictors, on the other hand, are provided in the argument `X` as a list object. Each element of `X` is a matrix containing the observations of p-th functional predictor. The model type to be fitted can be chosen with `model` argument. If `model = "full"`, then, all the functional predictors are used in the model. On the other hand, if `model = "selected"`, then, only the significant

functional predictor variables determined by the forward variable selection procedure of Beyaztas and Shang (2021) are used in the model. The method to be used for functional principal component decomposition is provided in the `emodel` argument. If `emodel = "classical"`, then, the classical functional principal component decomposition is performed to obtain principal components and the corresponding principal components scores and the coefficient vector of the regression problem of principal components scores of the functional response on the principal components scores are estimated via the least-squares method. If `emodel = "robust"`, then, the RFPCA of Bali *et al.* (2011) is performed to obtain the principal components and the corresponding principal components scores. In this case, the method used to estimate the coefficient matrix of the regression problem constructed by the principal components scores is provided in the `fmodel` argument. Here, one of the method among MCD, MLTS, MM, S, and tau can be chosen to estimate the parameter matrix. The number of B-spline basis expansion functions used to approximate the functional principal components of response and predictor variables are provided in the `nbasisY` and `nbasisX` arguments, respectively. The argument `nbasisY` is a numeric value while the argument `nbasisX` is a vector with length P. If `nbasisY = NULL` and `nbasisX = NULL`, then, 20 B-spline basis expansion functions are used to approximate the functional principal components of functional response and all the functional predictors. The grid points for the functional response and functional predictors are provided in the `gpY` and `gpX` arguments, respectively. The argument `gpY` is a vector consisting of the grid points of the functional response $\mathcal{Y}(t)$. On the other hand, the argument `gpX` is a list object and its p-th element is a vector containing the grid points of the p-th functional predictor $\mathcal{X}_p(s)$. If `gpY = NULL` and If `gpX = NULL`, then, equally spaced time points in the interval $[0, 1]$ are used for all the functional variables. The number of functional predictors to be computed for the functional response and functional predictors are provided in the arguments `ncompY` and `ncompX`, respectively. The argument `ncompY` is a numeric value while the argument `ncompX` is a vector with length P. If `ncompY = NULL` and `ncompX = NULL`, then, 4 principal components are used for each functional variable.

The estimated bivariate regression coefficient functions from a fitted FFRM is obtained by the `get.ff.coeffs()` interface:

```
get.ff.coeffs(object)
```

In this interface, the argument `object` is the output object obtained using the interface `rob.ff.reg()`. The interface `get.ff.coeffs()` produces a list object whose p-th element is a matrix containing the p-th bivariate regression coefficient function $\beta_p(s, t)$.

The 3D plots of the estimated bivariate regression coefficient functions can be obtained using the interface `plot_ff_coeffs()`:

```
plot_ff_coeffs(object, b, phi, theta, cex.axis)
```

In this interface, the argument `object` is the output object obtained by the interface `get.ff.coeffs()`. The argument `b` is an integer value indicating which regression parameter function to be plotted. The arguments `phi` and `theta` are numeric values defining the viewing directions. The argument `cex.axis` is a numeric value defining the size of the tick label. In Figure 5, the 3D plots of the regression coefficient functions obtained from a simulated data using `RFPCA` and MM estimator are presented. This Figure and the results can be produced by the following code:

```
library(robflreg)
# Generate a dataset with three functional predictors and 200
# observations at 101 equally spaced point in the interval [0, 1]
# for each variable for the function-on-function regression model
set.seed(123)
sim.data <- generate.ff.data(n.pred = 3, n.curve = 200, n.gp = 101)
# Response variable
Y <- sim.data$Y
# Predictors
X <- <- sim.data$X

gpY = seq(0, 1, length.out = 101) # grid points of Y
gpX <- rep(list(seq(0, 1, length.out = 101)), 3) # grid points of Xs

# Fit a function-on-function regression model for the generated data
# using the RFPCA and MM estimator:
model.fit <- rob.ff.reg(Y, X, model = "full", emodel = "robust",
fmodel = "MM", gpY = gpY, gpX = gpX)

# Estimated bivariate regression coefficient functions
coefs <- get.ff.coeffs(model.fit)
# Plot the first bivariate regression coefficient function
plot_ff_coeffs(object = coefs, b = 1, phi = 5, theta = 40, cex.axis = 0.75)
```

