Journal of Statistical Software
July 2018, Volume 85, Issue 9. doi: 10.18637/jss.v085.i09

Generalization, Combination and Extension of Functional Clustering Algorithms: The R Package funcy

Christina Yassouridis
University of Natural Resources and Life Sciences, Vienna

Dominik Ernst
University of Natural Resources and Life Sciences, Vienna

Friedrich Leisch
University of Natural Resources and Life Sciences, Vienna

Abstract

Clustering functional data is mostly based on the projection of the curves onto an adequate basis and building random effects models of the basis coefficients. The parameters can be fitted with an EM algorithm. Alternatively, distance models based on the coefficients are used in the literature. Similar to the case of clustering multidimensional data, a variety of derivations of different models has been published. Although their calculation procedure is similar, their implementations are very different including distinct hyperparameters and data formats as input. This makes it difficult for the user to apply and particularly to compare them. Furthermore, they are mostly limited to specific basis functions. This paper aims to show the common elements between existing models in highly cited articles, first on a theoretical basis. Later their implementation is analyzed and it is illustrated how they could be improved and extended to a more general level. A special consideration is given to those models designed for sparse measurements. The work resulted in the R package **funcy** which was built to integrate the modified and extended algorithms into a unique framework.

Keywords: R, functional mixed models, functional clustering, generalization, sparse models.

1. Introduction

Clustering is a popular technique in statistics to find or impose a structure onto the data. Especially when the dataset is large, similar profiles merged to a group can lead to a deeper insight into the behavior of observations. Whereas for multidimensional data, a variety of different models exist, the choice for functional data is limited. The first intuition is to apply

standard methods such as k-means on the raw datasets by neglecting the time dependency and using the L^2 distance as a distance measure. Febrero-Bande and de la Fuente (2012) apply k-means on any of several semi-metrics for functional data defined by Ferraty and Vieu (2006). In the majority of cases though, basis functions are used as elements of the generation process in order to reflect the continuity in the data. The projection of the curves onto their basis functions is the core part of this process. Yao, Müller, and Wang (2005) use the Karhunen-Loève expansion as an adequate basis, James and Sugar (2003), Jacques and Preda (2013) and Bouveyron and Jacques (2011) use B-splines while Serban and Wasserman (2005) project the curves onto a Fourier basis. Also wavelets can be used as projection basis e.g., Giacofci, Lambert-Lacroix, Marot, and Picard (2013), Antoniadis, Brossat, Cugliari, and Poggi (2013) and Ray and Mallick (2006). The curve projection is further processed by one of various methods, for instance by building a functional mixed model (FMM). In this study, different existing models are put into a more generalized context by analyzing their underlying FMM. The aim of the article is to carve out the common elements of the algorithms and to show how they could be modified and extended. The work resulted in the R (R Core Team 2017) package **funcy** (Yassouridis 2018) which was built to simplify their usage, improve their comparability and to bring them to a level, where the same input leads at least to a similar output. Therefore we constructed a framework with the aim to *UNITE*:

- *U*nify: Definition of the same input and output structure for each model.
- *N*ame: Giving the same names for input and output parameters.
- *I*mprove: Increasing the efficiency of the algorithms.
- *T*rade: Using specific methods to initialize the parameter of others.
- *E*xtend: Extending the algorithms to more basis functions.

As some of the models are quite complex, their derivation is explained step by step, so that the article is organized in the following way: Section 2 introduces functional data models. Therefore, Section 2.1 gives a quick introduction to functional data leading to the functional mixed model in Section 2.2. While no assumption regarding specific basis functions has been made yet, Section 3 applies them to a spline and eigenbasis. To conduct the clustering task, two techniques are introduced in Section 4. The expansion to a functional mixed mixture model is investigated in Section 4.1. Alternatively, a distance measure based on the FMM is introduced in Section 4.2. Section 5 gives an overview of deviations of the models existing in the literature. Section 6 shows how the models could be modified and extended resulting in the package **funcy** and Section 7 explains the usage of **funcy**. Finally, Section 8 closes with a summary and an outlook.

2. Functional data for clustering

2.1. Introduction to functional data

Functional data is characterized by the following points:

- The original measurements exist on a continuous domain such as time or space.

- Repeated measurements on the same individual exist.
- The measurements can be available on a regular equidistant, regular non-equidistant, or on an irregular grid. Irregular or sparse means that the number and/or the location of time measurements differs between the subjects.
- The underlying generating processes are smooth and autocorrelated.

With the assumptions from above, a measurement y at point t_j can be expressed as:

$$y(t_j) = s(t_j) + \epsilon(t_j), \tag{1}$$

where $s(t)$ represents a smooth curve and $\epsilon(t)$ are i.i.d. random noise terms.

Basis expansion

With adequate basis functions $x_1(t), \ldots, x_p(t)$, $s(t)$ can be approximated by:

$$s(t) = \sum_{k=1}^{p} \psi_k x_k(t), \tag{2}$$

so that in matrix notation, y evaluated on $t_1, \ldots, t_M$, can be expressed as:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\psi} + \boldsymbol{\epsilon}. \tag{3}$$

$\boldsymbol{y}$ and $\boldsymbol{\epsilon}$ are M-dimensional vectors, $\boldsymbol{\psi}$ a p-dimensional vector and $\boldsymbol{X}$ is an $M \times p$-dimensional matrix. If the evaluation grid is sparse, time points can differ from one curve to the next so that instead of $t_1, \ldots, t_M$ we have curve specific evaluations $t_{i,i_1}, \ldots, t_{i,i_M}$. Bold lower case letters represent vectors while bold upper case letters stand for matrices in this and the following sections.

Regularization

In the regular case, the N curves evaluated on M time points can be written as $M \times N$-dimensional matrix $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ and the coefficients $\Psi = (\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_N)$ can be estimated simultaneously through the least squares approximation:

$$\hat{\boldsymbol{\Psi}} = (\boldsymbol{X}^\top \boldsymbol{X} + p\boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{Y}. \tag{4}$$

with a ridge term p (Hoerl and Kennard 1970). The reconstruction of the curves follows from:

$$\hat{\boldsymbol{Y}} = \boldsymbol{X}\hat{\boldsymbol{\Psi}}^\top. \tag{5}$$

Any orthogonal basis can be used for this procedure. If the dataset is irregular, the coefficients $\hat{\boldsymbol{\psi}}$ in (4) need to be calculated separately for each curve since $\boldsymbol{X}$ in (4) is evaluated on the curve specific grid. For the reconstruction in (5), $\boldsymbol{X}$ can be evaluated on a different grid, e.g., the unique union of the time points of all curves, so that the dataset can be regularized in this way. An alternative to the regularization by projection, is the interpolation of the data. For a linear interpolation and two data points $(t_k, y(t_k))$, $(t_{k+1}, y(t_{k+1}))$, $y(t)$ can be calculated by

$$y(t) = y(t_k)\frac{t_{k+1} - t}{t_{k+1} - t_k} + y(t_{k+1})\frac{t - t_k}{t_{k+1} - t_k}, \tag{6}$$

on each interval (t_k, t_{k+1}) for $k = 1, \ldots, M-1$.

If only few repeated and quite irregular spaced measurements are available per subject, the regularization of the curves through (5) leads to poor results with respect to the similarity to the original curves. Replacing (3) by a mixed effects model as explained in the following section turned out to be the better technique.

2.2. Functional mixed models

Following Rice and Wu (2001) any functional mixed model can be written in the form of

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\boldsymbol{\gamma} + \boldsymbol{\epsilon}, \tag{7}$$

where $\boldsymbol{X}$ and $\boldsymbol{Z}$ are basis functions, $\boldsymbol{\beta}$ is a fixed effect, $\boldsymbol{\gamma}$ a random effect and $\boldsymbol{\epsilon}$ are the i.i.d. measurement errors with mean zero and constant variance $\boldsymbol{\Xi}$. Furthermore $\mathsf{E}(\boldsymbol{\gamma}) = \mathsf{E}(\boldsymbol{\epsilon}) = 0$.

Robinson (1991) shows that if $\boldsymbol{\gamma}$ and $\boldsymbol{\epsilon}$ are assumed to have variances $\boldsymbol{\Gamma}\sigma^2$ and $\boldsymbol{\Xi}\sigma^2$ with some scalar σ, the multivariate normal distribution of $\boldsymbol{\gamma}$ and $\boldsymbol{y}$ can be expressed as

$$\begin{pmatrix} \boldsymbol{\gamma} \\ \boldsymbol{y} \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Gamma} & \boldsymbol{\Gamma}\boldsymbol{Z}^\top \\ \boldsymbol{Z}\boldsymbol{\Gamma} & \boldsymbol{Z}\boldsymbol{\Gamma}\boldsymbol{Z}^\top + \boldsymbol{\Xi} \end{pmatrix} \sigma^2 \right), \tag{8}$$

so that parameter vector $\boldsymbol{\gamma}$ conditioned on $\boldsymbol{y}$ has the following distribution:

$$\begin{aligned} (\boldsymbol{\gamma}|\boldsymbol{y}) &\sim \mathcal{N}\left(\boldsymbol{\Gamma}\boldsymbol{Z}^\top(\boldsymbol{Z}\boldsymbol{\Gamma}\boldsymbol{Z}^\top + \boldsymbol{\Xi})^{-1}\boldsymbol{y}, \quad (\boldsymbol{\Gamma} - \boldsymbol{\Gamma}\boldsymbol{Z}^\top(\boldsymbol{Z}\boldsymbol{\Gamma}\boldsymbol{Z}^\top + \boldsymbol{\Xi})^{-1}\boldsymbol{Z}\boldsymbol{\Gamma})\sigma^2 \right) \\ &= \mathcal{N}\left((\boldsymbol{Z}^\top\boldsymbol{\Xi}^{-1}\boldsymbol{Z} + \boldsymbol{\Gamma}^{-1})^{-1}\boldsymbol{Z}^\top\boldsymbol{\Xi}^{-1}\boldsymbol{y}, \quad (\boldsymbol{Z}^\top\boldsymbol{\Xi}^{-1}\boldsymbol{Z} + \boldsymbol{\Gamma}^{-1})^{-1}\sigma^2 \right). \end{aligned} \tag{9}$$

