University of Seville

Faculty of Mathematics

Final Degree Project

Mathematics Degree

Classification of functional data. A mathematical optimization approach.

Belén del Rocío García Camino

Supervised by

Emilio Carrizosa Priego

Statistics and Operational Research Department

Abstract

Functional data appear more and more in modern societies, in which many processes are monitored in real time. Classification (both unsupervised and supervised) problems with such data are a challenging problem. In this work we describe some techniques for classification of functional data using Mathematical Optimization.

Resumen

Cada vez es más frecuente el uso de datos funcionales en las sociedades modernas, en las que muchos de los procesos son monitorizados en tiempo real. Abordar el problema de clasificación (tanto no supervisada como supervisada) con este tipo de datos es un desafío. En este trabajo se describen algunas técnicas de clasificación de datos funcionales mediante optimización matemática.

Contents

List of Figures

Chapter 1

Introduction

This work analyses classification procedures for functional data. To do that, we first review (Chapter 2) some basic ideas of Functional Data, then we address in Chapter 3 some basic classification procedures for such data, and then we describe in Chapter 4 the `R` routines enabling us to perform classification. The work ends with Chapter 5, in which we illustrate the approach with two real datasets `growth` and `CanadianWeather` (`fda` package).

Chapter 2

Functional Data Introduction

With the advancement of modern technology, more and more data are being recorded continuously during a time interval or intermittently at several discrete time points. These are both examples of functional data, which have become a commonly encountered type of data. A functional data or a time series is defined as a set of quantitative observations arranged in chronological order. Functional data analysis (FDA) encompasses the statistical methodology for such data. See [14] for further details.

The simplest dataset encountered in FDA is a sample of this type

$$X_n(t_{j,n}) \in \mathbb{R}, \quad t_{j,n} \in [T_1, T_2], \quad n = 1, 2, \dots N, \quad j = 1, \dots J_n. \tag{2.1}$$

By this we mean that N curves are observed in a common interval $[T_1, T_2]$. The values of the curves may be not known at all points $t \in [T_1, T_2]$, they are available only at some specific points $t_{j,n}$, which can be different for different curves X_n, therefore, j takes values in $1, \dots, J_n$. A fundamental idea of FDA is that the objects we wish to study are smooth curves, like

$$\{X_n(t) : t \in [T_1, T_2], n = 1, 2, \dots N\}, \tag{2.2}$$

for which the values $X_n(t)$ exist at any point t, but are observed only at selected points $t_{j,n}$. Instead of considering functional data as a continuous parameter t, we may consider that time is a discrete variable. Details are found in [8].

2.1 Where do the data live

Most theoretical developments require the assumption that the sample space $\mathbf{X}$ is a real separable Banach space whose norm is denoted by $\|\cdot\|$. Separability ensures that a linear combination of random elements valued at $\mathbf{X}$ is again a random element. Very often a structure of (separable) Hilbert space, with associated inner product $\langle\cdot,\cdot\rangle$ is needed for $\mathbf{X}$. Two standard choices for $\mathbf{X}$ are $\mathcal{C}[a,b]$, the Banach space of real continuous functions $X : [a,b] \to \mathbb{R}$ endowed with the supremum norm $\|X\| = \max_t |X(t)|$, and the Hilbert space $L^2[a,b]$ of square-integrable real functions on $[a,b]$ endowed with the usual inner product $\langle X,Y\rangle = \int_a^b X(t)Y(t)\,dt$.

2.2 Processing the data: basis representation and smoothing

Very often "raw data" $(X(t_1),\dots,X(t_N))$ require a preliminary treatment before applying FDA techniques. This is motivated in terms of dimension reduction in order to remove the noise present in the data measurements. Basis representation is a very usual way to transform the data. Assume that $X \in L^2[a,b]$, and $\{e_k(t)\}$ a orthonomal basis of that space. We may think of fitting a function $\tilde{X}$ from the raw data in the following way: we fix a number J of basis functions, typically smaller than N, and define

$$\tilde{X}(t) = \sum_{j=1}^{J} c_j e_j(t)$$

where the c_j are the Fourier coefficients, chosen in order to minimize

$$\sum_{k=1}^{N}\left(X(t_k) - \sum_{j=1}^{J} c_j e_j(t_k)\right)^2$$

Then, this representation process can be summarized in terms of two transformations,

$$(X(t_1),\dots,X(t_N)) \mapsto (c1,\dots c_J) \mapsto (\tilde{X}(t_1),\dots,\tilde{X}(t_N))$$

Therefore, this procedure provides both a more compact representation of the data (as J is typically much smaller than N) and a denoising process, since $\tilde{X}$ can be considered as a smoothed version of the original data X.

Note that an important advantage of such regularization processes is the possibility of using the first or second order derivatives of the original data. This is extremely useful in practice, as sometimes the relevant information is included in the derivatives rather than in the data themselves. Details are found in [4].