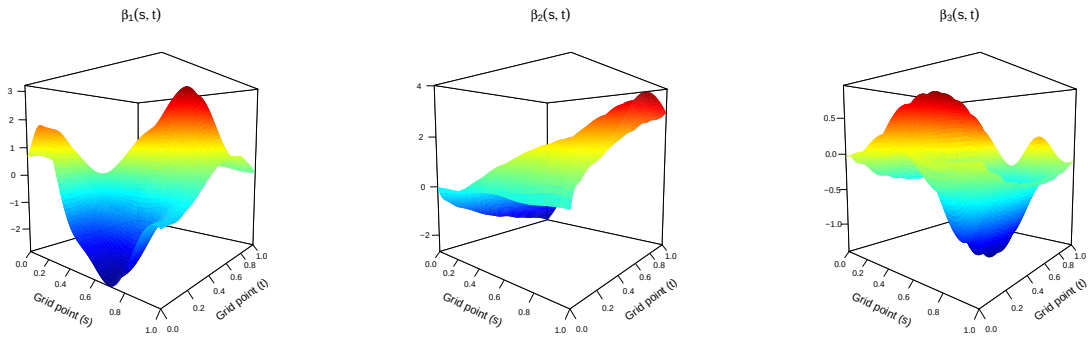

Figure 5: 3D plots of the estimated bivariate regression coefficient functions obtained from a simulated data using RFPCA and MM estimator.

Outlier detection in the functional response

Detection of outliers in functional data is an important problem (see, e.g., Sun and Genton 2011; Arribas-Gil and Romo 2014; Dai and Genton 2018). From a fitted FFRM, the **robflr** package with the interface `rob.out.detect()` allows to detect outliers in the functional response. While doing so, the functional depth-based outlier detection method of Febrero-Bande, Galeano, and Gonzalez-Mantelga (2008) together with the h-modal depth proposed by Cuaves, Febrero, and Fraiman (2007) is applied to the estimated residual functions obtained from `rob.ff.reg()` to determine the outliers in the

response variable. In the outlier detection algorithm, the threshold value used to identify outliers is determined by the smoothed bootstrap procedure proposed by Febrero-Bande *et al.* (2008). The `rob.out.detect()` is as follows:

```
rob.out.detect(object, alpha = 0.01, B = 200, fplot = FALSE)
```

Herein, the argument `object` is an output object obtained from `rob.ff.reg()`. `alpha`, whose default value is 0.01, denotes the percentile of the distribution of the functional depth. `B` denotes the number of bootstrap samples (the default value is `B = 200`). `fplot` is a logical argument, if `fplot = TRUE`, then, the outlying points flagged by the method is plotted along with the values of functional response $\mathcal{Y}(t)$.