3. Applications to different basis functions

3.1. Application to a spline basis

Let $\boldsymbol{y}_i$ be the ith realization of the total process $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ evaluated on a possibly irregular grid t_{ij}. If (7) is applied to a B-spline basis, then $\boldsymbol{y}_i$ can be expressed as:

$$y_i(t_{i_j}) = \sum_{k=1}^{p} \bar{\beta}_k \bar{b}_k(t_{i_j}) + \sum_{k=1}^{q} \beta_{ik} b_k(t_{i_j}) + \epsilon(t_{i_j}) \quad \forall i \in 1 \ldots N, j \in 1 \ldots i_M, \tag{10}$$

$$\mathsf{E}(y_i(t_{i_j})) = \mu(t) = \sum_{k=1}^{p} \bar{\beta}_k \bar{b}_k(t_{i_j}) \quad \forall i \in 1 \ldots N, j \in 1 \ldots i_M. \tag{11}$$

$\boldsymbol{\beta}$ is the random effect with covariance $\boldsymbol{\Gamma}$, $\bar{\boldsymbol{B}}$ the spline basis for the mean, $\boldsymbol{B}$ the basis for a possibly different dimensional space of spline functions and ϵ are i.i.d. measurement errors with mean zero and constant variance σ^2. p and q are the dimension of the spaces.

3.2. Application to an eigenbasis

Another basis with nice properties are functional principal components. The variation of the data around the mean is explained by the first few components better than by any other basis. After the Karhunen-Loève theorem, if $y(t)$ is assumed to be a mean square continuous stochastic process ($\lim_{\epsilon \to 0} \mathsf{E}\left(y(t+\epsilon) - y(t)\right)^2 = 0$) and $y \in L^2(T)$, there exists an orthonormal basis of eigenfunctions (principal components) $K = (k_1, k_2, k_3, \dots) \in L^2$ such that each $\boldsymbol{y}_i$ can be approximated by the first p basis functions (Alexanderian 2013):

$$y_i(t_{i_j}) = \mu(t_{i_j}) + \sum_{k=1}^{p} \kappa_{ik} k_k(t_{i_j}) + \epsilon(t_{i_j}) \quad \forall i \in 1 \dots N, j \in 1 \dots i_M. \tag{12}$$

The following properties hold:

$$\begin{aligned} \mathsf{E}(y_i(t)) &= \mu(t) \quad \forall i \in 1 \dots N, \\ G(s,t) &= \mathsf{COV}\{y(s), y(t)\}, \\ \langle G(\cdot, t), k_k(\cdot) \rangle &= \lambda_k k_k(t), \\ \kappa_k \sim \mathcal{N}(0, \lambda_k) \quad &\text{and} \quad \mathsf{E}(\kappa_k, \kappa_l) = 0. \end{aligned}$$

$\langle ., . \rangle$ represents the inner product $\langle f, g \rangle := \int_T f(t) g(t)\, dt$. ϵ are i.i.d. measurement errors with mean zero and constant variance σ^2. The coefficients κ are calculated by

$$\kappa_{ik} = \langle y_i - \mu, k_k \rangle. \tag{13}$$

For sparse datasets, a vector of the pooled time points is created first and the mean and covariance matrix are smoothed on these points. For the calculation of the coefficients and the reconstruction of the curves, the eigenfunctions are reduced to the evaluations on the curve specific grid. The procedure from smoothing the pooled mean up to the calculation of the eigenbasis is demonstrated in Figure 1.

The coefficients are calculated by (13) through numeric integration. If there are many curves for which only few measurements are available, calculating the expected values of the coefficients, as shown in the following section, is more robust.

3.3. Estimation of the parameters

Let $\boldsymbol{\gamma}$ be the coefficients of the basis functions ($\boldsymbol{\beta}$ in Section 3.1 and $\boldsymbol{\kappa}$ in Section 3.2). After (9), the expected value of $\boldsymbol{\gamma}$ and its variance are given by:

$$\begin{aligned} \mathsf{E}(\boldsymbol{\gamma}_i | \boldsymbol{y}_i) &= \boldsymbol{\Gamma} \boldsymbol{Z}_i^\top \boldsymbol{\Sigma}_{Y_i}^{-1} (\boldsymbol{y}_i - \boldsymbol{\mu}_i), \\ \mathsf{VAR}(\boldsymbol{\gamma}_i | \boldsymbol{y}_i) &= \boldsymbol{\Gamma} - \boldsymbol{\Gamma} \boldsymbol{Z}_i^\top \boldsymbol{\Sigma}_{Y_i}^{-1} \boldsymbol{Z}_i \boldsymbol{\Gamma}. \end{aligned} \tag{14}$$

For Section 3.1, $\boldsymbol{\Gamma}$ in (14) is the covariance of the basis coefficients $\boldsymbol{\gamma}$, and $\boldsymbol{Z}_i$ of Equation 14 represents $\boldsymbol{B}$ evaluated on all available time points for curve i . The covariance matrix for y_i, $\boldsymbol{\Sigma}_{y_i} := \mathsf{COV}(y_i(t), y_i(s))$ is given by $\boldsymbol{\Sigma}_{y_i} = \boldsymbol{B}_i \boldsymbol{\Gamma} \boldsymbol{B}_i^\top + \sigma^2 \boldsymbol{I}_i$ but σ remains to be estimated.

In Section 3.2 $\boldsymbol{\Gamma} = \boldsymbol{\Lambda}$ so it is simply the diagonal matrix of the eigenvalues and $\boldsymbol{Z}_i = \boldsymbol{K}$ evaluated on time points for curve i. $\boldsymbol{\Sigma}_{y_i} = \boldsymbol{K}_i \Lambda \boldsymbol{K}_i^\top + \sigma^2 \boldsymbol{I}_i$. After Chiou and Li (2007) and Yao *et al.* (2005) the variance σ^2 of the error terms ϵ can be estimated by $\hat{\sigma}^2 = \frac{2}{T} \int_0^{\frac{T}{2}} (\hat{V}(t) - \hat{C}(t,t)) dt$, where $\hat{V}$ is a one-dimensional smoother of the variance and $\hat{C}$ is a two-dimensional

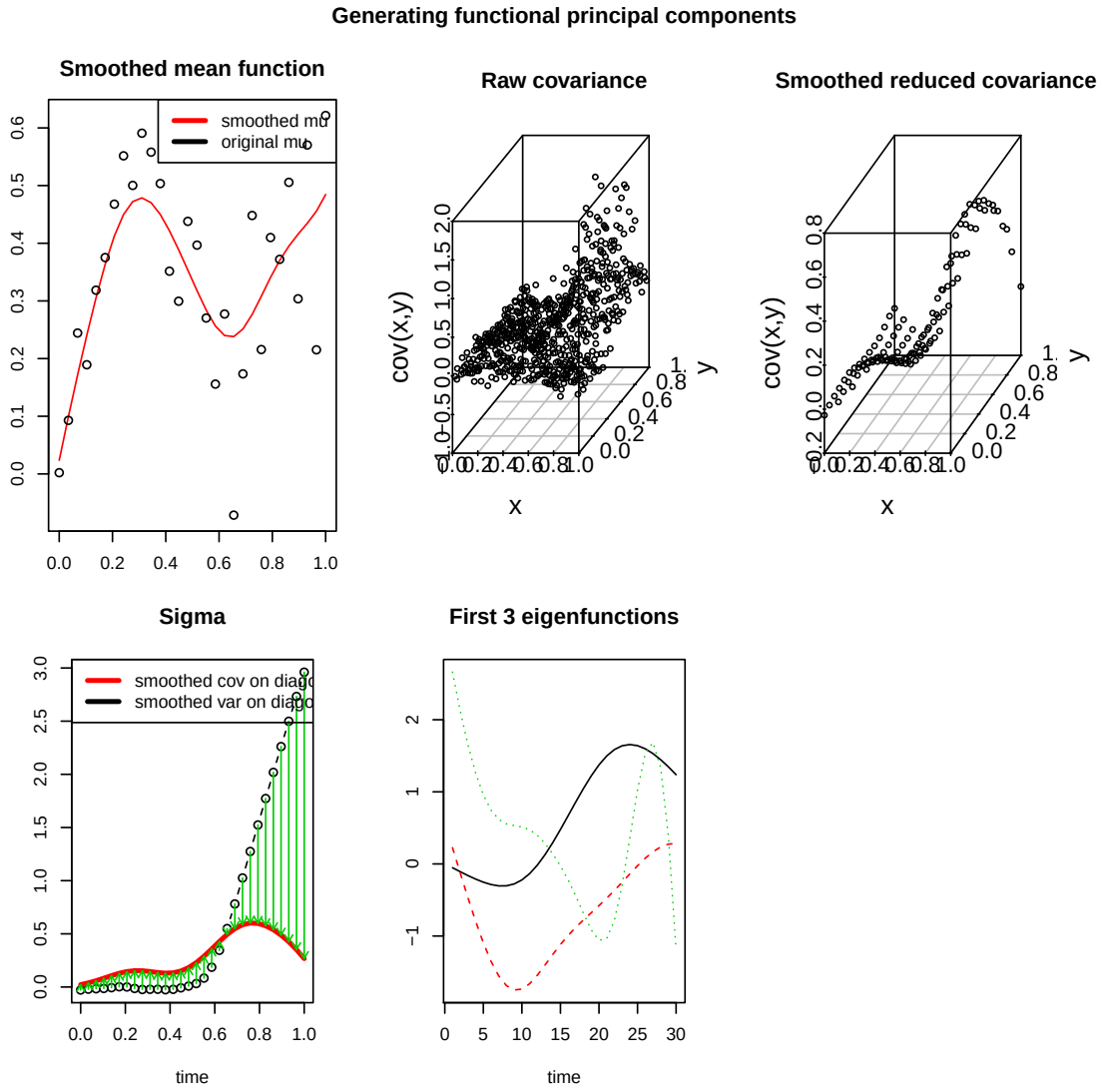

Figure 1: Calculation of the K-L expansion. The upper plots show the smoothing of the mean and the covariance matrix, the lower plots the calculation of the variance and the eigenfunctions.

smoother of the covariance at time point t. The estimation in (14) substitutes the numerical approximation of the integral (13). Yao *et al.* (2005) named their procedure PACE (principal components analysis through conditional expectation).