2.3 Cubic Spline Interpolation

Cubic spline can be used to achieving smooth curves. The fundamental idea behind cubic spline interpolation is to draw smooth and simple enough curves (piecewise third-degree polynomials) through a number of points. For each $n = 1, 2, \dots N$, the essential idea is to fit a function of the form

$$S_n(t) = \begin{cases} s_n^1(t) & if \quad t \in [t_{1,n}, t_{2,n}] \\ s_n^2(t) & if \quad t \in [t_{2,n}, t_{3,n}] \\ & \vdots \\ s_n^{J_{n-1}}(t) & if \quad t \in [t_{J_{n-1},n}, t_{J_n,n}] \end{cases} \tag{2.3}$$

where J_n is the number of observations of the curve x_n and s_n^j for $j = 1, \dots, J_{n-1}$ is a third degree polynomial defined by

$$s_n^j(t) = a_n^j(t - t_{j,n})^3 + b_n^j(t - t_{j,n})^2 + c_n^j(t - t_{j,n}) + d_n^j. \tag{2.4}$$

Our spline $S_n(t)$ will need to interpolate all data points $(t_{j,n}, x_n(t_{j,n}))$ for each $n = 1, 2, \dots N$ so

$$\begin{aligned} x_n(t_{j,n}) &= S_n(t_{j,n}) \\ &= s_n^j(t_{j,n}) \\ &= a_n^j(t_{j,n} - t_{j,n})^3 + b_n^j(t_{j,n} - t_{j,n})^2 + c_n^j(t_{j,n} - t_{j,n}) + d_n^j \\ &= d_n^j. \end{aligned} \tag{2.5}$$

Furthermore, the functions $S_n(t)$ and their derivatives $S'_n(t)$ and $S''_n(t)$ will be continuous on the interval $[t_{1,n}, t_{J_n,n}]$

$$\begin{aligned} s_n^j(t_{j+1,n}) &= s_n^{j+1}(t_{j+1,n}) \quad \text{for } j = 1, \dots J_{n-2} \\ s_n^{j'}(t_{j+1,n}) &= s_n^{j+1'}(t_{j+1,n}) \quad \text{for } j = 1, \dots J_{n-2} \\ s_n^{j''}(t_{j+1,n}) &= s_n^{j+1''}(t_{j+1,n}) \quad \text{for } j = 1, \dots J_{n-2}, \end{aligned} \tag{2.6}$$

where the equations of the derivatives

$$\begin{aligned} s_n^{j'}(t) =& \quad 3a_n^j(t-t_{j,n})^2 + 2b_n^j(t-t_{j,n}) + c_n^j \\ s_n^{j''}(t) =& \quad 6a_n^j(t-t_{j,n}) + 2b_n^j. \end{aligned} \tag{2.7}$$

2.4 Sample mean and variance

We now assume that the raw data have been converted to functional objects of the form (2.2) using the method described above in Section 2.3. The simplest statistics are the pointwise mean and the pointwise standard deviation.

Definition 2.1
Given n times series $\{X_1, \dots, X_N\}$, the pointwise mean or the center time series is given by

$$\overline{X}(t) = \frac{1}{N}\sum_{i=1}^{N} X_i(t). \tag{2.8}$$

■

Definition 2.2
Given n times series $\{X_1, \dots, X_N\}$, the pointwise standard deviation is given by

$$\sigma(t) = \left\{ \frac{1}{N-1}\sum_{i=1}^{N}(X_i(t) - \overline{X}(t))^2 \right\}^{1/2}. \tag{2.9}$$

■

The pointwise sample standard deviation gives us an idea about the typical variability of the curves at any point t, but it does not give information on how the values of the curves at the point t relate to those at the point s, for that is defined the sample covariance function as follows.

Definition 2.3

$$Cov(t,s) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i(t) - \overline{X}(t))(X_i(s) - \overline{X}(s)) \tag{2.10}$$

■

The interpretation of the values of Cov(t, s) is the same as for the usual variance-covariance matrix. For example, large values indicate that $X_i(t)$ and $X_i(s)$ tend to be simultaneously above or below the average values at these points.

2.5 Euclidean Distance and Dynamic Time Warping (DTW) Distance

Definition 2.4
The squared Euclidean distance between two time series X_1 and X_2, where each functional data is observed in the same m discrete time instants $t_1, \dots, t_m$ is given by [3]:

$$d^2(X_1, X_2) = \sum_{j=1}^{m} (X_1(t_j) - X_2(t_j))^2 \tag{2.11}$$

■

Rigid distances, such as the Euclidean distance or the Manhattan distance, present difficulties [1], such as their inability to naturally measure the distance between similar series, but which have some displacement in time.

Example 2.5 Using the example of the dataset `growth` of the package `fda.usc` [6], all functional data are measured at 31 discrete time points between the ages of 1 and 18. Given new functional data X, our objective is to predict to which group it will belong, if "girls" or "boys", classifying it in the closet group in terms of a metric. If the heights of this new data are recorded at the same 31 time points, the Euclidean distance is usually used to measure the distances between X and the rest of the dataset. Unfortunately, in most cases, X is observed in different discrete points, moreover outside of the range between 1 and 18 years.

■

The difficulties above can be solved using the DTW distance, but it carries a higher computational cost. One of the differences between the DTW distance and the Euclidean distance is that certain elements of a series can be matched with one of the other series.

