Robust functional linear regression models

by Ufuk Beyaztas and Han Lin Shang

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. 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, respectively. However, most existing estimation strategies are based on 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 and empirical datasets.

Introduction

In this paper, we aim 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.

The interest and need for developing statistical methods to analyze functional data have increased in the last few decades. Consult Ramsay and Dalzell (1991), Ramsay and Silverman (2002, 2006), Ferraty and Vieu (2006), Horváth and Kokoszka (2012), Cuevas (2014), Hsing and Eubank (2015), Marron et al. (2015), Srivastava and Klassen (2016), Dryden and Mardia (2016), 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 functional response-functional predictors, respectively. Consult Cardot et al. (1991, 2003), James (2002), Reiss and Ogden (2007), Chen et al. (2011), Jiang and Wang (2011), Goldsmith et al. (2011), Dou et al. (2012), Tucker et al. (2019), Ahn et al. (2020), and Beyaztas and Shang (2022) for the SFLRM and Yao et al. (2005), Harezlak et al. (2007), Şentürk and Müller (2008), Matsui et al. (2009), Ivanescu et al. (2015), Scheipl et al. (2015) Chiou et al. (2016), and Beyaztas and Shang (2020) for the FFLRM. In addition, please see the available R packages **fda** (Ramsay et al., 2022) and **refund** (Goldsmith et al., 2022) for the implementation of many functional data analysis methods, including SFLRM and 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 et al., 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 et al., 2011; Maronna and Yohai, 2013; Shin and Lee, 2016; Kalogridis and Aelst, 2019; Boente et al., 2020; Hullait et al., 2021; Beyaztas and Shang, 2022). The non-robustness of these methods may also be because the methods use cross-sectional analysis, and the mean used in the regression model needs to be representative of the underlying data generating mechanism. In this paper, we provide a hands-on tutorial for the implementation of functional principal component regression based on 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 et al. (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 et al. (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 et al. (2004), multivariate least trimmed squares estimator (MLTS) of Bali et al. (2008), MM estimator of Kudraszow and Moronna (2011), S estimator of Bilodeau and Duchesne (2000), and the tau estimator of Ben et al. (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 principal component regression model 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 implemented using the **robflreg** package. Conclusions are given at the end.

Functional linear regression models

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, out.p = 0)
```

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, `n.gp` denotes the number of grid points, i.e., a fine grid on the interval $[0, 1]$, and `out.p` is an integer between 0 and 1, denoting the outlier percentage in the generated data. 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). Suppose outliers are allowed in the generated data, i.e., $out.p > 0$, then, the randomly selected $n \times out.p$ of the data are generated differently from the process mentioned above. In more detail, if $out.p > 0$, the regression coefficient functions (possibly different from the previously generated coefficient functions) generated from the coefficient space with b^* (instead of b),

where b^* is generated from a uniform distribution between 3 and 5, are used to generate the outlying observations. In addition, in this case, the following process is used to generate functional predictors:

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

where κ_j^* is a vector generated from a Normal distribution with mean one and variance $\sqrt{a}j^{-1/2}$ and

$$\nu_j^*(s) = 2\sin(j\pi s) - \cos(j\pi s).$$

Moreover, the error process is generated from a normal distribution with mean one and variance one. A graphical display of the generated dataset with five functional predictors and `n = 400` observations at 101 equally spaced points in the interval $[0, 1]$ obtained by `generate.sf.data()` is presented in Figure 1.

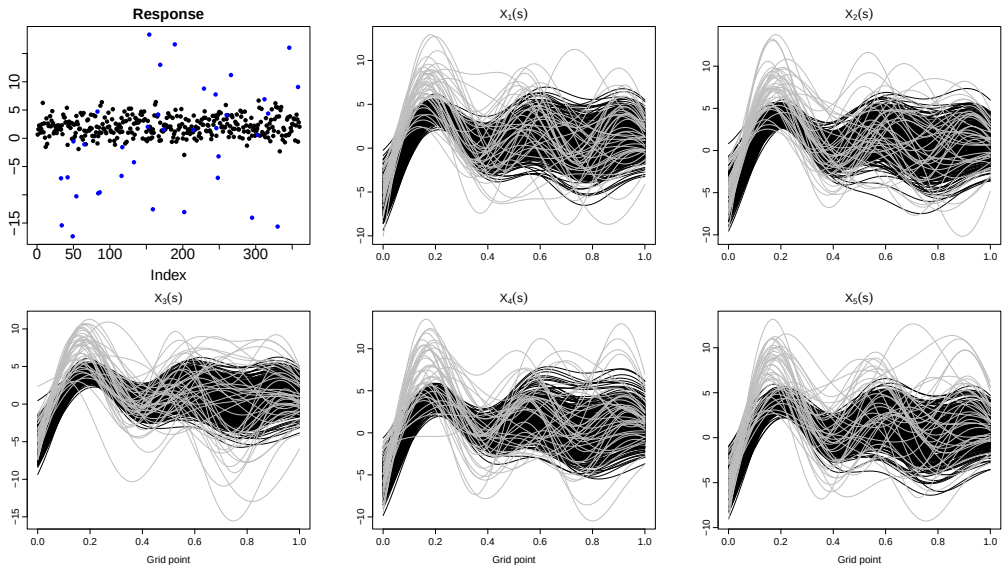

Figure 1: Plots of the simulated scalar response and functional predictor variables: In the plots, the black points and black curves denote the observations of scalar response and functional predictors generated under the smooth data generation process, respectively. The blue points and grey curves denote the outlying points in the scalar response and outlying curves in the functional predictors, respectively.

Figure 1 can be produced by the following code (refer to the supplement code file for all the reproducible code):

```
library(robflreg)
library(fda.usc)
set.seed(202)

# 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, out.p = 0.1)

# Response variable
Y <- sim.data$Y
# Predictors
X <- sim.data$X
# Regression coefficient functions
coeffs <- sim.data$f.coef
# Plot the scalar response
out.indx <- sim.data$out.indx
plot(Y[-out.indx,], type = "p", pch = 16, xlab = "Index", ylab = "",
main = "Response", ylim = range(Y))
```