4. The clustering procedure

4.1. Functional mixed mixture models

To define a clustering problem, we assume that each $\boldsymbol{y}_i$ was generated by one of M different processes with known probability distributions. This leads to the log-likelihood definition of

a mixture model (Bilmes 1998):

$$\log(\mathcal{L}(\boldsymbol{\Theta}|\boldsymbol{Y})) = \log\left(\prod_{i=1}^{N} p(\boldsymbol{y}_i|\boldsymbol{\Theta})\right) = \sum_{i=1}^{N} \log\left(\sum_{j=1}^{M} \alpha_j p(\boldsymbol{y}_i|\boldsymbol{\theta}_j)\right). \tag{15}$$

$\boldsymbol{\Theta}$ represents the unknown parameter family $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\Xi}, \boldsymbol{\Gamma}, \sigma)$ of model (7) and $\boldsymbol{\alpha} = (\alpha_1 \dots \alpha_M)$. N is the number of observations, $\boldsymbol{\theta}_j$ are the parameters of the jth generating process and α_j is the probability that $\boldsymbol{y}_i$ is the outcome of process j. If the cluster membership was known, (i.e., $z_i = k$ if curve i is in class k), the complete log-likelihood could be calculated:

$$\log(\mathcal{L}(\boldsymbol{\Theta}|\boldsymbol{Y}, \boldsymbol{Z})) = \sum_{i=1}^{N} \log(\alpha_{z_i} p_{z_i}(\boldsymbol{y}_i|\boldsymbol{\theta}_{z_i})). \tag{16}$$

Since this is not the case in unsupervised learning, the expectation of the complete log-likelihood is built. With a current cluster configuration and estimation of the parameters $\boldsymbol{\Theta}^{g-1}$, the expected log-likelihood with respect to $\boldsymbol{Y}$ and $\boldsymbol{\Theta}^{g-1}$ is given by:

$$\mathsf{E}(\log(\mathcal{L}(\boldsymbol{\Theta}|\boldsymbol{Y}, \boldsymbol{Z})|\boldsymbol{Y}, \boldsymbol{\Theta}^{g-1})) = \sum_{i=1}^{N}\sum_{j=1}^{M} [\log(\alpha_j^{g-1}) + \log(p(\boldsymbol{y}_i|\boldsymbol{\theta}_j^{g-1}))] p(j|\boldsymbol{y}_i, \boldsymbol{\Theta}^{g-1}), \tag{17}$$

where $p(j|\boldsymbol{y}_i, \boldsymbol{\Theta}^{g-1})$ denotes the posterior probability that observation $\boldsymbol{y}_i$ is from cluster j given the current parameter estimates $\boldsymbol{\Theta}^{g-1}$.

If $\boldsymbol{\theta}_j$ is a random variable itself, the expected log-likelihood in (17) becomes:

$$\sum_{i=1}^{N}\sum_{j=1}^{M} [\log(\alpha_j^{g-1}) + \log(p(\boldsymbol{y}_i|\boldsymbol{\theta}_j^{g-1})) + \log(p(\boldsymbol{\theta}_j^{g-1}))] p(z_i = j|\boldsymbol{y}_i, \boldsymbol{\Theta}^{g-1}). \tag{18}$$

With an initial cluster configuration, the expectation-maximization (EM) algorithm consists in iteratively maximizing (18) and calculating the parameters α_j^{g-1} and $\boldsymbol{\theta}_j^{g-1}$ until a convergence criterion is met or the maximum iteration number g is reached. While the procedure of the EM algorithm stays the same for all model-based algorithms, $p(\boldsymbol{y}_i|\boldsymbol{\theta}_j)$ distinguishes between the different mixture models.

In the case of (7) for example, $\boldsymbol{\Theta}$ includes the random variable $\boldsymbol{\gamma}$. For $\boldsymbol{\Xi} = \boldsymbol{I}$, equal $\boldsymbol{\Gamma}$ in all classes, and in the hard classification case with $p(j|\boldsymbol{y}_i, \boldsymbol{\Theta}^{g-1}) = z_{ij}$, the complete log-likelihood of (18) would be:

$$-\sum_{i=1}^{N}\sum_{j=1}^{M} z_{ij} \log(\alpha_j) + \sum_{i=1}^{N}\sum_{j=1}^{M} z_{ij}(n_i \log \sigma^2 + \frac{1}{\sigma^2}\|\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Z}\boldsymbol{\gamma}\|^2) + \sum_{i=1}^{N}(\log \|\boldsymbol{\Gamma}\| + \boldsymbol{\gamma}_i^\top \boldsymbol{\Gamma}^{-1} \boldsymbol{\gamma}_i). \tag{19}$$

4.2. Distance measure

An alternative clustering technique to the mixture model is, to define a distance measure for the curves. The most common distance measure between continuous data is the one, induced

by the L^2 norm. If an orthogonal basis $\boldsymbol{B}$ in a Hilbert space exists, the L^2 norm of $y(t)$ can be written as:

$$\|y(t)\|^2 = \sum_{b \in B} |\langle y, b \rangle|^2. \tag{20}$$

With Parseval's identity, the distance between two curves $y_i(t)$ and $y_j(t)$ on an interval T becomes therefore:

$$D(y_i(t), y_j(t)) = \left(\int_T \|y_i(t) - y_j(t)\|^2 \, dt \right)^{\frac{1}{2}} = \left(\sum_{k=-\infty}^{\infty} |\gamma_{ik} - \gamma_{jk}|^2 \right)^{\frac{1}{2}}.$$

$y_i(t)$ and $y_j(t)$ might be available on an irregular or sparse grid only, so that the expected distance between the curves conditioned on the available knowledge $\boldsymbol{v}_i, \boldsymbol{v}_j$ can be defined by:

$$\{\mathsf{E}(D^2(y_i(t), y_j(t))|\boldsymbol{v}_i, \boldsymbol{v}_j)\}^{\frac{1}{2}} = \{\mathsf{E}(\sum_{k=-\infty}^{\infty} (\gamma_{ik} - \gamma_{jk})^2 | \boldsymbol{v}_i, \boldsymbol{v}_j)\}^{\frac{1}{2}} \tag{21}$$

$$= \{\sum_{k=-\infty}^{\infty} [\mathsf{VAR}(\gamma_{ik}|\boldsymbol{v}_i) + \mathsf{VAR}(\gamma_{jk}|\boldsymbol{v}_j) + (\mathsf{E}(\gamma_{ik}|\boldsymbol{v}_i) - \mathsf{E}(\gamma_{jk}|\boldsymbol{v}_j))^2]\}^{\frac{1}{2}}. \tag{22}$$

If we assume that the data follows the mixed model in (7), the estimations in (9) can be used for the calculation (22).

5. Existing models and algorithms

All of the following models are based on the FMM (7).

5.1. Functional mixed models and the EM algorithm

Bouveyron and Jacques (2011), James and Sugar (2003) and Jiang and Serban (2012) apply a functional mixed model onto a p-dimensional spline basis $\boldsymbol{B}$. For a curve belonging to class k, the simplified model becomes:

$$\boldsymbol{y}_i = \boldsymbol{B}(\boldsymbol{\mu}_k + \boldsymbol{\eta}_i) + \boldsymbol{\epsilon}_i. \tag{23}$$

Method `"funHDDC"`

Bouveyron and Jacques (2011) assume that

- $\boldsymbol{\eta} \sim \mathcal{N}(0, \boldsymbol{\Gamma}_k)$,
- $\boldsymbol{\mu}_k = \boldsymbol{U}_k \boldsymbol{m}_k$,
- $\boldsymbol{\Gamma}_k = \boldsymbol{U}_k \boldsymbol{A}_k \boldsymbol{U}_k^\top + \boldsymbol{\Xi}_k$, with $\boldsymbol{A}_k = \mathrm{diag}(a_{k1}, \ldots, a_{kd_k})$,
- $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Xi})$.

Furthermore $\boldsymbol{\Xi}_k$ is constructed in such way, that $\boldsymbol{Q}_k^\top \boldsymbol{\Gamma}_k \boldsymbol{Q}_k = \text{diag}(a_{k1}, \ldots, \alpha_{kd_k}, b_k, \ldots, b_k)$. Therefore the variance of each class k is modeled by $a_{k1}, \ldots, a_{kd_k}$ (with d_k representing the dimension of class k) while the variance of the noise is represented by b_k. $\boldsymbol{Q}_k = [\boldsymbol{U}_k, \boldsymbol{V}_k]$ is an orthogonal matrix comprising two parts: $\boldsymbol{U}_k$ of size $p \times d_k$ and $\boldsymbol{V}_k$ of size $p \times (p - d_k)$ with $\boldsymbol{U}_k^\top \boldsymbol{U}_k = \boldsymbol{I}_{d_k}$, $\boldsymbol{V}_k^\top \boldsymbol{V}_k = \boldsymbol{I}_{p-d_k}$ and $\boldsymbol{U}_k^\top \boldsymbol{V}_k = \boldsymbol{0}$.