The objective of the DTW is to contrast two time series $X := (x_1, x_2, \dots, x_N)$ of length $N \in \mathbb{N}$ and $Y := (y_1, y_2, \dots, y_M)$ of length $M \in \mathbb{N}$. In this technique, each point of the first time series is compared with any arbitrary point from the second time series. [7]

Fixed a sample space $\mathbf{X}$, to compare $x_n, y_m \in \mathbf{X}$ for $n \in \{1, \dots, N\}$ and $m \in \{1, \dots, M\}$, one needs a *local cost measure*, sometimes also referred to as *local distance measure*, which is defined to be a function

$$c : \mathbf{X} \times \mathbf{X} \to \mathbb{R}_{\geq 0}. \tag{2.12}$$

Typically, $c(x_n, y_m)$ is small (low cost) if x_n and y_m are similar to each other, and otherwise $c(x_n, y_m)$ is large (high cost).

Taking into account the standard choices for $\mathbf{X}$ discussed above in Section 2.1, we use the Euclidean distance as local cost measure c. Manhattan distance can also be used.

Evaluating the local cost measure for each pair of elements of the functional data X and Y, one obtains the *cost matrix* $C \in \mathbb{R}^{N\times M}$, defined by $C(n, m) := c(x_n, y_m)$. The goal is then to find an alignment between X and Y with minimal overall cost. To formalize the notion of alignment, we define *warping path.* [10]

Definition 2.6
An (N, M)-warping path is a contiguous set of cost matrix elements $p = (p_1, \ldots, p_L)$ with $p_\ell = (n_\ell, m_\ell) \in \{1, \ldots, N\} \times \{1, \ldots, M\}$ for $\ell \in \{1, \ldots, L\}$, that defines a mapping between X and Y, satisfying the following three conditions.

1. *Boundary condition: $p_1 = (1, 1)$ and $p_L = (N, M)$.*

2. *Monotonicity condition: $n_1 \leq n_2 \leq \cdots \leq n_L$ and $m_1 \leq m_2 \leq \cdots \leq m_L$.*

3. *Step size condition: $p_{\ell+1} - p_\ell \in \{(1, 0), (0, 1), (1, 1)\}$ for $\ell \in \{1, \ldots, L - 1\}$.*

$\blacksquare$

Observation 2.7 The step size condition implies the monotonicity condition.

A (N, M)-warping path $p = (p_1, \ldots, p_L)$ defines an alignment between two sequences $X := (x_1, x_2, \ldots, x_N)$ and $Y := (y_1, y_2, \ldots, y_M)$ by assigning the element x_{n_ℓ} of X to the element y_{m_ℓ} of Y. The boundary condition enforces that the first elements of X and Y as well as the last elements are aligned to each other. The monotonicity condition forces the points in the warping path to be monotonically spaced in time: if an element in X precedes a second one this should also hold for the corresponding elements in Y , and vice versa. The step size condition expresses a kind of continuity condition: no element in X and Y can be omitted and there are no replications in the alignment i.e., all index pairs contained in a warping path p are pairwise distinct. Also, this condition restricts the allowable steps in the warping path to adjacent cells (including diagonally adjacent cells).

Example 2.8 Let p an $(4,3)$-warping path $p = (p_1, p_2, p_3, p_4)$ with $p_\ell = (n_\ell, m_\ell)$ for $\ell \in \{1,2,3,4\}$ and $p_1 = (1,1), p_2 = (2,2), p_3 = (3,3), p_4 = (4,3)$. Then

$$\begin{aligned} n_1 &= 1, & m_1 &= 1 \\ n_2 &= 2, & m_2 &= 2 \\ n_3 &= 3, & m_3 &= 3 \\ n_4 &= 4, & m_4 &= 3 \end{aligned} \tag{2.13}$$

The warping path p defines an alignment between two sequences $X = (x_1, x_2, x_3, x_4)$ and $Y = (y_1, y_2)$ by assigning x_{n_ℓ} of X to the element y_{m_ℓ} for $\ell \in \{1,2,3,4\}$. The assignment is

$$\begin{aligned} x_{n_1} = x_1 &\quad \text{to} \quad y_{m_1} = y_1 \\ x_{n_2} = x_2 &\quad \text{to} \quad y_{m_2} = y_2 \\ x_{n_3} = x_3 &\quad \text{to} \quad y_{m_3} = y_3 \\ x_{n_4} = x_4 &\quad \text{to} \quad y_{m_4} = y_3 \end{aligned} \tag{2.14}$$

The warping path above is shows in the next figure

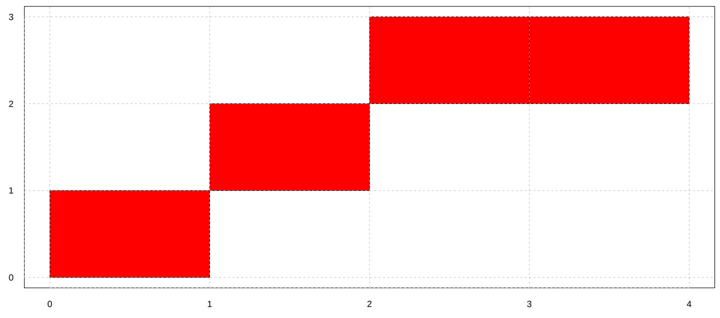
```
dev.new()
plot(NA, xlim=c(0, 4), ylim=c(0, 3), xlab='', ylab='',xaxt="na",yaxt="na")
rect(0, 0, 1, 1, col='red')
rect(1, 1, 2, 2, col="red")
rect(2, 2, 3, 3, col="red")
rect(3, 2, 4, 3, col="red")

# Vertical grid
axis(1,
     at = c(0:4),
     tck = 1, lty = 2, col = "gray")

# Horizontal grid
axis(2,
     at = c(0:3),
     tck = 1, lty = 2, col = "gray", las=1)
```