To show how the interface `rob.ff.reg()` works, we simulate an outlier-contaminated dataset for the FFRM. Then, we apply the outlier detection algorithm with the classical FPCA - least squares estimator and the RFPCA - MM estimator. The plots of the functional response and detected outlying observations are presented in Figure 6. The results show that the classical method mistakenly flags two non-outlying curves as outliers, while the robust procedure flags all the outliers correctly. The following code can produce the results and Figure 6:

```
library(robflreg)
# Generate a dataset with five functional predictors and 200
# observations at 101 equally spaced point in the interval [0, 1]
# for each variable for the function-on-function regression model
set.seed(987)
sim.data <- generate.ff.data(n.pred = 5, n.curve = 200, n.gp = 101)
# Response variable
Y <- sim.data$Y
# Predictors
X <- <- sim.data$X

gpY = seq(0, 1, length.out = 101) # grid points of Y
gpX <- rep(list(seq(0, 1, length.out = 101)), 5) # grid points of Xs

# Generate another dataset for the function-on-function regression model
# to contaminate the original data
set.seed(123)
sim.data2 <- generate.ff.data(n.pred = 5, n.curve = 100, n.gp = 101)
# Contaminate 10\% of the functional response
out.index <- sample(1:100, 20)
Y[out.index,] <- sim.data2$Y[out.index,] + 5
# Contaminate 10\% of each functional predictor
for(i in 1:5)
  X[[i]][out.index,] <- sim.data2$X[[i]][out.index,]

# Perform classical function-on-function regression using least-squares
model.classical <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "classical",
                        gpY = gpY, gpX = gpX)

# Perform robust function-on-function regression using MM-estimator
model.MM <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "robust", fmodel = "MM",
```

```
                        gpY = gpY, gpX = gpX)

# Detect outliers using rob.out.detect function
rob.out.detect(object = model.classical, fplot = TRUE)
# outlying functions are: 6 7 13 28 35 38 39 45 50 51 55 56 57 62 65 66
# 72 85 88 91 99 111 113
# rob.out.detect(object = model.MM, fplot = TRUE)
outlying functions are: 6 7 13 28 35 38 39 45 50 51 55 56 57 62 65 66
# 72 85 91 99

# Compare with the original outliers
sort(out.index)
# [1]  6  7 13 28 35 38 39 45 50 51 55 56 57 62 65 66 72 85 91 99

outlying functions are: 6 7 13 28 35 38 39 45 50 51 55 56 57 62 65 66 72 85 88 91 99 111 113

outlying functions are: 6 7 13 28 35 38 39 45 50 51 55 56 57 62 65 66 72 85 91 99
```

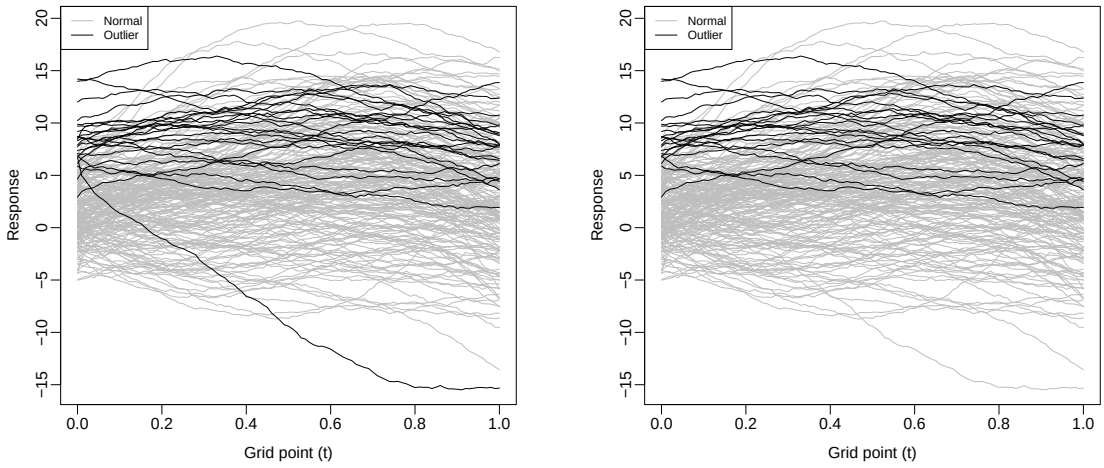

Figure 6: Plots of the functional response and detected outlier. Classical method (left panel) vs. Robust method (right panel).

Prediction

We review the prediction problem for a new set of functional predictors based on a fitted SFLRM and FFLRM.

Prediction for the SFRM

When robustly predicting the unknown values of the scalar response variable for a given new set of

functional predictors ($\mathcal{X}^*(s)$), the principal component scores of the new set of functional predictors (ξ^*) are obtained as follows:

$$\xi^*_{pk} = \int_0^1 \mathcal{X}^*_{pk}(s)\widehat{\psi}_{pk}(s)ds,$$

where $\widehat{\psi}_{pk}(s)$ is the k-th eigenfunction of the p-the functional predictor obtained by the RFPCA. Then, the predictions corresponding to the new set of functional predictors are obtained as follows:

$$\widehat{Y}^* = \sum_{p=1}^{P}\sum_{k=1}^{K_p}\widehat{b}_{pk}\xi^*_{pk},$$

where $\widehat{b}_{pk}$ is the estimated parameter vector obtained from the fitted model `rob.sf.reg()`.