```
points(out.indx, Y[out.indx,], type = "p", pch = 16, col = "blue")
# Plot the first functional predictor
fX1 <- fdata(X[[1]], argvals = seq(0, 1, length.out = 101))
plot(fX1[-out.indx,], lty = 1, ylab = "", xlab = "", col = "black",
     main = expression(X[1](s)), mgp = c(2, 0.5, 0), ylim = range(fX1))
lines(fX1[out.indx,], lty = 1, col = "grey")
```

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 the simulation of a dataset for the FFLRM as follows:

```
generate.ff.data(n.pred, n.curve, n.gp, out.p = 0)
```

In this interface, `n.pred` denotes the number of functional predictors to be generated, `n.curve` denotes the number of observations for each functional variable to be generated, `n.gp` denotes the number of grid points, i.e., a fine grid on the interval $[0,1]$, and `out.p` is an integer between 0 and 1, denoting the outlier percentage in the generated data. 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. If outliers are allowed in the generated data, i.e., $out.p > 0$, then, the randomly selected $n \times out.p$ of the data are generated differently from the process mentioned above. In more detail, if $out.p > 0$, the regression coefficient functions (possibly different from the previously generated coefficient functions) generated from the coefficient space with b^* (instead of b), where b^* is generated from a uniform distribution between 1 and 2, are used to generate the outlying observations. In addition, in this case, the following process is used to generate functional predictors:

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

where κ_j^* is a vector generated from a Normal distribution with mean one and variance $\sqrt{a}j^{-3/2}$ and

$$\nu_j^*(s) = 2\sin(j\pi s) - \cos(j\pi s).$$

A graphical display of the generated dataset with five functional predictors and `n = 200` observations at 101 equally spaced points in the interval $[0,1]$ obtained by `generate.ff.data()` is presented in Figure 2.

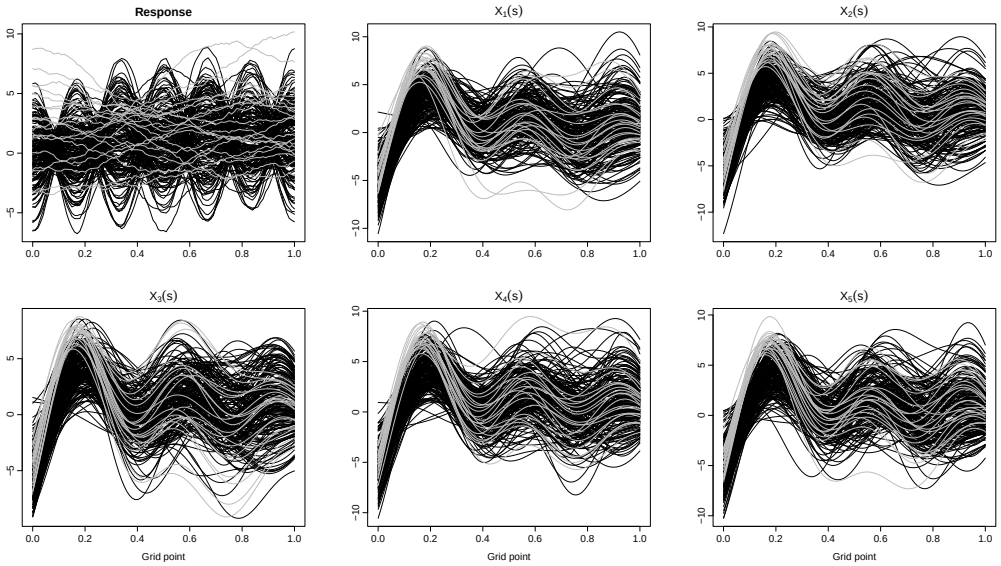

Figure 2: Plots of the simulated functional response and functional predictor variables: In the plots, the black curves denote the observations of the response and functional predictors generated under the smooth data generation process. The grey curves denote the outlying observations in the functional variables.

Figure 2 can be produced by the following code (refer to the supplement code file for all the reproducible code):

```
library(robflreg)
library(fda.usc)
set.seed(202)

# 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
sim.data <- generate.ff.data(n.pred = 5, n.curve = 200, n.gp = 101, out.p = 0.1)

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

Estimation

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

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. 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. 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 set is provided in the `data` argument as a matrix. The grid points of the functional data are provided in the `gp` argument as a vector. `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 argument `emodel` denotes the method 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. Figure 3 presents the plot of

five functional principal components computed from simulated functional data using RFPCA and `nbasis = 20` B-spline basis expansion functions.

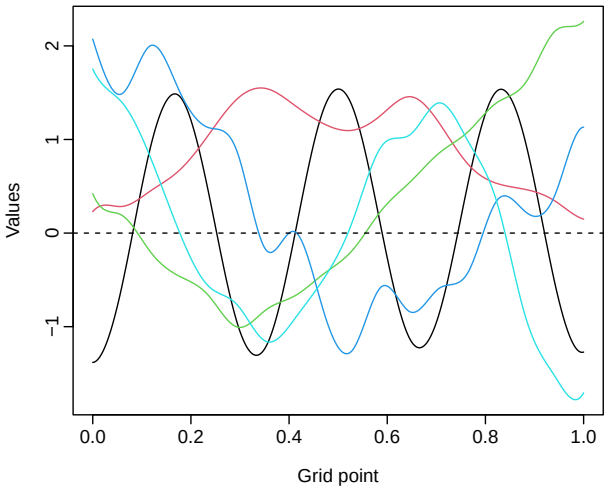

Figure 3: A plot of the first five functional principal components of the simulated functional data: The principal components were obtained using the RFPCA and different colors correspond to different principal components.