Definition 2.9
The total cost $c_p(X,Y)$ of a warping path between X and Y with respect to the local cost measure c is defined as

$$c_p(X,Y) := \sum_{\ell=1}^{L} c(x_{n_\ell}, y_{m_\ell}). \tag{2.15}$$



■

There are exponentially many warping paths that satisfy the above conditions. However, we are only interested in the path that minimizes the total cost.

Definition 2.10
An optimal warping path between X and Y is a warping path p^* having a minimal total cost among all possible warping paths.

■

Definition 2.11
The DTW distance $\mathrm{DTW}(X,Y)$ *between X and Y is then defined as the total cost of p^*:*

$$\begin{aligned}\mathrm{DTW}(X,Y) &:= c_{p^*}(X,Y) \\ &= \min\{c_p(X,Y) \mid p \textit{ is an } (N,M)\textit{-warping path}\}\end{aligned} \tag{2.16}$$

■

As a result, time series with similar patterns occurring in different time periods are considered similar.

To determine an optimal path p^*, one could test every possible warping path between X and Y. The complexity of this procedure is exponential in the lengths N and M. We will now introduce an $O(NM)$ algorithm that is based on dynamic programming. To this end, we define the sequences $X(1:n) := (x_1, x_2, \ldots, x_n)$ for $n \in \{1, \ldots, N\}$ and $Y(1:m) := (y_1, y_2, \ldots, y_m)$ for $m \in \{1, \ldots, M\}$ and set

$$D(n,m) = \mathrm{DTW}(X(1:n), Y(1:m)). \tag{2.17}$$

The values $D(n,m)$ define an $N \times M$ matrix D, which is also referred to as the *accumulated cost matrix*. One has $X(1:N)$ is equal to X and $Y(1:M)$ is equal to Y, so $D(N,M) = \mathrm{DTW}(X(1:N), Y(1:M)) = \mathrm{DTW}(X,Y)$. The next theorem shows how D can be computed efficiently.

Theorem 2.12 *The accumulated cost matrix D satisfies the following identities:*

$$\begin{aligned} D(n,1) &= \sum_{k=1}^{n} c(x_k, y_1) for n \in \{1,\dots,N\}, \\ D(1,m) &= \sum_{k=1}^{m} c(x_1, y_k) for m \in \{1,\dots,M\}, and \end{aligned} \tag{2.18}$$

$$D(n,m) = min\{D(n-1,m-1) + D(n-1,m) + D(n,m-1)\} + c(x_n, y_m) \tag{2.19}$$

for $1 < n \leq N$ and $1 < m \leq M$. In particular $\mathrm{DTW}(X,Y) = D(N,M)$ can be computed with $O(NM)$ operations.

Proof. Let $m = 1$ and $n \in \{1,\dots,N\}$. Then

$$\begin{aligned} D(n,1) &= \mathrm{DTW}(X(1:n), Y(1:1)) \\ &= \mathrm{DTW}(X(1:n), y_1) \\ &= c_{p^*}(X(1:n), y_1) \\ &= \sum_{k=1}^{n} c(x_k, y_1) \end{aligned} \tag{2.20}$$

The first equality is given by the definition of an accumulated cost matrix element 2.19. The second is by the definition of sequence $Y(1:m)$ with $m = 1$ 2.5. Then we use the definition of the DTW distance 2.11. There is only one possible warping path between $X(1:n)$ and $Y(1:1) = y_1$ given by $p^* = ((1,1),(2,1),\dots,(n,1))$ that satisfies the three conditions of warping path definition 2.6, so it is the minimum of all the possible warping paths. Finally, we obtain the total cost of this warping path 2.9 as $c(x_1,y_1) + c(x_2,y_1) + \cdots + c(x_n,y_1) = \sum_{k=1}^{n} c(x_k,y_1)$.

Similarly, one obtains the formula for $D(1,m) = D(1,m) = \sum_{k=1}^{m} c(x_1,y_k)$ for $m \in \{1,\dots,M\}$.