Main function for the robust prediction of a SFRM and its arguments

The main function for the robust prediction of a SFRM is called `predict_sf_regression()`:

```
predict_sf_regression(object, Xnew)
```

In the interface `predict_sf_regression()`, the argument `object` is an output object obtained from `rob.sf.reg`. The new set of functional predictors is provided in the `Xnew` argument as a list object whose p-th element is a matrix denoting the new observations of $\mathcal{X}_p(s)$. `Xnew` must have the same length and the same structure as the input `X` of `rob.sf.reg`.

To evaluate the prediction performance of classical and robust methods, we simulate a dataset with size $n = 400$ for the SFRM. Then, the simulated dataset is divided into a training sample with a size of 200 and a test sample with a size of 200. Random outliers contaminate the training sample, and both the classical and robust methods with tau estimator are applied to the training sample to predict the values of the response variable in the test sample. To compare both methods, we compute the mean squared prediction error (MSPE):

$$\text{MSPE} = \frac{1}{200}\sum_{i=1}^{200}(Y^*_i - \widehat{Y}^*_i)^2,$$

where Y^*_i and $\widehat{Y}^*_i$ denote the observed and predicted values of the scalar response in the test sample. Our results indicate that the robust method considerably outperforms the classical method. The MSPE computed from the classical method is 4.7584, while the MSPE obtained from the robust method is 1.9434. The reproducible code to obtain those results is as follows:

```
library(robflreg)
# Generate a dataset with five functional predictors and 400
# observations at 101 equally spaced point in the interval [0, 1]
# for each variable for the scalar-on-function regression model
set.seed(987)
sim.data <- generate.sf.data(n = 400, n.pred = 5, n.gp = 101)
# Response variable
Y <- sim.data$Y
# Predictors
```

```
X <- <- sim.data$X

# Split the data into training and test samples.
Y.train <- Y[1:200,]
Y.test <- Y[201:400,]
X.train <- X.test <- list()
for(i in 1:5){
  X.train[[i]] <- X[[i]][1:200,]
  X.test[[i]] <- X[[i]][201:400,]
}
gp <- rep(list(seq(0, 1, length.out = 101)), 5) # grid points of Xs
# Generate another dataset for the scalar-on-function regression model
# to contaminate the original data
set.seed(123)
sim.data2 <- generate.sf.data(n = 400, n.pred = 5, n.gp = 101)
# Contaminate 10\% of the scalar response
out.index <- sample(1:200, 20)
Y.train[out.index] <- sim.data2$Y[out.index] + 10
# Contaminate 10\% of each functional predictor
for(i in 1:5)
  X.train[[i]][out.index,] <- sim.data2$X[[i]][out.index,]
# Perform classical scalar-on-function
# regression model using training samples
model.classical <- rob.sf.reg(Y.train, X.train, emodel = "classical", gp = gp)
# Perform robust scalar-on-function
# regression using training samples and tau-estimator
model.tau <- rob.sf.reg(Y.train, X.train, emodel = "robust", fmodel = "tau", gp = gp)
# Predict the observations in Y.test using model.classical
pred.classical <- predict_sf_regression(object = model.classical, Xnew = X.test)
# Predict the observations in Y.test using model.tau
pred.tau <- predict_sf_regression(object = model.tau, Xnew = X.test)
# Compute mean squared errors for the test sample
round(mean((Y.test - pred.classical)^2), 4) # 4.7584 (classical method)
round(mean((Y.test - pred.tau)^2), 4) # 1.9434 (tau method)
```

Prediction for the FFRM

In the robust prediction of the FFRM for a given new set of functional predictors, as in the scalar-on-function regression case, the principal component scores of the new set of functional predictors are first obtained:

$$\xi_{pk}^{*} = \int_0^1 \mathcal{X}_{pk}^{*}(s)\widehat{\psi}_{pk}(s)ds,$$

where $\widehat{\psi}_{pk}(s)$ is the k-th eigenfunction of the p-the functional predictor obtained by the RFPCA. Then, the predictions of functional response ($\widehat{\mathcal{Y}}(t)$) corresponding to the new set of functional predictors are obtained as follows:

$$\widehat{\mathcal{Y}}^{*}(t) = \sum_{k=1}^{K}\left(\sum_{p=1}^{P}\sum_{j=1}^{K_p}\widehat{b}_{pkj}\xi_{pj}^{*}\right)\widehat{\phi}_k(t),$$

where $\widehat{\phi}_k(t)$ is the k-th eigenfunction of the functional response obtained by RFPCA and $\widehat{b}_{pkj}$ is the estimated parameter matrix obtained from the fitted model `rob.ff.reg()`