Figure 3 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(202)
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 = 5, gp = gpY, emodel = "robust")

# Principal components
PCs <- rob.fpca$PCAcoef

plot(PCs, xlab = "Grid point", ylab = "Values", lty = 1)
```

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:

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

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 $A_p = (a_{pl})$ be 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 et al., 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, X.scl = NULL, 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 as a list object in the `X` argument. 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 `rob.sf.reg()` interface also allows for scalar predictors, which can be provided in the `X.scl` as an $n \times R$-dimensional matrix, where R denotes the number of scalar predictors. In this case, the following SFRM is considered:

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

where $\boldsymbol{\gamma}$ denotes the vector of coefficients for the scalar predictors' matrix. The functional principal component decomposition method 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. 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. Suppose `nbasis = NULL`, then $\min(20, L_p/4)$ number of B-spline basis expansion functions are used for each functional predictor. 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 principal components to be computed for the functional predictors are provided in the `ncomp` argument as a vector with length P. If `ncomp = NULL`, then, for each functional predictor, the number whose usage results in at least 95% explained variation is used as the number of principal components.

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

```
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 Figure 4, the plots of the regression coefficient functions obtained from simulated data (outlier-contaminated) using `RFPCA` and MM estimator, as well as the classical functional principal component regression, are presented. In this interface, the argument `object` is the output object obtained by the interface `get.sf.coeffs()`. On the other hand, the argument `b` is an integer value indicating which regression parameter function to plot. It is clear from this figure that, compared with the classical functional principal component regression, the robust approach produces estimated parameter functions more similar to the true parameter functions.

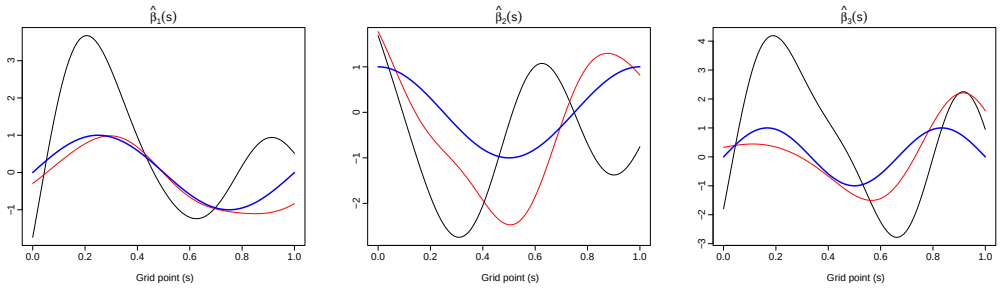

Figure 4: A plot of the estimated regression coefficient functions obtained from simulated data (outlier-contaminated) using RFPCA and MM estimator and the classical functional principal component regression. Blue lines denote the true parameter functions. On the other hand, black and red lines denote the estimated parameter functions by the classical functional principal component regression and robust MM-based functional principal component regression, respectively.

The following code can produce this Figure and the results (refer to the supplement code file for all the reproducible code):

```
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(202)
sim.data <- generate.sf.data(n = 400, n.pred = 3, n.gp = 101, out.p = 0.1)

# True parameter functions
true.b1 <- sim.data$f.coef[[1]]
true.b2 <- sim.data$f.coef[[2]]
true.b3 <- sim.data$f.coef[[3]]

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

# Fit a functional principal component regression model for the generated data
# using the robust functional principal component analysis method and tau estimator:
robust.fit <- rob.sf.reg(Y, X, emodel = "robust", fmodel = "MM", gp = gp)

# Estimated regression coefficient functions
classical.coefs <- get.sf.coeffs(classical.fit)
robust.coefs <- get.sf.coeffs(robust.fit)

# The first estimated regression coefficient function
plot_sf_coeffs(object = classical.coefs, b = 1)
lines(gp[[1]], robust.coefs$coefficients[[1]], col = "red")
```

```
lines(gp[[1]], true.b1, col = "blue", lwd = 2)
```

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. On the other hand, the functional predictors 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 functional principal component decomposition method 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 methods, among MCD, MLTS, MM, S, and tau estimators, 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. Suppose `nbasisY = NULL`

and `nbasisX = NULL`, then $\min(20, L_y)$ and $\min(20, L_p)$ B-spline basis expansion functions are used to approximate the functional principal components of functional response and p-th the functional predictor, where L_y and L_p respectively denote the number of grid points for $\mathcal{Y}(t)$ and $\mathcal{X}_p(s)$. 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 the number whose usage results in at least 95% explained variation is used as the number of principal components for each functional variable.

The estimated bivariate regression coefficient functions from a fitted functional principal component regression model are 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 image plots of the estimated bivariate regression coefficient functions can be obtained using the interface `plot_ff_coeffs()`:

```
plot_ff_coeffs(object, b)
```

In Figure 5, the image plots of the regression coefficient functions obtained from simulated data using `RFPCA` and MM estimator are presented. 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 is to be plotted.

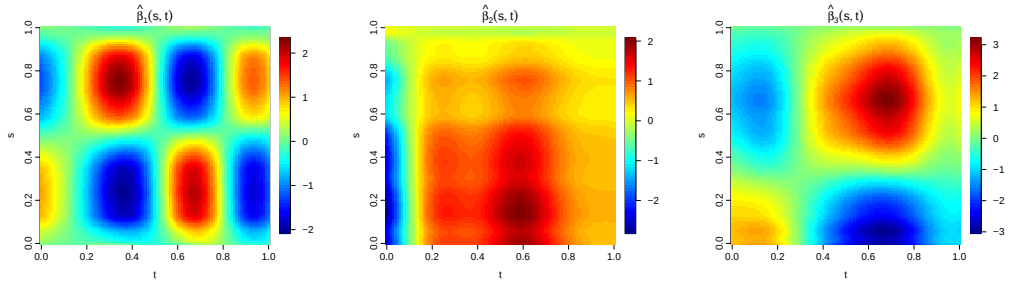

Figure 5: Image plots of the estimated bivariate regression coefficient functions obtained from simulated data using RFPCA and MM estimator.