Method `"fitfclust"`

In the model of James and Sugar (2003), the following assumptions are made:

- $\boldsymbol{\eta} \sim \mathcal{N}(0, \boldsymbol{\Gamma})$,
- $\boldsymbol{\mu}_k = \boldsymbol{\lambda}_0 + \boldsymbol{\Lambda}\boldsymbol{\alpha}_k$,
- $\boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$.

The first model is implemented in the R package **funHDDC** (Charles and Julien 2014). The dimension of the spline spaces can differ between the classes and is optimized during the algorithm. $\boldsymbol{U}$ and $\boldsymbol{Q}$ are orthogonal matrices resulting from a principal component analysis. The covariance matrix is class specific and diagonalizable. The second model includes the curve specific B-spline basis so that $\boldsymbol{B}$ in (23) is replaced by $\boldsymbol{B}_i$. This allows sparse measurements of the curves. The covariance matrix stays the same in each class but no specific structure is imposed to it. Both models feature the reduction of the original spline space to a lower space (dimension of $\boldsymbol{\mu}_k$). This allows a graphical interpretation for $d \leq 3$. The parameters including class affiliation are calculated by the EM algorithm. In the latter case, $\log(p(\boldsymbol{\theta}_j))$ in (18) is maximized by setting $\boldsymbol{\Gamma} = \frac{1}{N} \sum \mathsf{E}[\boldsymbol{\gamma}_i \boldsymbol{\gamma}_i^\top | \boldsymbol{y}_i]$ in (19) and the estimation in Section 3.3 is used for the calculation.

Method `"fscm"`

Jiang and Serban (2012) bring spatial dependency into the game. In the original model (23), $\boldsymbol{\epsilon}_i$ are not independent any longer but follow a normal distribution with a covariance depending on the local coordinates of $\boldsymbol{y}_i$. Additionally, $p(j|\boldsymbol{y}_i, \boldsymbol{\Theta})$ in (18) are not independent any longer but depend on the neighborhood ∂_i of the observation $\boldsymbol{y}_i$. Jiang and Serban (2012) define therefore a random Markov field where $p(z_{ik} = 1 | z_{\partial i})$ follows a Gibbs distribution. The local dependency can be dissolved by a Karhunen-Loève expansion, so that (23) is divided into a cluster specific temporal trend $\boldsymbol{B}\boldsymbol{\mu}_k$ and a spatial trend $\boldsymbol{Z}\boldsymbol{\eta}_i$:

$$\boldsymbol{B}\boldsymbol{\mu}_k + \boldsymbol{Z}\boldsymbol{\eta}_i + \boldsymbol{\epsilon}_i. \tag{24}$$

Thereby $\boldsymbol{\eta}_i$ represent the spatial random coefficients following a distribution $\mathcal{N}(0, \boldsymbol{\Sigma}_s)$ and $\boldsymbol{Z}$ are orthogonal basis functions retrieved from the Matérn covariance matrix of the local coordinates of the curves.

5.2. Functional mixed models with a robust distance measure: `"distclust"`

The parameter estimations in Section 3.3 used as input for the truncated version of (22) lead to the distance measure by Peng and Müller (2008):

$$\begin{aligned}\{\mathsf{E}(D^2(y_i(t), y_j(t)) | \boldsymbol{\upsilon}_i, \boldsymbol{\upsilon}_j)\}^{\frac{1}{2}} = \text{tr}(\boldsymbol{\Gamma} - \boldsymbol{\Gamma}\boldsymbol{Z}_i^\top \boldsymbol{\Sigma}_{y_i}^{-1} \boldsymbol{Z}_i \boldsymbol{\Gamma}) + \text{tr}(\boldsymbol{\Gamma} - \boldsymbol{\Gamma}\boldsymbol{Z}_j^\top \boldsymbol{\Sigma}_{y_j}^{-1} \boldsymbol{Z}_j \boldsymbol{\Gamma}) + \\ \|\boldsymbol{\Gamma}\boldsymbol{Z}_i^\top (\boldsymbol{Z}_i \boldsymbol{\Gamma} \boldsymbol{Z}_i^\top + \boldsymbol{\Xi})^{-1} (\boldsymbol{\upsilon}_i - \boldsymbol{\mu}) + \boldsymbol{\Gamma}\boldsymbol{Z}_j^\top (\boldsymbol{Z}_j \boldsymbol{\Gamma} \boldsymbol{Z}_j^\top + \boldsymbol{\Xi})^{-1} (\boldsymbol{\upsilon}_j - \boldsymbol{\mu})\|_2^2. \end{aligned} \tag{25}$$

The original algorithm applies multidimensional scaling on this distance and clusters the resulting independent variables by the k-means algorithm. Alternatively, the distance can serve as immediate input of other clustering algorithms such as partitioning around medoids or hierarchical clustering (Reynolds, Richards, de la Iglesia, and Rayward-Smith 2006).

5.3. Subspace projection: `"iterSubspace"`

The algorithm of Chiou and Li (2007) is the counterpart of the k-means algorithm for functional data. Alternating, curves are assigned to classes and classes are calculated anew depending on their assigned curves. Other than in k-means, the curves are not allocated to the cluster with the closest centroid but each curve is projected into all eigenspaces k by:

$$y_i^{(k)}(t_{ij}) = \mu^{(k)}(t_{ij}) + \sum_{j=1}^{p_k} \phi_{ij}^{(k)} \Phi_j^{(k)}(t_{ij}) \quad \forall i \in 1 \dots N, j \in 1 \dots i_M, \tag{26}$$

and assigned to the one according to $k^*(y) = \underset{k \in \{1,\dots,K\}}{\arg\min}\, D(y(t), y^{(k)}(t))$.

The method was originally implemented in MATLAB (The MathWorks Inc. 2014) and we transferred it to R. It is designed for sparse measurements, but not based on the conditional distance of (21).

6. The R package funcy

Not all of the described methods were available in form of executable code, and if so, they were naturally implemented in different manners. Sometimes the algorithms were used to introduce a new method but have their limits, e.g., when the user wants to try out other basis functions and it is difficult to compare the different implementations, since each requires different input parameters and input formats for the datasets. In the simplest case, the model based implementations have many parameters in common, but only different names for them. To overcome this problem, we created the R package **funcy** with the goal to *UNITE*. The particular steps to achieve this goal are described in the following with small code chunks for a better understanding. The detailed usage of **funcy** is explained in the next section.

6.1. Unify

The main function `funcit()`

In order to define the same input and output structures for the algorithms, wrapper functions were built around each one. Method calls are therefore the same for each model by using the function `funcit()` and specifying the desired method, e.g., `method = "fitfclust"`. More than one method can be used at a time by setting `method` to a vector of method names. Parallel processing by setting `parallel = TRUE` is optional. The result is a list with class '`funcyOutList`', accessible by all kinds of generic functions such as `summary()`, `calcTime()` or `plot()`.

Method `"funHDDC"` can result in an error caused by infinite values in the eigenvalue decomposition of the coefficient matrix Γ_k (see Section 5). In this case, we successively reduced the

number of clusters by one in the wrapper function until calculation was possible. Method "funclust" equally reduces the cluster number if the algorithm converged to a solution with an empty cluster. Therefore the final cluster numbers are not necessarily equal between the methods.

As an example, the genes dataset is clustered by the methods "fitfclust", "distclust" and "fscm" with four classes.

```
R> library("funcy")
R> data("genes", package = "funcy")
R> dat <- genes$data
R> res <- funcit(data = dat, methods = c("fitfclust", "distclust", "fscm"),
+    k = 4)
```

Input formats

The input data formats for the original methods varied a lot. Especially when the dataset is sparse, storage is not evident. funcit() allows two formats as input: A general format, "Format1", that is applicable to regular and irregular or sparse datasets, and "Format2" applicable to regular datasets only. In the first case, number and/or location of time points can differ, while for regular datasets, they are the same. funcit() recognizes the format and therefore the data regularity automatically. To switch back and forth between formats the function formatFuncy() can be used. A third format, "Format3", was included into formatFuncy(), consisting of a list of three matrices. One matrix Yin for the curves, one Tin for time points and one isobs, a matrix of indices. The reason we included "Format3" as well, is, because we found algorithms in the literature demanding for this particular input format. The original genes dataset is saved in a matrix in "Format2" of dimension 44×77 (44 time measurements for 77 curves).

```
R> dim(dat)

[1] 44 77
```

It is converted to format "Format1", a long matrix consisting of three columns: curve ID, curve evaluation and time points.

```
R> dat1 <- formatFuncy(dat, format = "Format1")
R> head(dat1, 6)

     curveIndx      yin tin
[1,]         1 -0.79346   1
[2,]         1 -0.72282   2
[3,]         1 -1.15070   3
[4,]         1 -1.25930   4
[5,]         1 -1.12930   5
[6,]         1 -1.82890   6
```

6.2. Name

`funcit()` takes as input single standard parameters such as the dataset `data` and number of clusters `k`. We categorized the models in Section 5 as model-based and non model-based. Parameters, other than the standard ones (e.g., base type or dimension of the basis), that are available for all methods, can be modified through the control class '`funcyCtrl`'. If the clustering algorithm is model-based, '`funcyCtrl`' is extended to '`funcyCtrlMbc`', where further parameters such as the convergence threshold `eps` for the EM algorithm, can be specified. The control class is an optional input argument for `funcit()`.

```
R> ctrl <- new("funcyCtrlMbc", baseType = "splines", dimBase = 3,
+    eps = 0.01)
R> res <- funcit(data = dat, methods = "fitfclust", k = 4,
+    funcyCtrl = ctrl)
```

All other method specific parameters (not standard and not in the control class) can be passed as `...` arguments to `funcit()`, as long as only one method is called. '`funcyOutList`' is a list of '`funcyOut`' objects, one for each method. Similar as in the control class, common results for all methods such as cluster assignment or calculation time are saved in a simple '`funcyOut`' object. Method specific results, e.g., AIC for model-based methods are stored in extensions to '`funcyOut`'. The dataset is saved in '`funcyOutList`' by setting `save.data` to `TRUE` when `funcit()` is invoked. Saving the data is necessary to use some of the plot functions.