Main function for the robust prediction of a FFRM and its arguments

The main function for the robust prediction of a FFRM is called `predict_ff_regression()`:

```
predict_ff_regression(object, Xnew)
```

Here, the argument `object` is an output object obtained from `rob.ff.reg`. The new set of functional predictors is provided in the `Xnew` argument as a list object whose p-th element is a matrix denoting the new observations of $\mathcal{X}_p(s)$. `Xnew` must have the same length and the same structure as the input `X` of `rob.ff.reg`.

We simulate a dataset with size $n = 200$ for the FFRM to investigate and compare the prediction performance of the classical and robust methods. The simulated dataset is divided into a training sample with a size of 100 and a test sample with a size of 100. Random outliers contaminate the training sample, and both the classical and robust methods with MM estimator are applied to the training sample to predict the values of the response variable in the test sample. To compare both methods, we compute the following MSPE:

$$\text{MSPE} = \frac{1}{100}\sum_{i=1}^{200} \|\mathcal{Y}_i^*(t) - \widehat{\mathcal{Y}}_i^*(t))\|_{\mathcal{L}_2}^2,$$

where $\mathcal{Y}_i^*(t)$ and $\widehat{\mathcal{Y}}_i^*(t))$ denote the observed and predicted values of the functional response in the test sample. Our results show that the robust method produces a significantly smaller MSPE value than the classical method. The MSPE computed from the classical method is 2.3456, while the MSPE obtained from the robust method is 0.9318. The reproducible code to obtain those results is as follows:

```
library(robflreg)
# Generate a dataset with five functional predictors and 200
# observations at 101 equally spaced point in the interval [0, 1]
# for each variable for the function-on-function regression model
set.seed(987)
sim.data <- generate.ff.data(n.pred = 5, n.curve = 200, n.gp = 101)
# Response variable
Y <- sim.data$Y
# Predictor variables
X <- sim.data$X
# Split the data into training and test samples.
Y.train <- Y[1:100,]

Y.test <- Y[101:200,]
X.train <- X.test <- list()
for(i in 1:5){
  X.train[[i]] <- X[[i]][1:100,]
  X.test[[i]] <- X[[i]][101:200,]
}
```

```
gpY = seq(0, 1, length.out = 101) # grid points of Y
gpX <- rep(list(seq(0, 1, length.out = 101)), 5) # grid points of Xs

# Generate another dataset for the function-on-function regression model
# to contaminate the original data
set.seed(123)
sim.data2 <- generate.ff.data(n.pred = 5, n.curve = 100, n.gp = 101)
# Contaminate 10\% of the functional response
out.index <- sample(1:100, 10)
Y.train[out.index,] <- sim.data2$Y[out.index,]
# Contaminate 10\% of each functional predictor
for(i in 1:5)
  X.train[[i]][out.index,] <- sim.data2$X[[i]][out.index,]
# Perform classical function-on-function
# regression model using training samples
model.classical <- rob.ff.reg(Y = Y.train, X = X.train, model = "full",
                              emodel = "classical", gpY = gpY, gpX = gpX)
# Perform robust function-on-function
# regression using training samples and MM-estimator
model.MM <- rob.ff.reg(Y = Y.train, X = X.train, model = "full", emodel = "robust",
                       fmodel = "MM", gpY = gpY, gpX = gpX)
# Predict the functions in Y.test using model.classical
pred.classical <- predict_ff_regression(object = model.classical, Xnew = X.test)
# Predict the functions in Y.test using model.MM
pred.MM <- predict_ff_regression(object = model.MM, Xnew = X.test)
# Compute mean squared errors for the test sample
round(mean((Y.test - pred.classical)^2), 4) # 2.3456 (classical method)
round(mean((Y.test - pred.MM)^2), 4) # 0.9318 (MM method)
```

Conclusion

The R package **robflreg** provides an implementation of several robust procedures to fit and predict SFLRM and FFLRM. These methods are centered on the RFPCA of Bali *et al.* (2011), which is a popular robust dimension reduction technique in functional data and several robust regression parameter estimators. In addition, the package **robflreg** allows to fit and predict SFLRM and FFLRM via the classical FPCA and least-squares estimator. Several simulation examples show that the robust procedures provide better inference for the functional linear regression models when outliers are presented in the response and predictor variables.