This Figure and the results can be produced by the following code (refer to the supplement code file for all the reproducible 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(202)
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 functional principal component 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)
```

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 functional principal component regression for scalar response and scalar predictors, the **robflr** package with the interface `rob.out.detect()` allows to detection of outliers in the functional response. While doing so, the functional depth-based outlier detection method of Febrero-Bande et al. (2008) together with the h-modal depth proposed by Cuaves et al. (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 are 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 fails to flag 13 outlying curves, while the robust procedure fails to flag only two outlying curves.

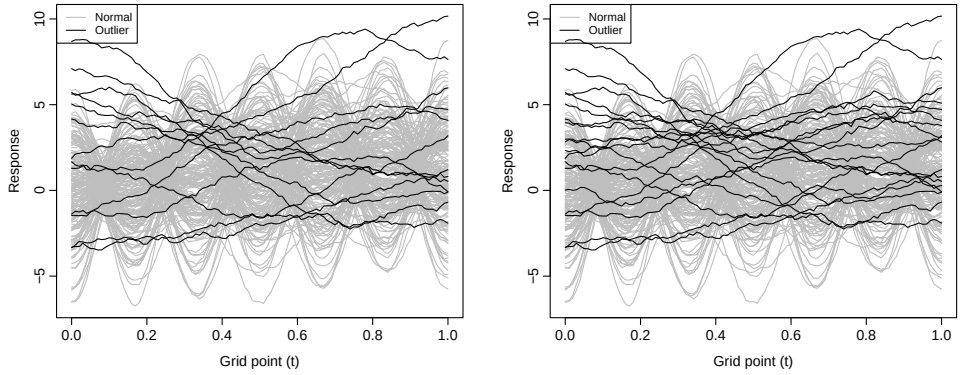

Figure 6: Plots of the functional response and detected outlier: Classical method (left panel) vs. Robust method (right panel). In the plots, the detected outlying curves are denoted by black, while the non-outlying curves are denoted by grey.

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(202)
sim.data <- generate.ff.data(n.pred = 5, n.curve = 200, n.gp = 101, out.p = 0.1)
out.indx <- sim.data$out.indx
```

```
# 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

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

# Perform robust functional principal component 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: 16 56 69 70 71 80 92 96 117 138 140 173 188
rob.out.detect(object = model.MM, fplot = TRUE)
# outlying functions are: 2 16 56 69 70 71 80 82 92 96 117 134 138 140
# 173 188 197 199

# Compare with the original outliers
sort(out.indx)
# [1] 2 16 47 56 69 70 71 80 82 92 96 117 134 138 140 162 173 188 197 199
```

Prediction

We review the prediction problem for a new set of functional predictors based on a fitted functional principal component regression model.

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, Xnew.scl = NULL)
```

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`. If scalar predictors are used in the SFRM, then, in the prediction process, the new set of scalar predictors is provided as a matrix in the `Xnew.scl` argument. The argument `Xnew.scl` must have the same length and the same structure as the input `X.scl` 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 data are divided into a training sample with a size of 280 and a test sample with a size of 120. Random outliers contaminate the training sample, and both the classical and robust methods with the 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 20.9388, while the MSPE obtained from the robust method is 1.868. 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(202)
sim.data <- generate.sf.data(n = 400, n.pred = 5, n.gp = 101, out.p = 0.1)
out.indx <- sim.data$out.indx
# Response variable
Y <- sim.data$Y
# Predictors
X <- sim.data$X

# Split the data into training and test samples.
indx.test <- sample(c(1:400)[-out.indx], 120)
indx.train <- c(1:400)[-indx.test]

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

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

# Perform classical functional principal component regression model using training samples
model.classical <- rob.sf.reg(Y.train, X.train, emodel = "classical", gp = gp)
# Perform robust functional principal component regression model
# 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) # 2.49 (classical method)
round(mean((Y.test - pred.tau)^2), 4) # 1.1457 (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 data are divided into a training sample with a size of 140 and a test sample with a size of 60. 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 3.3213, while the MSPE obtained from the robust method is 0.5925. 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(202)
sim.data <- generate.ff.data(n.pred = 5, n.curve = 200, n.gp = 101, out.p = 0.1)
out.indx <- sim.data$out.indx
# Response variable
Y <- sim.data$Y
# Predictor variables
X <- sim.data$X
# Split the data into training and test samples.
indx.test <- sample(c(1:200)[-out.indx], 60)
indx.train <- c(1:200)[-indx.test]
Y.train <- Y[indx.train,]
Y.test <- Y[indx.test,]
X.train <- X.test <- list()
for(i in 1:5){
  X.train[[i]] <- X[[i]][indx.train,]
  X.test[[i]] <- X[[i]][indx.test,]
}

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

# Perform classical functional principal component 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 functional principal component 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) # 1.5705 (classical method)
round(mean((Y.test - pred.MM)^2), 4) # 0.8166 (MM method)
```

Data analysis

We consider the MaryRiverFlow dataset available in the **robflreg** package to present the superiority of the robust functional principal component regression models over the classical model when the dataset includes outliers. The MaryRiverFlow dataset consists of hourly river-flow measurements obtained from January 2009 to December 2014 (6 years in total) in the Mary River, Australia. River-flow time series varies throughout the years due to the variation of the seasons and the amount of rainfall received at particular catchments. This problem still needs to be solved in the hydrological domain to be addressed appropriately using theoretical models.

Our first aim with the MaryRiverFlow dataset is to assess how the previous river flow curve time series, $\mathcal{X}_i(s)$, affects the current day's maximum river flow, Y_i. Here, the observations of predictor are assumed to be functions of hours, i.e., there are 2188 functional observations $\mathcal{X}_i(t)\big(1 \leq t \leq 24, i = 1, \ldots, 2188\big)$ while the response is a scalar predictor. A graphical display of the response and predictor is presented in Figure 7. From this figure, both the scalar response and functional predictor include clear outliers, which motivates us to apply the robust functional principal component regression models to better model this dataset.

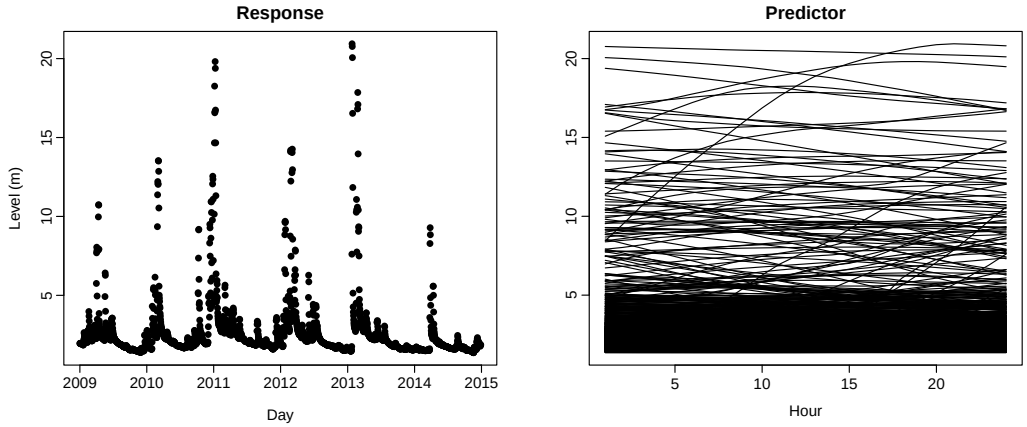

Figure 7: A plot of the daily maximum river flow measurement (left panel) and the functional time series plot (right panel) of the flow level in the Mary River.