Now, let $n > 1$ and $m > 1$ and let $q = (q_1,\dots,q_{L-1},q_L)$ be an optimal path for $X(1:n)$ and $Y(1:m)$. Then the boundary condition implies $q_L = (n,m)$. Setting $(a,b) := q_{L-1}$, the step size condition implies $(a,b) \in \{(n-1,m-1),(n-1,m),(n,m-1)\}$. Furthermore, it follows that $(q_1,\dots,q_{L-1})$ must be an optimal warping path for $X(1:a)$ and $Y(1:b)$ (otherwise, q would not be optimal for

$X(1:n)$ and $Y(1:m)$). Since

$$\begin{aligned} D(n,m) = & \ \mathrm{DTW}(X(1:n),Y(1:m)) \\ = & \ c_q(X(1:n),Y(1:m)) \\ = & \ c_{(q_1,\ldots,q_{L-1})}(X(1:a),Y(1:b)) + c(x_n,y_m) \\ = & \ min\{D(n,m-1),D(n-1,m),D(n-1,m-1)\} + c(x_n,y_m) \end{aligned} \tag{2.21}$$

The DTW distance between the sequences $X(1:n)$ and $Y(1:m)$ is the total cost of the optimal warping path q. This cost can be broken down into two addends: the total cost of the optimal warping path for $X(1:a)$ and $Y(1:b)$, where $a=n$ or $a=n-1$ and $b=m$ or $b=m-1$ attend the step size condition, and the local cost, for example, the Euclidean distance, between x_n and y_m. The election of a and b shall determine which of the next set $\{D(n,m-1),D(n-1,m),D(n-1,m-1)\}$ corresponds whit the DTW distance, the one who gives the minimum.■

Theorem 2.12 facilitates a recursive computation of the matrix D. The initialization can be simplified by extending the matrix D with an additional row and column and formally setting $D(n,0) := \infty$ for $n \in \{1,\ldots,N\}$, $D(0,m) := \infty$ for $m \in \{1,\ldots,M\}$ and $D(0,0) = 0$. Then the recursion of (2.19) holds for $n \in \{1,\ldots,N\}$ and $m \in \{1,\ldots,M\}$. The time and space complexity of this method is $O(NM)$. The following algorithm compute an optimal warping path p^*.

Algorithm: Optimal Warping Path

Input: Accumulated cost matrix D.

Output: Optimal warping path p^*.

Procedure: The optimal path $p^* = (p_1,\ldots,p_L)$ is computed in reverse order of the indices starting with $p_L = (N,M)$. Suppose $p_\ell = (n,m)$ has been computed. In case (n,m)=(1,1), we are finished. Otherwise,

$$p_{\ell-1} := \begin{cases} (1,m-1) & \text{if} n=1 \\ (n-1,1) & \text{if} m=1 \\ \arg\min\{D(n,m-1),D(n-1,m),D(n-1,m-1)\} & \text{otherwise} \end{cases} \tag{2.22}$$

where we take the lexicographically smallest pair in case $\arg\min\{D(n,m-1),D(n-1,m),D(n-1,m-1)\}$ is not unique.

Observation 2.13 The Euclidean distance between two sequences can be seen as a special case of the DTW distance, where each element of the optimal warping path is constrained to $p_\ell = (n_\ell, m_\ell)$ with $n_\ell = m_\ell = \ell$. Note that it is only defined in the special case where the two sequences have the same length $N = M$. [5]

Chapter 3

Introduction to the Functional Data Classification

Our aim here is to summarize the main ideas and techniques used so far in the classification of functional data.

3.1 Supervised and unsupervised functional classification

In statistics, the word *classification* also has the same usual double meaning as in the ordinary language, where this term stands for both "to assign (an element) to a particular class or category" and for "arrange (a group of elements) in classes according to shared characteristics". The first meaning corresponds to the statistical methodology called *supervised classification* or *discriminant analysis*. The second one would fit better with *clustering methodology*, which roughly corresponds to the *unsupervised classification* theory. Here, the term "supervised" in the first problem refers to the fact that there is a "training"(sample) dataset of elements which are assumed to be well-classified. Then the problem is to classify the new incoming elements. In the unsupervised class, no such help is available, the problem is just to group the data into "clusters" of mutually alike elements. The reader is referred to [2] for more details

3.2 Unsupervised Classification

The purpose of unsupervised classification techniques is to partition a (usually large) data sample $\{X_1, \dots, X_N\}$ into a number K of clusters or groups. The members of the same cluster are similar to each other according to certain characteristics. Some well-known algorithms for grouping a given dataset are based on the mutual distances between the data. So, they can be adapted to the case of functional data, provided that a suitable distance is defined.

The number of groups K must be given in advance, but most grouping procedures include some guidelines for the choice of K. Unsupervised classification is typically used when a large amount of data is available and some internal structure of data groups is suspected.

3.2.1 K-means Clustering

K-means clustering is a method for finding clusters and cluster centers in a set of unlabeled data. Once the desired number of cluster centers was elected, say K, and the K-means procedure iteratively moves the centers to minimize the total within variance [13]. Given an initial set of K centers, the K-means algorithm alternates the two steps:

- for each center X_k for $k \in \{1, \dots, K\}$ we identify the subset of training functional data (its cluster) that is closer to it than any other center.
- the mean 2.4 for the functional data in each cluster are computed, and this mean becomes the new center for that cluster.

These two steps are iterated until convergence. Typically, the initial centers are K randomly chosen observations from the training data.

3.2.2 Gaussian Mixtures

Each cluster is described in terms of a Gaussian density, which has a centroid and a covariance matrix. The two steps of the alternating EM algorithm are very similar to the two steps in K-means:

- In the E-step, each observation is assigned a weight for each cluster, based on the likelihood of each of the corresponding Gaussian's. Observations close to the center of a cluster will most likely get weight 1 for that cluster and weight 0 for every other cluster. Observations half-way between two clusters divide their weights accordingly.

- In the M-step, each observation contributes to the weighted means (and covariance) for every cluster.