References

Arribas-Gil A, Romo J (2014). "Shape outlier detection and visualization for functional data: the outliergram." *Biostatistics*, **15**(4), 603–619.

Bali JL, Boente G, Tyler DE, Wang JL (2008). "The multivariate least-trimmed squares estimator." *Journal of Multivariate Analysis*, **99**(3), 311–338.

Bali JL, Boente G, Tyler DE, Wang JL (2011). "Robust functional principal components: A projection-pursuit approach." *The Annals of Statistics*, **39**(6), 2852–2882.

Ben MG, Martinez E, Yohai VJ (2006). "Robust estimation for the multivariate linear model based on a τ scale." *Journal of Multivariate Analysis*, **97**(7), 1600–1622.

Beyaztas U, Shang HL (2020). "On function-on-function regression: Partial least squares approach." *Environmental and Ecological Statistics*, **27**(1), 95–114.

Beyaztas U, Shang HL (2021). "A partial least squares approach for function-on-function interaction regression." *Computational Statistics*, **36**(2), 911–939.

Beyaztas U, Shang HL (2022). "A robust functional partial least squares for scalar-on-multiple-function regression." *Journal of Chemometrics*, **36**(4), e3394.

Bilodeau M, Duchesne P (2000). "Robust estimation of the SUR model." *The Canadian Journal of Statistics*, **28**(2), 277–288.

Boente G, Salibian-Barrera M, Vena P (2020). "Robust estimation for semi-functional linear regression models." *Computational Statistics & Data Analysis*, **152**, 107041.

Cardot H, Ferraty F, Sarda P (1991). "Functional linear model." *Statistics and Probability Letters*, **45**, 11–22.

Cardot H, Ferraty F, Sarda P (2003). "Spline estimators for the functional linearmodel." *Statistica Sinica*, **13**(3), 571–591.

Chen D, Hall P, Müller HG (2011). "Single and multiple index functional regression models with nonparametric link." *The Annals of Statistics*, **39**(3), 1720–1747.

Chiou JM, Yang YF, Chen YT (2016). "Multivariate functional linear regression and prediction." *Journal of Multivariate Analysis*, **146**, 301–312.

Croux C, Ruiz-Gazen A (1996). "High breakdown estimators for principal components: the projection-pursuit approach revisited." *Journal of Multivariate Analysis*, **95**(1), 206–226.

Şentürk D, Müller HG (2008). "Generalized varying coefficient models for longitudinal data." *Biometrika*, **95**(3), 653–666.

Cuaves A, Febrero M, Fraiman R (2007). "Robust estimation and classification for functional data via projection-based depth notions." *Computational Statistics*, **22**(3), 481–496.

Cuevas A (2014). "A partial overview of the theory of statistics with functional data." *Journal of Statistical Planning and Inference*, **147**, 1–23.

Dai W, Genton MG (2018). "Multivariate functional data visualization and outlier detection." *Journal of Computational and Graphical Statistics*, **27**(4), 923–934.

Dou WW, Pollard D, Zhou HH (2012). "Estimation in functional regression for general exponential families." *The Annals of Statistics*, **40**(5), 2421–2451.

Febrero-Bande M, Galeano P, Gonzalez-Mantelga 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, New York.

Harezlak J, Coull BA, Laird NM, Magari SR, Christiani DC (2007). "Penalized solutions to functional regression problems." *Computational Statistics and Data Analysiss*, **51**(10), 4911–4925.

Horváth L, Kokoszka P (2012). *Inference for Functional Data with Applications*. Springer, New York.