Figure 7 can be produced by the following code:

```
library(robflreg)
library(fda.usc)
data("MaryRiverFlow")
X <- MaryRiverFlow[1:2188,]
Y <- apply(MaryRiverFlow[2:2189,], 1, max)
Day <- seq(as.Date("2009/01/01"), as.Date("2014/12/28"), by="days")
plot(Day, Y, type = "p", pch = 16, ylab = "Level (m)", main = "Response")
X <- fdata(X, argvals = 1:24)
plot(X, lty = 1, ylab = "", xlab = "Hour", col = "black",
     main = "Predictor", mgp = c(2, 0.5, 0))
```

We assume the SFLRM, $Y_i = \beta_0 + \int \mathcal{X}_i(s)\beta(s)ds$, where Y_i is the maximum river flow measurement for the current day and $\mathcal{X}_i(s)$ is the true river flow measurements recorded in the previous day. An expanding-window approach is considered to compare the predictive performance of the robust methods with the classical one. While doing so, The entire data are divided into two parts: a training

sample containing the days from 01/01/2009 to 20/12/2014 and a test sample including days from 21/12/2014 to 30/12/2014. First, the models are constructed using the entire training data to forecast the maximum river flow measurement on 21/12/2014. Then, the maximum river flow measurement is forecasted by increasing the training samples by one day. This procedure is repeated until the training samples cover the entire dataset. The MSPE is computed for each method, and the computed mean MSPEs ($\times 10^{-3}$) along with standard errors ($\times 10^{-3}$ given in bracket) are 7.559 (5.674), 1.635 (1.303), 1.671 (1.376), 1.617 (1.274), and 1.617 (1.274) for the classical, LTS, MM, S, and tau-estimator based SFLRM, respectively. From the results, all the robust methods produce significantly smaller MSPE values than the classical method. These results can be obtained by the following code:

```
library(robflreg)
data("MaryRiverFlow")
MaryRiverFlow <- as.matrix(MaryRiverFlow)
X <- list(MaryRiverFlow[1:2188,])
Y <- apply(MaryRiverFlow[2:2189,], 1, max)
gp <- rep(list(1:24), 1)

MSPE <- matrix(, ncol = 5, nrow = 10)
colnames(MSPE) <- c("classical","LTS","MM","S","tau")
starting_value <- 2178

for(i in 1:10){

# Divide the data into training and test samples
    Y.train <- Y[1:starting_value]
    Y.test <- Y[(starting_value+1)]

    X.train <- list(X[[1]][1:starting_value,])
    X.test <- list(matrix(X[[1]][(starting_value+1),], nrow = 1))

    # Perform classical and robust functional principal component regression models
    model.classical <- rob.sf.reg(Y.train, X.train, emodel = "classical", gp = gp)
   model.LTS <- rob.sf.reg(Y.train, X.train, emodel = "robust", fmodel = "LTS", gp = gp)
    model.MM <- rob.sf.reg(Y.train, X.train, emodel = "robust", fmodel = "MM", gp = gp)
    model.S <- rob.sf.reg(Y.train, X.train, emodel = "robust", fmodel = "S", gp = gp)
   model.tau <- rob.sf.reg(Y.train, X.train, emodel = "robust", fmodel = "tau", gp = gp)

    # Predict the maximum river flow measurement of the current day
    pred.classical <- predict_sf_regression(object = model.classical, Xnew = X.test)
    pred.LST <- predict_sf_regression(object = model.LTS, Xnew = X.test)
    pred.MM <- predict_sf_regression(object = model.MM, Xnew = X.test)
    pred.S <- predict_sf_regression(object = model.tau, Xnew = X.test)
    pred.tau <- predict_sf_regression(object = model.tau, Xnew = X.test)

    # Record the MSPE values
    MSPE[i,1] <- (Y.test - pred.classical)^2
    MSPE[i,2] <- (Y.test - pred.LST)^2
    MSPE[i,3] <- (Y.test - pred.MM)^2
    MSPE[i,4] <- (Y.test - pred.S)^2
    MSPE[i,5] <- (Y.test - pred.tau)^2
    starting_value <- starting_value + 1
}

apply(MSPE, 2, mean); apply(MSPE, 2, sd)
```

Our second aim with the MaryRiverFlow dataset is to assess how the previous river flow curve time series, $\mathcal{X}_i(s)$ affects the current day's river flow curve time series, $\mathcal{Y}_i(t)$. In this case, the elements of both the response and predictor are assumed to be functions of hours. Here, we assume the FFLRM, $\mathcal{Y}_i(t) = \beta_0(t) + \int \mathcal{X}_i(s)\beta(s,t)dsdt$. The similar expanding-window approach used in the SFLRM example is considered, i.e., the entire dataset is divided into two parts: a training sample containing the days from 01/01/2009 to 20/12/2014 and a test sample including days from 21/12/2014 to 30/12/2014. The functional principal component regression models are constructed using the entire training data to forecast the river flow curve time series on 21/12/2014. Then, the river flow curve time series is forecasted by increasing the training samples by one day. This procedure is repeated until the training samples cover the entire dataset. The MSPE is computed for each method, and