6.3. Improve

Some of the original algorithms could be improved in terms of efficiency. Method `"fitfclust"` is applicable to sparse datasets. Its original version required a list of three vectors consisting of the curve ID, the time points and the measurements of all curves concatenated to vectors. All calculation was furthermore based on this input format. But if the dataset is regular, matrix storage is possible and matrix operations can be used instead. We re-implemented all functions of method `"fitfclust"` for the regular case. As R is a vectorized language this speeds up the calculation significantly (factor 20).

Chiou and Li (2007) used two for-loops in their algorithm `"distclust"` (see Section 5.3) originally implemented in MATLAB. One for the curves and one for the subspace projection, i.e., each curve is projected onto all subspaces and the distance is stored. Exchanging these loops increases the system time since the subspace projection is, what makes the calculation costly. For a regular dataset, the algorithm is then identical to the function `kmPCA()` in the R package **modelcf** (Auder 2012).

6.4. Trade

In order to calculate the distance in Section 4.2, the parameters $\mathbf{\Gamma}$ and σ need to be estimated. If the base type is an eigenbasis, estimation follows the procedure as explained in Section 3.3. For base types other than the eigenbasis, we applied the EM algorithm of James and Sugar (2003) for the functional mixture model, and set the class number to one. The double sum in (15) reduces therefore to a single sum ($\alpha_1 = 1, \alpha_2, \ldots, \alpha_k = 0$) and $\mathbf{\Gamma}$ and σ can be used as input parameters for (25).

6.5. Extend

Multiple base types

The estimation of the parameters in Section 3.3 is not restricted to a certain basis, so that we could complete all implementations of the mixed models with new base types. We used B-spline, exponential, Fourier, polynomial and power bases from the R package **fda** (Ramsay, Wickham, Graves, and Hooker 2014). To implement the distance calculation in Section 4.2 for these base types, a singular value decomposition was applied to them, to orthogonalize them.

Outsourcing and smoothing

For the calculation of the the scores of an eigenbasis, a smoothing technique as explained in Section 3.2 needs to be applied to the mean and covariance matrix. Therefore, smoothing parameters and a smoother have to be defined. Chiou and Li (2007) implement their own smoothers `sm1()` and `sm2()` while Peng and Müller (2008) use a non-parametric regression `sm.regression()` already implemented in the R package **sm** (Bowman and Azzalini 2014). While `sm.regression()` is applied on the raw mean and covariance function in `"distclust"`, Chiou and Li (2007) calculate the average mean and average covariance matrix before applying their smoothers (differs only for repeated measurement). If the curves do not have many common time points, the smoothing procedure can take very long (curves must be evaluated on every unique point). Therefore the evaluation points should be reduced to a maximum number. The decomposition of a stochastic process into its Karhunen-Loève expansion (see Figure 1) was part of the original algorithms `"distclust"` and `"iterSubspace"` (Peng and Müller 2008; Chiou and Li 2007) since both of them were based on an eigenbasis. We decided to simplify and outsource this piece of code including the reduction of evaluation points, so that each method can have access to it. On the other hand, by this, `"distclust"` and `"iterSubspace"` are not restricted to an eigenbasis any longer. For the smoothing step, we created a functional principal component control class '`fpcCtrl`' including all necessary parameters for the K-L calculation such as smoothing technique, number of maximal evaluation points, kernel width etc. Such as '`funcyCtrl`', '`fpcCtrl`' is an optional input argument for `funcit()`. It is ignored if the base type is not set to `"eigenbasis"`. As mentioned in Section 3.3, the calculation of the coefficients by numerical integration (see (13)) leads to poor results, if the dataset contains many observations with only few measurements. Therefore the conditional estimation (see Section 3.3) is the better choice. We included the idea of calculating the basis coefficients this way, so that the user can decide if they want to execute the calculation by an integration or an estimation step. This option is again part of the '`fpcControl`' class.

Regularization

We implemented regularization through basis projection, as introduced in Section 2.1. The ridge term p in Equation 4 was set to 0.01 but any value would have been possible. For interpolation, the R function `approxfun` was used. For evaluation outside the input point, the points were extrapolated to the closest boundary point. The function `regFuncy()` performs the regularization step with one of the two methods. Therefore `method` can be set to `"interpolate"` or `"project"` (see Section 7 for an example of usage).

Plotting

We provided different plots for the 'funcyOutList' objects which can be called by setting the type in the function plot(). Method independent plot types are "all", "centers", "shadow", "dist2centers". While the first two generate a multiple plot figure with one plot for each method, the latter two refer to only one method which must be specified by select. No specification selects the first method per default. Plot type "all" generates plots with curves, cluster centers and the corresponding cluster labels. The clusters had been relabeled according to the minimal L^2 distance between their centers before, so that identical colors correspond to similar clusters. Plot type "centers" limits the plots to their cluster centers. Type "dist2centers" shows one plot of the curves for each cluster. The thickness of the lines corresponds to the distance of the curves to their cluster centers. For the maximal distance, the line width is close to 0 while for the minimal distance it is 1. Another plot type that can be used is the "shadow" plot imported from package **flexclust** (Leisch 2006, 2010). The shadow plot relates the distance from the curves to their closest and second closest cluster center by a shadow value $s(x)$. If $s(x)$ is close to 0, the curve is very close to its cluster center, if it is 1, the point is in the middle of its closest and second closest centers. Well separated clusters are distinguished by many subjects with small shadow values $s(x)$. Again, the distance measure refers to the L^2 distance. If the base type for the curve projection had been set to an eigenbasis (see Section 7), the process of calculating the bases can be plotted. Figure 1 for example was generated by setting the type to "fpc".

Cluster methods "fitfclust" and "fscm" provide additional plots. For method specific plots, the plot function must be limited to the method by select = "fitfclust". Method "fitfclust" provides confidence intervals for the clusters by setting type to "conf". Additionally discriminant functions can be plotted showing the time intervals where the highest discrimination between the clusters occurs (James and Sugar 2003). Method "fscm" provides the types "deviations", "locations" and "overview". The first one plots the location of the points colored according to their spatial dependency. The darker the coloring of the points the higher the dependency value. Type "locations" shows the original location of the points and the configuration of the points after applying multidimensional scaling on the the Matérn covariance matrix (see Section 5.1). The "overview" plot shows 4 plots. One with the curve locations, one showing the clustered temporal trends, one including the overall trend and finally the spatial dependency plot. Examples of using the method independent and method dependent plot function are given below.

```
R> plot(res, type = "centers")
R> plot(res, select = "fitfclust", type = "conf")
```

Method	Advantage	Disadvantage
"fitfclust"	High performance in run time and cluster results Predicted curves Confidence Intervals	Same dimension for clusters
"distclust"	Very fast for eigenbasis	Same dimension for clusters
"iterSubspace"	Varying dimensions for clusters	Quite slow for irregular data

Table 1: Advantages vs. disadvantages of the clustering algorithms applicable to sparse data.

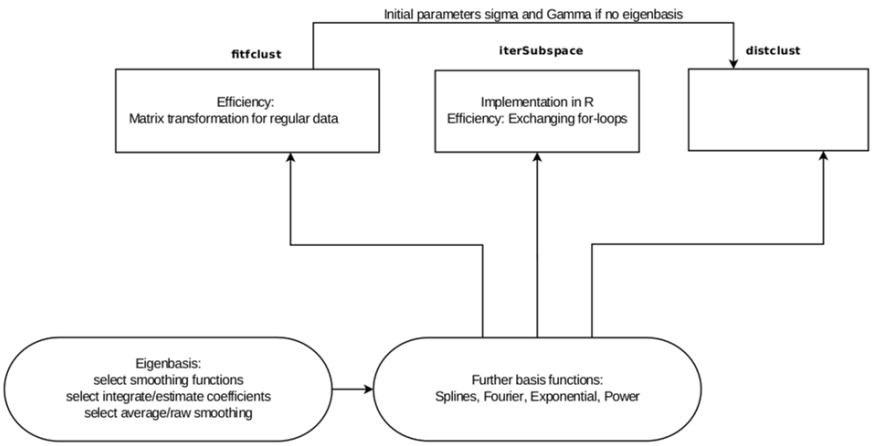

Figure 2: Structural plot of modifications. Model specific modifications are explained in boxes while modifications for all methods are explained in ellipses.

6.6. Summary

The ideas of *UNITE* are summarized in Figure 2. The graphic shows only the methods for sparse measurements since those are the ones that we applied major modification to. The advantages and disadvantages of each method are listed in Table 1.

7. Using funcy

We included seven clustering models into the package **funcy**. The first three (`"fitfclust"`, `"distclust"`, `"iterSubspace"`) are applicable to sparse datasets, while the others work exclusively on regular data. The package could be an incentive to put new models for clustering functional data into this framework, in order to clarify their methodology and to simplify their usage. In the following we will show how **funcy** is used by applying functions to a simulation example first and then to a real life dataset.

7.1. Simulated dataset

Simulating datasets

As a data example, we generated an irregular set of 100 curves evaluated on 3 to 10 time points per curve that were derived from 5 data generating processes. A function to simulate such a dataset is `sampleFuncy()`. It can generate up to 6 classes. Cluster centers are drawn randomly from the functions in Table 2. For the generation of the curves, a normally distributed error term was added with standard deviation `sd` which can be specified by the user.

The result is an object of class '`sampleFuncy`'. Data and clusters can be accessed by the corresponding functions.