3.3 Supervised classification

This methodology applies when K populations are given in advance. The available data consist of a "training sample" (X_n, Y_n) where X_n for $i \in \{1, \dots, N\}$ are functional data and $Y_n = k$ if the n-th individual belongs to the population P_k. The final aim is to classify a newly appearing observation X into one of the populations P_k, i.e., we want to predict the corresponding value Y using the information provided by the training sample.

3.3.1 k-Nearest Neighbors Method

Assume that $\mathbf{X}$ is a metric space. To classify a data X, look at the k training data closest to X (in the metric of $\mathbf{X}$) and assign X to P_j when the majority of these k data belong to P_j, in the case of our training sample consist of K clusters $P_1, \dots, P_K$. For the latter, an appropriate functional distance must be chosen. This is equivalent to specifying the feature metric space $\mathbf{X}$ where the functional data are supposed to take values. See Section 2.1.

3.3.2 Prototype Methods

Throughout this section, our training data consists of the N pairs $(X_1, Y_1) \dots,$ (X_N, Y_N) where Y_n for $n \in \{1, \dots, N\}$ is a class label taking values in $1, 2, \dots, K$. Prototype methods represent the training data by a set of functional data in the feature space $\mathbb{X}$. These prototypes are typically not examples from the dataset.

Each prototype has an associated class label, and the classification of a query point X is made into the class of the closest prototype. Closeness is usually defined by the Euclidean distance in the feature space. The Euclidean distance usually requires two time series to have the same length. Otherwise, the DTW distance is used, as we discuss in Section 2.5. The main challenge is to figure out how many prototypes to use and where to put them.

3.4 K-fold cross-validation

Ideally, if we had enough data, we would set aside a validation set and use it to assess the performance of our prediction model. Since data are often scarce, this is not usually possible. To avoid the problem, K-fold cross-validation uses part of the available data to fit the model, and a different part to test it. We split the data into K roughly equal-sized parts.

For the k-th part $k \in \{1, \dots, K\}$, we fit the model to the other $K - 1$ parts of the data and calculate the prediction error of the fitted model when predicting the k-th part of the data. We do this for $k = 1, 2, ..., K$ and combine the K estimates of the prediction error. The way we combine the K estimates is doing the mean.

Typical choices for K are 5 or 10. If we have a dataset $\{X_1, \dots, X_N\}$, the case $K = N$ is known as leave-one-out cross-validation: for each functional data X_n, we fit the model with the other $N - 1$ time series, and calculate the prediction error of the fitted model when predicting X_n. We repite this for each function in our dataset and estimate the prediction error as the mean of the N estimators.

Observation 3.1 It is important to randomly select the sample because it is possible that the records follow a pattern linked with their position in the dataset. For example, `growth` dataset of `fda.usc` package [6] consists of heights measured in boys and girls at 31 discrete time points between the ages of 1 and 18. Then, we could choose "boys" curves to test the model and "girls" ones to fit it. In this case, the error will be considerably greater.

In the folds of different models, there is a common part; this is the reason that the K estimators of the error will not be independent. However, the so-obtained estimator will be more precise in the sense of variance.

Chapter 4

Automatic procedures by R

4.1 Introduction to `splines2`

The `R` package `splines2` provides functions to construct basis matrices of natural cubic splines, along with their integrals and derivatives of a given order.

The function `naturalSpline()` returns nonnegative basis functions (within the boundary) for natural cubic splines. When `integral = TRUE`, `naturalSpline()` returns the integral of each natural spline basis. To obtain the derivatives of the spline basis functions, we may specify the argument `derivs = k`, `k` being the order of the derivative, or the `deriv()` method.

4.2 Conversion of raw data or other functional data classes into `fdata` class

The function `fdata()` of the `fda.usc` package [6] creates a functional data object of class fdata from `matrix`, `data.frame`, `numeric`, `integer`, `fd`, `fds`, `fts` or `sfts` class data.

```
fdata(mdata,
      argvals = NULL,
      rangeval = NULL,
      names = NULL,
      fdata2d = FALSE)
```

Arguments

- `mdata` is a matrix of set cases with dimension $(n \times m)$, where n is the number of curves and m are the points observed in each curve.
- `argvals` by default $1 : m$, is a vector of the points observed.
- `rangeval` range of discretization points, by default: `range(argvals)`. `names` list with tree components:
  - `main` an overall title.
  - `xlab` title for `x` axis.
  - `ylab` title for `y` axis.
- `fdata2d` = `TRUE` if the functional data is observed in at least two grids. In this case, `argvals` is a list of vectors. This argument is `FALSE` by default.

The result is a `fda` class object with:

- `data` matrix of the set of cases of the argument `mdata`.
- `argvals` the discretizations points.
- `rangevals` range of the discretizations points.
- `names` list `names` of the argument.