the computed mean MSPEs ($\times 10^{-3}$) along with standard errors ($\times 10^{-3}$ given in bracket) are 5.721 (3.884), 1.565 (0.965), 1.377 (0.847), 1.546 (0.949), 1.511 (0.920), and 1.377 (0.849) for the classical, MCD, MLTS, MM, S, and tau-estimator based FFLRMs, respectively. From the results, the robust methods produce improved MSPEs over the classical FFLRM, i.e., the robust method models the MaryRiverFlow data better than the classical functional principal component regression model. These results can be obtained by the following code:

```
library(robflreg)
data("MaryRiverFlow")
MaryRiverFlow <- as.matrix(MaryRiverFlow)

# The following function is used to obtain the functional response and predictor
var_fun = function(data,order){
  n = dim(data)[1]
  y = data[((order+1):n),]
  x=list()
  a=1
  b=order
  for(i in 1:order){
    x[[i]] = data[(a:(n-b)),]
    a = a+1
    b = b-1
  }
  return(list(x=x,y=y))
}

# Grid points for the functional response and predictor
gpY <- 1:24 # grid points of Y
gpX <- rep(list(1:24), 1) # grid points of Xs

MSPE <- matrix(, ncol = 6, nrow = 10)
colnames(MSPE) <- c("classical","MCD","MLTS","MM","S","tau")
starting_value <- 2179
# In two cases (when i = 3 and i = 6) the covariance is not decomposed.
# Thus, try() is used to ignore these two cases.
for(i in 1:10){
  try({
  data.i <- MaryRiverFlow[1:starting_value,]
  # Obtain the functional response and predictor
  XY = var_fun(data=data.i, order=1)
  X = XY$x
  Y = XY$y

  # Perform classical and robust functional principal component regression models
  model.classical <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "classical",
                        gpY = gpY, gpX = gpX)
  model.MCD <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "robust",
                        fmodel = "MCD", gpY = gpY, gpX = gpX)
  model.MLTS <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "robust",
                        fmodel = "MLTS", gpY = gpY, gpX = gpX)
  model.MM <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "robust",
                        fmodel = "MM", gpY = gpY, gpX = gpX)
  model.S <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "robust",
                        fmodel = "S", gpY = gpY, gpX = gpX)
  model.tau <- rob.ff.reg(Y = Y, X = X, model = "full", emodel = "robust",
                        fmodel = "tau", gpY = gpY, gpX = gpX)
  Xnew = list(t(as.matrix(Y[dim(Y)[1],])))

  # Predict the maximum river flow measurement of the current day
  predict.classical <- predict_ff_regression(object = model.classical, Xnew = Xnew)
  predict.MCD <- predict_ff_regression(object = model.MCD, Xnew = Xnew)
  predict.MLTS <- predict_ff_regression(object = model.MLTS, Xnew = Xnew)
  predict.MM <- predict_ff_regression(object = model.MM, Xnew = Xnew)
  predict.S <- predict_ff_regression(object = model.S, Xnew = Xnew)
```

```
  predict.tau <- predict_ff_regression(object = model.tau, Xnew = Xnew)

  # Record the MSPE values
  MSPE[i,1] <- mean((predict.classical - MaryRiverFlow[starting_value+1,])^2)
  MSPE[i,2] <- mean((predict.MCD - MaryRiverFlow[starting_value+1,])^2)
  MSPE[i,3] <- mean((predict.MLTS - MaryRiverFlow[starting_value+1,])^2)
  MSPE[i,4] <- mean((predict.MM - MaryRiverFlow[starting_value+1,])^2)
  MSPE[i,5] <- mean((predict.S - MaryRiverFlow[starting_value+1,])^2)
  MSPE[i,6] <- mean((predict.tau - MaryRiverFlow[starting_value+1,])^2)
  starting_value <- starting_value +1
  },silent=T)
}

apply(MSPE, 2, mean, na.rm=TRUE); apply(MSPE, 2, sd, na.rm=TRUE)
```

Conclusion

The R package **robflreg** provides an implementation of functional principal component regression model based on several robust procedures to fit and predict SFLRM and FFLRM. These methods are centered on the RFPCA of Bali et al. (2011), a popular robust dimension reduction technique in functional data, and several robust regression parameter estimators. In addition, the package **robflreg** allows us to fit and predict SFLRM and FFLRM via the classical FPCA and least-squares estimator. Several simulation examples and empirical data analysis show that the robust procedures provide better inference for the functional linear regression models when outliers are presented in the response and predictor variables. The functions for the package **robflreg** can be found on the code repository GitHub https://github.com/cran/robflreg.

The aspects that the current version of the R package **robflreg** can be improved in the future are listed below.

1) In the current version, the scalar predictors are allowed in the SFLRM, only. In the next versions of the package, it is planned to improve the interfaces `rob.ff.reg` and `predict_ff_regression` to include scalar predictors (whose effects are constant and/or vary along the continuum of the functional predictor).
2) The current package version does not allow for modeling the function-on-scalar regression model, where the response consists of random functions, and the predictors include scalar observations. The robust procedures used in the FFLRM are planned to extend the function-on-scalar regression model in the subsequent versions of the package.
3) In the current version, the observations of the functional variables are assumed to be densely observed. In the next versions of the package, the robust methods are planned to extend the models where the elements of functional variables can also be observed over irregular and curve-specific grids.

Acknowledgement

We thank three reviewers for their careful reading of our manuscript and valuable suggestions and comments, which have helped us produce an improved version of our manuscript and the R package. The first author was supported by The Scientific and Technological Research Council of Turkey (TUBITAK) (grant no: 120F270). The second author was partially supported by an Australian Research Council Discovery Project (grant no: DP230102250). This study is dedicated to the people who lost their lives in the earthquake that occurred in Turkey on February 6, 2023.