$f_1(x) = x^2$	$f_2(x) = \sqrt{x}$	$f_3(x) = \sin(2\pi x)$	$f_4(x) = x^3$	$f_5(x) = -x^2$	$f_6(x) = x - 1$

Table 2: Simulated cluster centers.

```
R> set.seed(2705)
R> ds <- sampleFuncy(obsNr = 100, timeNrMin = 3, timeNrMax = 10, k = 5,
+    sd = 0.4, reg = FALSE)
R> dat <- Data(ds)
R> cls <- Cluster(ds)
```

Clustering the data

The dataset is now clustered by all methods usable for sparse datasets and a summary of the results is printed. `summary()` is a print method to show the cluster proportions, Rand indices (Hubert and Arabie 1985) and calculation times for the different models. If true cluster membership was given as input for `funcit()`, the diagonal in the Rand indices matrix shows the performance of the methods. Otherwise all diagonal elements are `NA`s. Each non-diagonal entry indicates the similarity between the methods.

In our example, method `"fitfclust"` performs best on this dataset since its value on the diagonal is the highest.

```
R> res1 <- funcit(data = dat, methods = c("fitfclust", "distclust",
+    "iterSubspace"), seed = 2804, clusters = cls, k = 5, save.data = TRUE,
+    parallel = TRUE)
R> summary(res1)

'funcyOutList' object with called algorithm(s):

fitfclust distclust iterSubspace

call:
funcit(data = dat, k = 5, methods = c("fitfclust", "distclust",
    "iterSubspace"), seed = 2804, clusters = cls, parallel = TRUE,
    save.data = TRUE)

Summary of the Cluster Proportions:
                 V1     V2     V3     V4     V5
fitfclust    0.1956 0.2197 0.3027 0.1615 0.1205
distclust    0.1900 0.1100 0.2000 0.3000 0.2000
iterSubspace 0.2100 0.1900 0.2700 0.1400 0.1900

Summary of the Rand Indices:
             fitfclust distclust iterSubspace
fitfclust       0.5372    0.6299       0.4141
distclust       0.6299    0.4502       0.4972
iterSubspace    0.4141    0.4972       0.3357

Summary of the Calculation Time:
             user.self sys.self elapsed user.child sys.child
fitfclust        2.404    0.192   2.657      0.024     0.004
distclust        4.228    0.108   4.360      0.000     0.000
iterSubspace     0.776    0.056   0.871      0.000     0.000
```

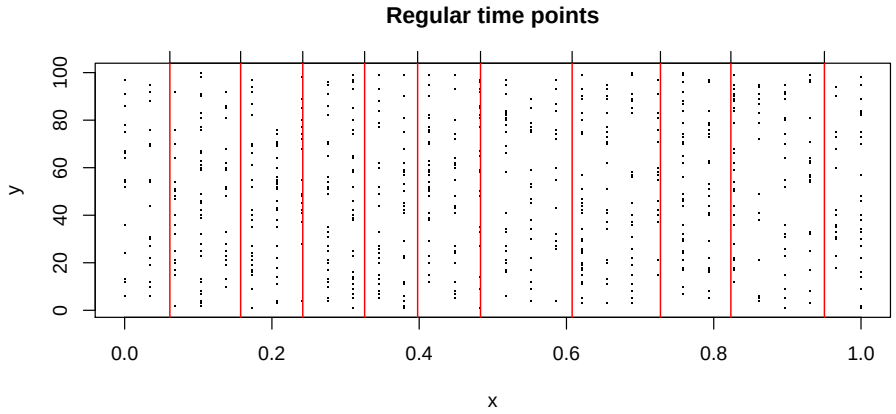

Figure 3: Evaluation grid generated from irregular points by the function `makeCommonTime()`.

Regularization

The dataset can be regularized by using the function `regFuncy()`. We use the method `"interpolate"`. `regFuncy()` implements a process where an evaluation grid is calculated based on the dataset. To see an example for such a grid we can use the `makeCommonTime()` function (Figure 3).

```
R> set.seed(2705)
R> makeCommonTime(dat, timeNr = 10, plot = TRUE)

 [1] 0.06139613 0.15752351 0.24188641 0.32589588 0.39787798 0.48328109
 [7] 0.60770791 0.72763618 0.82327586 0.95029109
```

For the regular dataset, further clustering algorithms become available: `"fscm"`, `"funclust"` and `"funHDDC"`. To use the latter two, the R packages **Funclustering** (Soueidatt 2014) and **funHDDC** (Charles and Julien 2014) must be installed first. We regularize the data and cluster the new dataset with all available methods. A plot of the clustered data is shown in Figure 4. Note that the class number was reduced for methods `"funclust"` and `"funHDDC"` in this example (see Section 6.1).

```
R> datReg <- regFuncy(dat, method = "interpolate", plot = FALSE)
R> res2 <- funcit(data = datReg$data, regTime = datReg$time, methods = "ALL",
+    seed = 2805, clusters = cls, k = 5, save.data = TRUE, parallel = TRUE)
R> plot(res2, legendPlace = "top")
```

Other base types

The default base type is a B-spline basis. Let us now use a Fourier basis with 7 basis functions. Therefore a '`funcyCtrl`' object is defined and the original dataset is clustered with method `"fitfclust"`.

```
R> ctrl <- new("funcyCtrl", baseType = "fourier", dimBase = 7)
R> res3 <- funcit(data = dat, methods = "fitfclust", seed = 2806,
+    clusters = cls, k = 5, funcyCtrl = ctrl)
```

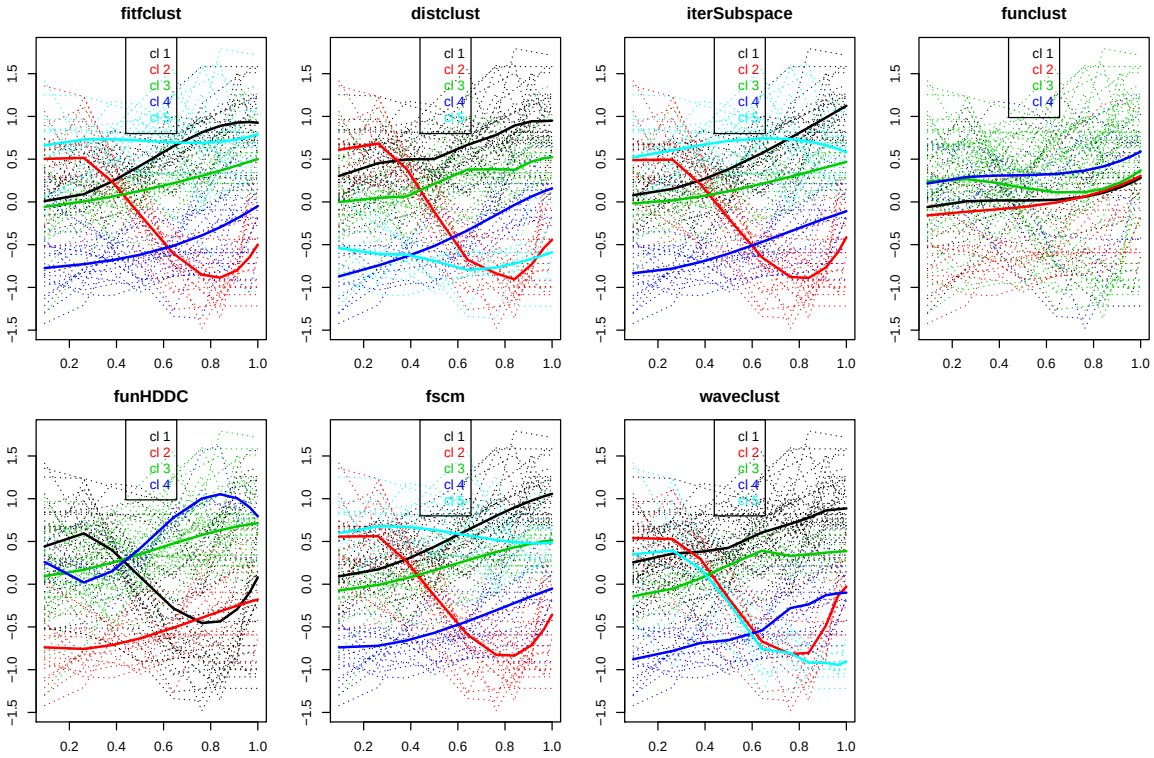

Figure 4: Simulated data with 5 cluster centers, clustered by all available methods from **funcy** with an initial class number of 5.

Alternatively we can use an eigenbasis and specify smoothing parameters for the calculation procedure. The resulting center functions for both base types are shown in Figure 5.

```
R> ctrl@baseType <- "eigenbasis"
R> fpcCtrl <- new("fpcCtrl", h1Dim = 0.01, coeffsCalc = "estimate")
R> res4 <- funcit(data = dat, regTime = datReg$time, methods = "fitfclust",
+    seed = 2807, k = 5, clusters = cls, funcyCtrl = ctrl, fpcCtrl = fpcCtrl)
R> plot(res3, type = "centers")
R> plot(res4, type = "centers")
```

If we would like to study the distances of the curves to their centers we can take a look at the plot type `"dist2centers"` . Each cluster is plotted separately. Thick lines mean that curves lie close to the centers while thinner lines indicate, that they are further away (see Figure 6).

```
R> plot(res4, type = "dist2centers")
```

7.2. Real life dataset

As an example for a real life dataset we clustered the irregular dataset `bones` with the available methods for sparse data and plotted the results. The clustered data is shown in Figure 7.

```
R> data("bones", package = "funcy")
R> dat <- bones$data
```

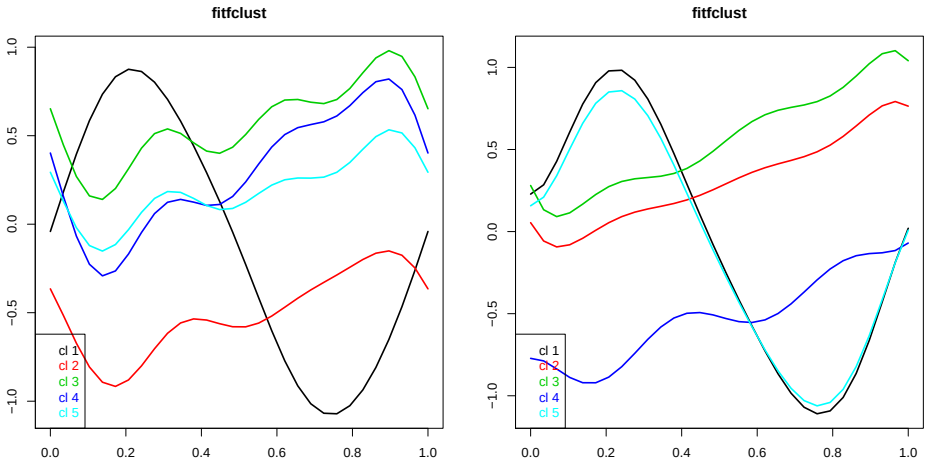

Figure 5: Cluster centers for simulated data, clustered by `"fitfclust"` with a fourier and eigenbasis.