4.3 Central and dispersion measures for functional data

```
func.mean.formula(formula, data = NULL, ...)

func.mean(x)

func.var(fdataobj)

func.trim.FM(fdataobj, ...)
func.trim.mode(fdataobj, ...)

func.trim.RP(fdataobj, ...)

func.med.FM(fdataobj, ...)

func.med.mode(fdataobj, ...)

func.med.RP(fdataobj, ...)

func.trimvar.FM(fdataobj, ...)

func.trimvar.mode(fdataobj, ...)

func.trimvar.RP(fdataobj, ...)

func.trimvar.RPD(fdataobj, ...)
```

Arguments

- `x` `fdata` or `ldata` class object.
- `fdata` class object.
- `formula` a formula, such as *y group*, where *y* is a `fdata` object to be split into groups according to the grouping variable (usually a factor).
- `data` `ldata` class object. It is a list containing the variables in the formula. The item called "*df*" is a data frame with the grouping variable. The item called "*fdata*" is a `fdata` object.

Value

- `func.mean.formula(formula, data = NULL, ...)` Returns a `fdata` object containing the mean curves for the groups.
- `func.mean(x)` Gives mean curve.
- `func.var(fdataobj)` Gives variance curve.

- `func.trim.FM(fdataobj, ...)` Returns the average from the `(1-trim)%` deepest curves following FM criteria.
- `func.trim.mode(fdataobj, ...)` Returns the average from the `(1-trim)%` deepest curves following mode criteria.
- `func.trim.RP(fdataobj, ...)` Returns the average from the `(1-trim)%` deepest curves following RP criteria.
- `func.med.FM(fdataobj, ...)` Returns the deepest curve following FM criteria.
- `func.med.mode(fdataobj, ...)` Returns the deepest curve following mode criteria.
- `func.med.RP(fdataobj, ...)` Returns the deepest curve following RP criteria.
- `func.trimvar.FM(fdataobj, ...)` Returns the marginal variance from the deepest curves followinng FM criteria.
- `func.trimvar.mode(fdataobj, ...)` Returns the marginal variance from the deepest curves followinng mode criteria.
- `func.trimvar.RP(fdataobj, ...)` Returns the marginal variance from the deepest curves followinng RP criteria.

4.4 Derivatives of functional data

```
fdata.deriv(
  fdataobj,
  nderiv = 1,
  method = "bspline",
  class.out = "fdata",
  nbasis = NULL,
  ...
)
```

Arguments

- `fdataobj` `fdata` class object.
- `nderiv` Order of derivation.
- `method`
  - if `method` = “*bspline*”, “*exponential*”, “*fourier*”, “*monomial*” or “*polynomial*”, `fdata.deriv` function creates a basis to represent the functional data. First, the functional data are converted to class `fd` using the indicated basis. Finally, the function calculates the derivative of order `nderiv`.
  - if `method` = “*fmm*”, “*periodic*”, “*natural*” or “*monoH.FC*” is used `splinefun` of the `stats` package.
  - if `method` = “*diff*”, raw derivation is applied. It is not recommended to use these methods when the values are not equally spaced.
- `class.out` Class of functional data returned: `fdata` or `fd` class.
- `nbasis` Number of basis for `fdataobj`$data when `method` = “*bspline*”, “*exponential*”, “*fourier*”, “*monomial*” or “*polynomial*”.

`fdata.deriv` returns the derivative of functional data of `fd` class if `class.out` = “*fd*” or `fdata` class if `class.out` = “`fdata`”.

4.5 Euclidean distance matrix computation

The squared Euclidean distance between two time series or functional data observed at m discrete time points is given in Section 2.5 by

$$d_E^2(X_1, X_2) = \sum_{j=1}^{m} (X_1(t_j) - X_2(t_j))^2 \tag{4.1}$$

In this term, the function `metric.dist` of the package `fda.usc` [6] computes the Euclidean distances between the rows of two data matrix by using the specified distance measure.

```
metric.dist(x,y = NULL, method = "euclidean", p = 2, dscale = 1, ...)
```

Arguments

- `x` is a data matrix with dimension $n1 \times m$ i.e., $n1$ functional data measured in m discrete time points.
- `y` is a data matrix with dimension $n2 \times m$ i.e., $n2$ functional data measured in m discrete time points. If `y = NULL` the function `metric.dist` returns a symmetric matrix with the distances between two functional data of `x`.
- `method` in this case, `method` = '*euclidean*". Other rigid distances are available: "*maximum*", "*manhattan*", "*canberra*", "*binary*" or "*minkowski*".

This function returns a distance matrix of dimension $(n1 \times n1)$ if `y = NULL`, $(n1 \times n2)$ otherwise.

4.6 Dynamic time warping distance matrix

The function `metric.DTW` of the package `fda.usc` [6] computes distances time warping for functional data.

```
metric.DTW(
  fdata1,
  fdata2 = NULL,
  p = 2,
  w = min(ncol(fdata1), ncol(fdata2)),
  ...
)
```

Arguments

- `fdata1` Functional data object where `fdata$data` is a data matrix with dimension $n_1 \times m$ i.e., n_1 functional data measured in m discrete time points.
- `fdata2` Functional data object where `fdata$data` is a data matrix with dimension $n_2 \times m$ i.e., n_2 functional data measured in m discrete time points.
- `p` L^p norm, by default it uses `p = 2`.
- `w` Vector of weights with length m, If `w = 1` (by default) approximates the metric L^p by Simpson's rule.