Bibliography

M. K. Ahn, J. D. Tucker, W. Wu, and A. Srivastava. Regression models using shapes of functions as predictors. *Computational Statistics and Data Analysis*, 151:107017, 2020. doi: https://doi.org/10.1016/j.csda.2020.107017. [p1]

A. Arribas-Gil and J. Romo. Shape outlier detection and visualization for functional data: the outliergram. *Biostatistics*, 15(4):603–619, 2014. doi: https://doi.org/10.1093/biostatistics/kxu006. [p12]

J. L. Bali, G. Boente, D. E. Tyler, and J.-L. Wang. The multivariate least-trimmed squares estimator. *Journal of Multivariate Analysis*, 99(3):311–338, 2008. doi: https://doi.org/10.1016/j.jmva.2006.06.005. [p2]

J. L. Bali, G. Boente, D. E. Tyler, and J.-L. Wang. Robust functional principal components: A projection-pursuit approach. *The Annals of Statistics*, 39(6):2852–2882, 2011. doi: https://doi.org/10.1214/11-AOS923. [p1, 6, 7, 8, 10, 19]

M. G. Ben, E. Martinez, and V. J. Yohai. Robust estimation for the multivariate linear model based on a τ scale. *Journal of Multivariate Analysis*, 97(7):1600–1622, 2006. doi: https://doi.org/10.1016/j.jmva.2005.08.007. [p2]

U. Beyaztas and H. L. Shang. On function-on-function regression: Partial least squares approach. *Environmental and Ecological Statistics*, 27(1):95–114, 2020. doi: https://doi.org/10.1007/s10651-019-00436-1. [p1]

U. Beyaztas and H. L. Shang. A partial least squares approach for function-on-function interaction regression. *Computational Statistics*, 36(2):911–939, 2021. doi: https://doi.org/10.1080/03610926.2022.2065018. [p10]

U. Beyaztas and H. L. Shang. A robust functional partial least squares for scalar-on-multiple-function regression. *Journal of Chemometrics*, 36(4):e3394, 2022. doi: https://doi.org/10.1002/cem.3394. [p1]

M. Bilodeau and P. Duchesne. Robust estimation of the sur model. *The Canadian Journal of Statistics*, 28 (2):277–288, 2000. doi: https://doi.org/10.2307/3315978. [p2]

G. Boente, M. Salibian-Barrera, and P. Vena. Robust estimation for semi-functional linear regression models. *Computational Statistics & Data Analysis*, 152:107041, 2020. doi: https://doi.org/10.1016/j.csda.2020.107041. [p1]

H. Cardot, F. Ferraty, and P. Sarda. Functional linear model. *Statistics and Probability Letters*, 45(1): 11–22, 1991. doi: https://doi.org/10.1016/S0167-7152(99)00036-X. [p1]

H. Cardot, F. Ferraty, and P. Sarda. Spline estimators for the functional linearmodel. *Statistica Sinica*, 13(3):571–591, 2003. [p1]

D. Chen, P. Hall, and H.-G. Müller. Single and multiple index functional regression models with nonparametric link. *The Annals of Statistics*, 39(3):1720–1747, 2011. doi: https://doi.org/10.1214/11-AOS882. [p1]

J. M. Chiou, Y. F. Yang, and Y. T. Chen. Multivariate functional linear regression and prediction. *Journal of Multivariate Analysis*, 146:301–312, 2016. doi: https://doi.org/10.1016/j.jmva.2015.10.003. [p1]

C. Croux and A. Ruiz-Gazen. High breakdown estimators for principal components: the projection-pursuit approach revisited. *Journal of Multivariate Analysis*, 95(1):206–226, 1996. doi: https://doi.org/10.1016/j.jmva.2004.08.002. [p1]

D. Şentürk and H.-G. Müller. Generalized varying coefficient models for longitudinal data. *Biometrika*, 95(3):653–666, 2008. doi: https://doi.org/10.1093/biomet/asn006. [p1]

A. Cuaves, M. Febrero, and R. Fraiman. Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics*, 22(3):481–496, 2007. doi: https://doi.org/10.1007/s00180-007-0053-0. [p12]

A. Cuevas. A partial overview of the theory of statistics with functional data. *Journal of Statistical Planning and Inference*, 147:1–23, 2014. doi: https://doi.org/10.1016/j.jspi.2013.04.002. [p1]

W. Dai and M. G. Genton. Multivariate functional data visualization and outlier detection. *Journal of Computational and Graphical Statistics*, 27(4):923–934, 2018. doi: https://doi.org/10.1080/10618600.2018.1473781. [p12]

W. W. Dou, D. Pollard, and H. H. Zhou. Estimation in functional regression for general exponential families. *The Annals of Statistics*, 40(5):2421–2451, 2012. doi: https://doi.org/10.1214/12-AOS1027. [p1]

I. L. Dryden and K. V. Mardia. *Statistical Shape Analysis, with Applications in R*. Wiley Series in Probability and Statistics, New York, 2016. [p1]

M. Febrero-Bande, P. Galeano, and W. Gonzalez-Mantelga. Outlier detection in functional data by depth measures, with application to identify abnormal NO_x levels. *Environmetrics*, 19(4):331–345, 2008. doi: https://doi.org/10.1002/env.878. [p12]

F. Ferraty and P. Vieu. *Nonparametric Functional Data Analysis*. Springer, New York, 2006. [p1]

J. Goldsmith, J. Bobb, C. M. Crainiceanu, B. Caffo, and D. Reich. Penalized functional regression. *Journal of Computational and Graphical Statistics*, 20(4):830–851, 2011. doi: https://doi.org/10.1198/jcgs.2010.10007. [p1]

J. Goldsmith, F. Scheipl, L. Huang, J. Wrobel, C. Di, J. Gellar, J. Harezlak, M. W. McLean, B. Swihart, L. Xiao, C. Crainiceanu, and P. T. Reiss. *refund: Regression with Functional Data*, 2022. URL https://CRAN.R-project.org/package=refund. R package version 0.1-28. [p1]

J. Harezlak, B. A. Coull, N. M. Laird, S. R. Magari, and D. C. Christiani. Penalized solutions to functional regression problems. *Computational Statistics and Data Analysiss*, 51(10):4911–4925, 2007. doi: https://doi.org/10.1016/j.csda.2006.09.034. [p1]

L. Horváth and P. Kokoszka. *Inference for Functional Data with Applications*. Springer, New York, 2012. [p1]

T. Hsing and R. Eubank. *Theoretical Foundations of Functional Data Analysis, with an Introduction to Linear Operators*. John Wiley & Sons, Chennai, India, 2015. [p1]

H. Hullait, D. S. Leslie, N. G. Pavlidis, and S. King. Robust function-on-function regression. *Technometrics*, 63(3):396–409, 2021. doi: https://doi.org/10.1080/00401706.2020.1802350. [p1]