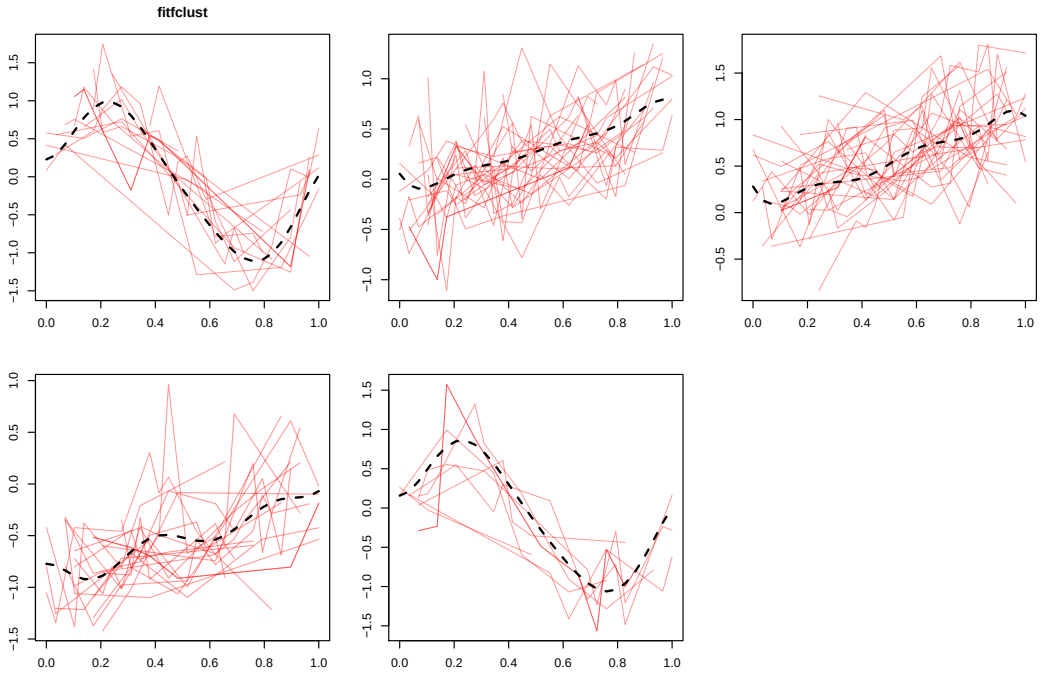

Figure 6: Separate plots for each cluster with line width corresponding to L^2 distance to the corresponding cluster center.

```
R> res5 <- funcit(data = dat, methods = 1:3, seed = 28, k = 2,
+    parallel = TRUE)
R> plot(res5)
```

To investigate the distances from the curves to their centers in this example, an alternative to the plot type `"dist2centers"` is used – the `"shadow"` plot (see Figure 8). We see, that `"fitfclust"` separates the curves the best. As a last example we want to use the spatial cluster algorithm `"fscm"` and apply it to the dataset `lowflow`. The dataset consists of 82

gauges in Upper Austria where stream flow minima were measured over 33 winters (1976–2008). For the `lowflow` data we have a data and a location matrix where the coordinates of the measuring points are stored.

```
R> data("lowflow", package = "funcy")
```

We choose the algorithm `"fscm"` since it is applicable to spatial dependent data. As spatial input parameter we need to pass the `location` matrix.

```
R> res6 <- funcit(data = lowflow$data, location = lowflow$location,
+    methods = "fscm", seed = 2809, k = 5)
```

Once we have clustered the data, we can plot the result with the method-specific plot-type `"overview"`. We get a plot of the location colored according to their clusters, the separate

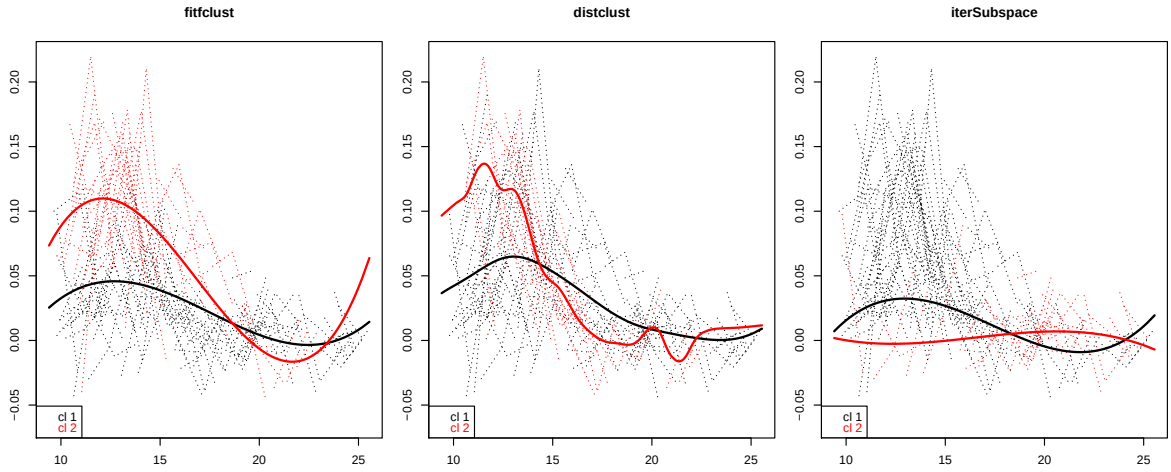

Figure 7: The 2 clusters of the `bones` data with center curves clustered by methods `"fitfclust"`, `"distclust"` and `"iterSubspace"`.

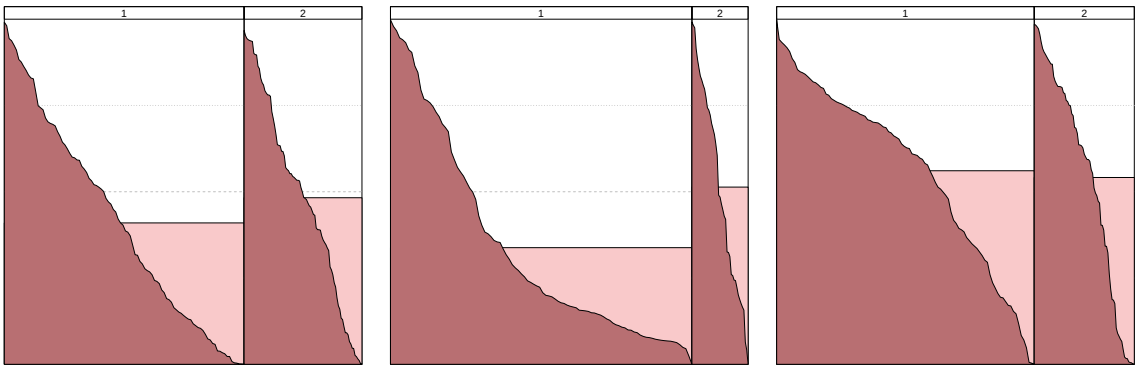

Figure 8: Shadow plots for the clustered `bones` data with methods `"fitfclust"`, `"distclust"` and `"iterSubspace"`. The first plot separates the curves the best since shadow values are close to zero for most points and close to 1 for only few.

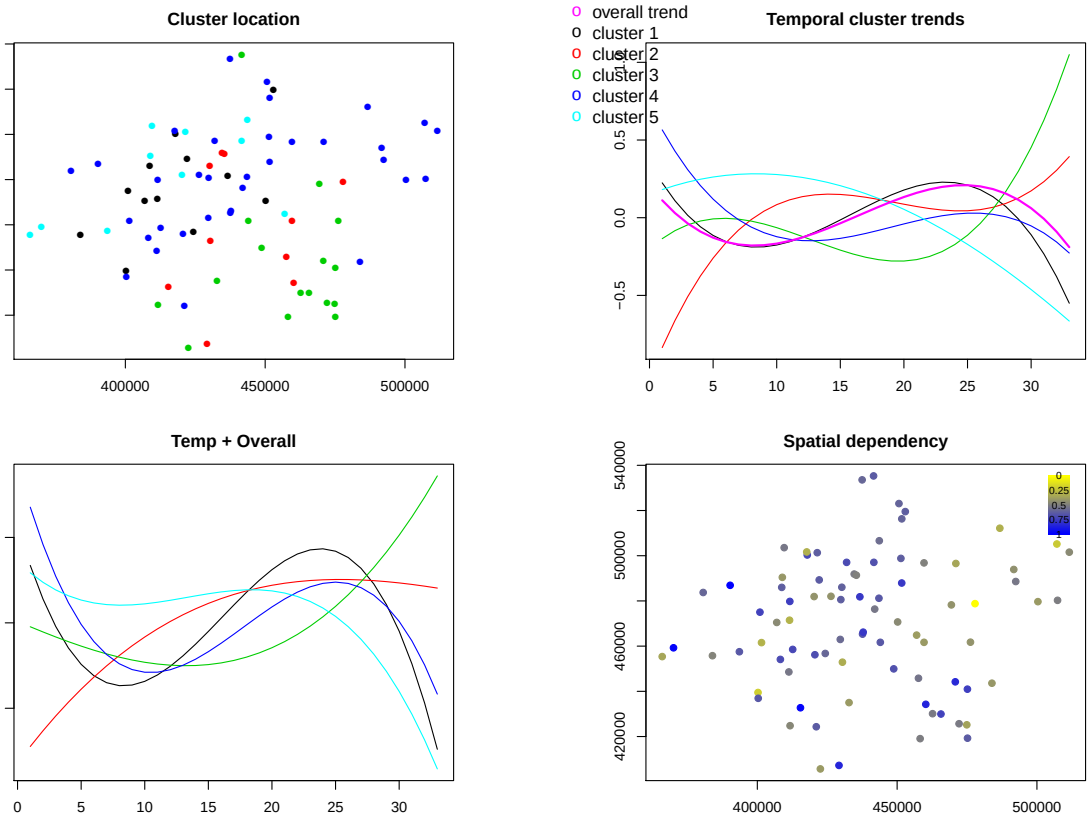

Figure 9: The `type = "overview"` plot of the clustered `lowflow` data with method `"fscm"`. From left to right, curve locations with cluster assignments, temporal cluster trends, overall plus temporal cluster trends and the spatial dependency plot.