Hsing T, Eubank R (2015). *Theoretical Foundations of Functional Data Analysis, with an Introduction to Linear Operators*. John Wiley & Sons, Chennai, India.

Hullait H, Leslie DS, Pavlidis NG, King S (2021). "Robust function-on-function regression." *Technometrics*, **63**(3), 396–409.

Ivanescu AE, Staicu AM, Scheipl F, Greven S (2015). "Penalized function-on-function regression." *Computational Statistics*, **30**(2), 539–568.

James GM (2002). "Generalized linear models with functional predictors." *Journal of Royal Statistical Society, Series B*, **64**(3), 411–432.

Jiang C, Wang JL (2011). "Functional single index models for longitudinal data." *The Annals of Statistics*, **39**(1), 362–388.

Kalogridis I, Aelst SV (2019). "Robust functional regression based on principal components." *Journal of Multivariate Analysis*, **173**, 393–415.

Kokoszka P, Reimherr M (2017). *Introduction to Functional Data Analysis*. CRC Press, Boca Raton.

Koller M, Stahel WA (2011). "Sharpening Wald-type inference in robust regression for small samples." *Computational Statistics & Data Analysis*, **55**(8), 2504–2515.

Kudraszow NL, Moronna RA (2011). "Estimates of MM type for the multivariate linear model." *Journal of Multivariate Analysis*, **102**(9), 1280–1292.

Maronna RA, Yohai VJ (2013). "Robust functional linear regression based on splines." *Computational Statistics and Data Analysis*, **65**, 46–55.

Matsui H, Kawano S, Konishi S (2009). "Regularized functional regression modeling for functional response and predictors." *Journal of Math-for-Industry*, **1**(A3), 17–25.

Ocana FA, Aguilera AM, Escabias M (2007). "Computational considerations in functional principal component analysis." *Computational Statistics*, **22**(3), 449–465.

Raña P, Aneiros G, Vilar JM (2015). "Detection of outliers in functional time series." *Environmetrics*, **26**(3), 178–191.

Ramsay JO, Dalzell CJ (1991). "Some tools for functional data analysis." *Journal of the Royal Statistical Society, Series B*, **53**(3), 539–572.

Ramsay JO, Silverman BW (2002). *Applied Functional Data Analysis*. Springer, New York.

Ramsay JO, Silverman BW (2006). *Functional Data Analysis*. Springer, New York.

Reiss PT, Ogden TR (2007). "Functional principal component regression and functional partial least squares." *Journal of the American Statistical Association: Theory and Methods*, **102**(479), 984–996.

Rousseeuw PJ (1984). "Least median of squares regression." *Journal of the American Statistical Association: Theory and Methods*, **79**(388), 871–881.

Rousseeuw PJ, Driessen KV, Aelst SV, Agullo J (1984). "Robust multivariate regression." *Technometrics*, **46**(3), 293–305.

Salibian-Barrera M, Willems G, Zamar R (2008). "The fast-tau estimator for regression." *Journal of Computational and Graphical Statistics*, **17**(3), 659–682.

Shin H, Lee S (2016). "An RKHS approach to robust functional linear regression." *Statistica Sinica*, **26**, 255–272.

Sun Y, Genton MG (2011). "Functional boxplots." *Journal of Computational and Graphical Statistics*, **20**(2), 316–334.

Yao F, Müller HG, Wang JL (2005). "Functional linear regression analysis for longitudinal data." *The Annals of Statistics*, **33**(6), 2873–2903.

Yohai VJ (1987). "High breakdown-point and high efficiency estimates for regression." *The Annals of Statistics*, **15**(2), 642–665.

Zhu H, Brown PJ, Morris JS (2011). "Robust, adaptive functional regression in functional mixed model framework." *Journal of the American Statistical Association*, **106**(1), 1167–1179.

Affiliation:

Ufuk Beyaztas
Marmara University
Department of Statistics
Goztepe Campus, 34722, Kadikoy, Istanbul, Turkey
E-mail: ufuk.beyaztas@marmara.edu.tr

Han Lin Shang
Macquarie University
Department of Actuarial Studies and Business Analytics
Level 7, 4 Eastern Road
Sydney, NSW 2109, Australia
E-mail: hanlin.shang@mq.edu.au