A. E. Ivanescu, A. M. Staicu, F. Scheipl, and S. Greven. Penalized function-on-function regression. *Computational Statistics*, 30(2):539–568, 2015. doi: https://doi.org/10.1007/s00180-014-0548-4. [p1]

G. M. James. Generalized linear models with functional predictors. *Journal of Royal Statistical Society, Series B*, 64(3):411–432, 2002. doi: https://doi.org/10.1111/1467-9868.00342. [p1]

C. Jiang and J. L. Wang. Functional single index models for longitudinal data. *The Annals of Statistics*, 39(1):362–388, 2011. doi: https://doi.org/10.1214/10-AOS845. [p1]

I. Kalogridis and S. V. Aelst. Robust functional regression based on principal components. *Journal of Multivariate Analysis*, 173:393–415, 2019. doi: https://doi.org/10.1016/j.jmva.2019.04.003. [p1]

P. Kokoszka and M. Reimherr. *Introduction to Functional Data Analysis*. CRC Press, Boca Raton, 2017. [p1]

M. Koller and W. A. Stahel. Sharpening wald-type inference in robust regression for small samples. *Computational Statistics & Data Analysis*, 55(8):2504–2515, 2011. doi: https://doi.org/10.1016/j.csda.2011.02.014. [p1]

N. L. Kudraszow and R. A. Moronna. Estimates of mm type for the multivariate linear model. *Journal of Multivariate Analysis*, 102(9):1280–1292, 2011. doi: https://doi.org/10.1016/j.jmva.2011.04.011. [p2]

R. A. Maronna and V. J. Yohai. Robust functional linear regression based on splines. *Computational Statistics and Data Analysis*, 65:46–55, 2013. doi: https://doi.org/10.1016/j.csda.2011.11.014. [p1]

J. S. Marron, J. O. Ramsay, L. M. Sangalli, and A. Srivastava. Functional data analysis of amplitude and phase variation. *Statistical Science*, 30(4):468–484, 2015. doi: https://doi.org/10.1214/15-STS524. [p1]

H. Matsui, S. Kawano, and S. Konishi. Regularized functional regression modeling for functional response and predictors. *Journal of Math-for-Industry*, 1(A3):17–25, 2009. [p1]

F. A. Ocana, A. M. Aguilera, and M. Escabias. Computational considerations in functional principal component analysis. *Computational Statistics*, 22(3):449–465, 2007. doi: https://doi.org/10.1007/s00180-007-0051-2. [p8]

P. Raña, G. Aneiros, and J. M. Vilar. Detection of outliers in functional time series. *Environmetrics*, 26 (3):178–191, 2015. doi: https://doi.org/10.1002/env.2327. [p1]

J. O. Ramsay and C. J. Dalzell. Some tools for functional data analysis. *Journal of the Royal Statistical Society, Series B*, 53(3):539–572, 1991. doi: https://doi.org/10.1111/j.2517-6161.1991.tb01844.x. [p1]

J. O. Ramsay and B. W. Silverman. *Applied Functional Data Analysis*. Springer, New York, 2002. [p1]

J. O. Ramsay and B. W. Silverman. *Functional Data Analysis*. Springer, New York, 2006. [p1, 2]

J. O. Ramsay, S. Graves, and G. Hooker. *fda: Functional Data Analysis*, 2022. URL https://CRAN.R-project.org/package=fda. R package version 6.0.5. [p1]

P. T. Reiss and T. R. Ogden. Functional principal component regression and functional partial least squares. *Journal of the American Statistical Association: Theory and Methods*, 102(479):984–996, 2007. doi: https://doi.org/10.1198/016214507000000527. [p1]

P. J. Rousseeuw. Least median of squares regression. *Journal of the American Statistical Association: Theory and Methods*, 79(388):871–881, 1984. doi: https://doi.org/10.1080/01621459.1984.10477105. [p1]

P. J. Rousseeuw, K. V. Driessen, S. V. Aelst, and J. Agullo. Robust multivariate regression. *Technometrics*, 46(3):293–305, 2004. doi: https://doi.org/10.1198/004017004000000329. [p2]

M. Salibian-Barrera, G. Willems, and R. Zamar. The fast-tau estimator for regression. *Journal of Computational and Graphical Statistics*, 17(3):659–682, 2008. doi: https://doi.org/10.1198/106186008X343785. [p1]

F. Scheipl, A.-M. Staicu, and S. Greven. Functional additive mixed models. *Journal of Computational and Graphical Statistics*, 24(2):477–501, 2015. doi: https://doi.org/10.1080/10618600.2014.901914. [p1]

H. Shin and S. Lee. An RKHS approach to robust functional linear regression. *Statistica Sinica*, 26: 255–272, 2016. [p1]

A. Srivastava and E. P. Klassen. *Functional and Shape Data Analysis*. Springer, New York, 2016. [p1]

Y. Sun and M. G. Genton. Functional boxplots. *Journal of Computational and Graphical Statistics*, 20(2): 316–334, 2011. doi: https://doi.org/10.1198/jcgs.2011.09224. [p12]

J. D. Tucker, J. R. Lewis, and A. Srivastava. Elastic functional principal component regression. *Statistical Analysis and Data Mining*, 12(2):101–115, 2019. doi: https://doi.org/10.1002/sam.11399. [p1]

F. Yao, H.-G. Müller, and J.-L. Wang. Functional linear regression analysis for longitudinal data. *The Annals of Statistics*, 33(6):2873–2903, 2005. doi: https://doi.org/10.1214/009053605000000660. [p1]

V. J. Yohai. High breakdown-point and high efficiency estimates for regression. *The Annals of Statistics*, 15(2):642–665, 1987. doi: https://doi.org/10.1214/aos/1176350366. [p1]

H. Zhu, P. J. Brown, and J. S. Morris. Robust, adaptive functional regression in functional mixed model framework. *Journal of the American Statistical Association*, 106(1):1167–1179, 2011. doi: https://doi.org/10.1198/jasa.2011.tm10370. [p1]

Ufuk Beyaztas
Marmara University
Department of Statistics, Goztepe Campus, 34722, Kadikoy, Istanbul
Turkey
(ORCID 0000-0002-5208-4950)
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
(ORCID 0000-0003-1769-6430)
hanlin.shang@mq.edu.au