temporal and overall cluster trends, the final cluster centers and a spatial plot (see Figure 9). The spatial cluster plot indicates where spatial dependency is low and high, ranging from yellow to blue.

```
R> plot(res6, type = "overview")
```

8. Summary

As in the multidimensional case, there is no single, best method for functional data. Dozens of models exist in the literature and it is neither easy to find the appropriate one for a specific dataset, nor to use it. Nevertheless, a structure can be imposed to them because most methods are based on the projection of the curves to certain basis functions and building a functional mixed model (FMM). The article showed how existing algorithms in the literature, that seem very different at first sight, actually do quite similar things. Therefore they were broken down to their core part, the functional mixed model. With this knowledge in mind, the methods could be modified, extended and combined. This led to a greater efficiency and flexibility in the clustering procedure. The R package **funcy** was built with the goal to *UNITE* all of

the introduced models and is available on the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=funcy. For example, each algorithm was put inside a wrapper function so that all control parameters for the FMM have the same names. To account for irregular datasets, two different formats can be used as input for the models and they were extended to different basis functions. To compare the algorithms, generic functions such as `calcTime()` for the calculation time or `randIndex()` showing the Rand indices between all pairs of methods, were implemented. We hope that this work leads to a greater flexibility, comparability and finally to more robust results, since the user is not restricted to one solution any longer.

9. Future work

Model selection

The package **funcy** does not contain a method to find the best number of clusters. However, it could help to find a good one by trying out different cluster numbers for more than one algorithm and using the one, where they show the most agreement. Nevertheless similar methods might lead to high Rand indices although the cluster number was set to a false number. The model-based algorithms `"fitfclust"`, `"funclust"`, `"funHDDC"` and `"waveclust"` provide the AIC and BIC which could be used for model selection. In our experiences they were not very reliable since they led to a wrong number of clusters in many cases. A structural study evaluating indicators such as the Rand indices between the methods or the AICs for different cluster numbers could be helpful. Finding the right number of clusters remains an open question up to now.

PACE

As mentioned in Section 5.3, the principal components analysis through conditional expectation (PACE) was originally implemented in MATLAB. We had built an interface in form of the R package **funcyOctave** so that we could use it for the regularization of sparse datasets. However **funcyOctave** depends on **RcppOctave** which has meanwhile been removed from CRAN. Implementing the complete code of PACE in `R` would be useful and lead to the platform-independent integration into **funcy**. We leave this for future work.

References

Alexanderian A (2013). "A Brief Note on the Karhunen-Loève Expansion." *Technical report*, The University of Texas at Austin. URL http://users.ices.utexas.edu/~alen/articles/KL.pdf.

Antoniadis A, Brossat X, Cugliari J, Poggi JM (2013). "Clustering Functional Data Using Wavelets." *International Journal of Wavelets, Multiresolution and Information Processing*, **11**(01), 1350003. doi:10.1142/s0219691313500033.

Auder B (2012). ***modelcf**: Modeling Physical Computer Codes with Functional Outputs Using*

Clustering and Dimensionality Reduction. R package version 2.1.1, URL https://CRAN.R-project.org/package=modelcf.

Bilmes J (1998). "A Gentle Tutorial of the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models." *Technical report*, Georgia Institute of Technology. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.613.

Bouveyron C, Jacques J (2011). "Model-Based Clustering of Time Series in Group-Specific Functional Subspaces." *Advances in Data Analysis and Classification*, **5**(4), 281–300. doi:10.1007/s11634-011-0095-6.

Bowman A, Azzalini A (2014). ***sm**: Nonparametric Smoothing Methods*. University of Glasgow, UK and Università di Padova, Italia. R package version 2.2-5.4, URL https://CRAN.R-project.org/package=sm.

Charles B, Julien J (2014). ***funHDDC**: Model-Based CLustering in Group-Specific Functional Subspaces*. R package version 1.0, URL https://CRAN.R-project.org/package=funHDDC.

Chiou JM, Li PL (2007). "Functional Clustering and Identifying Substructures of Longitudinal Data." *Journal of the Royal Statistical Society B*, **69**(4), 679–699. doi:10.1111/j.1467-9868.2007.00605.x.

Febrero-Bande M, de la Fuente MO (2012). "Statistical Computing in Functional Data Analysis: The R Package **fda.usc**." *Journal of Statistical Software*, **51**(4), 1–28. doi:10.18637/jss.v051.i04.

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

Giacofci M, Lambert-Lacroix S, Marot G, Picard F (2013). "Wavelet-Based Clustering for Mixed-Effects Functional Models in High Dimension." *Biometrics*, **69**(1), 31–40. doi:10.1111/j.1541-0420.2012.01828.x.

Hoerl A, Kennard R (1970). "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics*, **12**(1), 55–67. doi:10.2307/1267351.

Hubert L, Arabie P (1985). "Comparing Partitions." *Journal of Classification*, **2**(1), 193–218. doi:10.1007/bf01908075.

Jacques J, Preda C (2013). "Funclust: A Curves Clustering Method Using Functional Random Variables Density Approximation." *Neurocomputing*, **112**, 164–171. doi:10.1016/j.neucom.2012.11.042.

James GM, Sugar CA (2003). "Clustering for Sparsely Sampled Functional Data." *Journal of the American Statistical Association*, **98**(462), 397–408. doi:10.1198/016214503000189.

Jiang H, Serban N (2012). "Clustering Random Curves Under Spatial Interdependence with Application to Service Accessibility." *Technometrics*, **54**(2), 108–119. doi:10.1080/00401706.2012.657106.

Leisch F (2006). "A Toolbox for K-Centroids Cluster Analysis." *Computational Statistics & Data Analysis*, **51**(2), 526–544. doi:10.1016/j.csda.2005.10.006.

Leisch F (2010). "Neighborhood Graphs, Stripes and Shadow Plots for Cluster Visualization." *Statistics and Computing*, **20**(4), 457–469. doi:10.1007/s11222-009-9137-8.

Peng J, Müller HG (2008). "Distance-Based Clustering of Sparsely Observed Stochastic Processes, with Applications to Online Auctions." *The Annals of Applied Statistics*, **2**(3), 1056–1077. doi:10.1214/08-aoas172.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Ramsay J, Wickham H, Graves S, Hooker G (2014). ***fda**: Functional Data Analysis*. R package version 2.4.4, URL https://CRAN.R-project.org/package=fda.

Ray S, Mallick B (2006). "Functional Clustering by Bayesian Wavelet Methods." *Journal of the Royal Statistical Society B*, **68**(2), 305–332. doi:10.1111/j.1467-9868.2006.00545.x.

Reynolds AP, Richards G, de la Iglesia B, Rayward-Smith VJ (2006). "Clustering Rules: A Comparison of Partitioning and Hierarchical Clustering Algorithms." *Journal of Mathematical Modelling and Algorithms*, **5**(4), 475–504. doi:10.1007/s10852-005-9022-1.

Rice J, Wu C (2001). "Nonparametric Mixed Effects Models for Unequally Sampled Noisy Curves." *Biometrics*, **1**(57), 253–259. doi:10.1111/j.0006-341x.2001.00253.x.

Robinson GK (1991). "That BLUP Is a Good Thing: The Estimation of Random Effects." *Statistical Science*, **6**(1), 15–32. doi:10.1214/ss/1177011926.

Serban N, Wasserman L (2005). "CATS: Clustering After Transformation and Smoothing." *Journal of the American Statistical Association*, **100**(471), 990–999. doi:10.1198/016214504000001574.

Soueidatt M (2014). ***Funclustering**: A Package for Functional Data Clustering*. R package version 1.0.1, URL https://CRAN.R-project.org/package=Funclustering.

The MathWorks Inc (2014). *MATLAB – The Language of Technical Computing, Version R2014b*. Natick. URL http://www.mathworks.com/products/matlab/.

Yao F, Müller HG, Wang JL (2005). "Functional Data Analysis for Sparse Longitudinal Data." *Journal of the American Statistical Association*, **100**(470), 577–590. doi:10.1198/016214504000001745.

Yassouridis C (2018). ***funcy**: Functional Clustering Algorithms*. R package version 1.0.0, URL https://CRAN.R-project.org/package=funcy.

Affiliation:

Christina Yassouridis
Institute of Applied Statistics and Computing
Department of Landscape, Spatial and Infrastructural Sciences
University of Natural Resources and Life Sciences
1190 Vienna, Austria
E-mail: chris.yassou@gmail.com

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
July 2018, Volume 85, Issue 9 *Submitted:* 2015-10-13
doi:10.18637/jss.v085.i09 *Accepted:* 2016-08-07