4.7 Functional Classification using K-fold cross-validation

The function `classif.kfold` of the package `fda.usc` [6] computes functional classification using K-fold cross-validation.

```
classif.kfold(
  formula,
  data,
  classif = "classif.glm",
  par.classif,
  kfold = 10,
  ...
)
```

Arguments

- `formula` is an object of class `formula`: a symbolic description of the model to be fitted.
- `data` is a `list` containing the variables in the model.
- `classif` is a character, name of classification method to be used in the fitting model.
- `par.classif` is a `list` of arguments used in the classification method.
- `kfold` is an `integer`, the number of K-fold.

4.8 k-Nearest Neighbors classifier from Functional Data

The function `classif.knn` fits supersvised classification for functional data.

```
classif.knn(
  group,
  fdataobj,
  knn = NULL,
  metric,
  ...
)
```

Arguments

- `group` is a factor vector of length N of the corresponding group for each functional data in `fdataobj`. Following the notation in Section 3.3.1, `group = (`Y_1`,...,`Y_N`)`.
- `fdataobj` is an object of `fdata` class.
- `knn` is a vector of number of nearest neighbors considered.
- `metric` metric function, “metric.lp”, “metric.dist” or “metric.DTW” are available.

4.9 Predicts from a fitted classif object

The function `predict` of the package `fda.usc` [6] classifies a functional data by kernel method using functional data object of class `classif`. It returns the predicted classes using a previously trained model.

```
predict(object, new.fdataobj = NULL, type = "class", ...)
```

Arguments

- `object` Object estimated by: k-nearest neighbors method `classif.knn`, kernel method `classif.kernel`.
- `new.fdataobj` New functional explanatory data of `fdata` class.

- `type` Type of prediction (class or probability of each group membership).

If `type`="class", produces a vector of predictions. If `type`="probs", a list with the following components is returned:

- `group.pred` the vector of predictions.
- `prob.group` the matrix of predicted probability by factor level.

4.10 K-means clustering for functional data

The function `kmeans.fd` of `fda.usc` package [6] performs K-means clustering for functional data. Returns a vector of the indexes of groups assigned `clusters` and a fdata object of curves centers.

```
kmeans.fd(
  fdataobj,
  ncl = 2,
  metric = metric.lp,
  max.iter = 100,
  method = "sample",
  cluster.size = 5,
  draw = TRUE,
  ...
)
```

Arguments

- `fdataobj` fdata class object.
- if `ncl` is an integer, indicating the number of groups to classify, `ncl` initial centers are selected using `kmeans.center.ini` function of the `fda.usc` package [6]. If `ncl` is a vector of integers, indicating the position of the initial centers with `length(ncl)` equal to number of groups. If `ncl` is a fdata class objecct, `ncl` are the initial centers curves with `nrow(ncl)` number of groups.
- `metric` Metric function, by default `metric.lp`.

- `max.iter`. Maximum number of iterations for the detection of centers.
- `method` for selectiong initial centers. If `method`="*sample*" takes a random selection by the `ncl` centers. The `ncl` curves with greater distance are the initial centers. If `method`="*exact*", all combinations of `ncl` centers are calculated (if ¡ 1e+6) of `ncl` centers. The `ncl` curves with greater distance are the initial centers (this method may be too slow).
- `cluster.size` Minimum cluster size (by default is 5). If a cluster has fewer curves, it is eliminated and the process is continued with a less cluster.
- `draw=TRUE`, draw the curves in the color of the centers.

Chapter 5

Experiments

5.1 `growth` dataset

5.1.1 Data description

Berkeley (California) Growth Study data, downloaded from the package `fda` [12] is a list containing the heights of 39 boys and 54 girls between the ages of 1 to 18, and a vector with such ages. This list contains the following components:

- `growth$hgtm` Is a 31 by 39 numeric matrix giving the heights in centimeters of 39 boys at 31 ages between 1 and 18 years.
- `growth$hgtf` Is a 31 by 54 numeric matrix giving the heights in centimeters of 54 girls at 31 ages between 1 and 18 years.
- `growth$age` a numeric vector of length 31 giving the ages at which the heights were measured. Ages are not equally spaced.

First, we upload the `fda` [12] library to use the dataset `growth` and visualize the elements of the list.

```
library(fda)

data(growth)
```

```
growth$hgtm[,1:4]

growth$hgtf[,1:4]

growth$age
```

```
> library(fda)
> data(growth)
> growth$hgtm[,1:4]
     boy01 boy02 boy03 boy04
1     81.3  76.2  76.8  74.1
1.25  84.2  80.4  79.8  78.4
1.5   86.4  83.2  82.6  82.6
1.75  88.9  85.4  84.7  85.4
2     91.4  87.6  86.7  88.1
3    101.1  97.0  94.2  98.6
4    109.5 104.6 100.4 104.4
5    115.8 112.3 107.1 111.0
6    121.9 118.9 112.3 116.3
7    130.0 125.0 118.6 123.2
8    138.2 130.1 124.0 129.9
8.5  141.1 133.0 126.5 133.0
9    144.3 135.4 128.9 136.0
9.5  147.5 137.5 131.2 138.7
10   150.5 139.7 133.4 141.4
10.5 153.4 142.2 135.8 144.0
11   156.2 144.2 138.4 146.4
11.5 159.7 146.2 141.0 148.8
12   163.8 148.1 143.6 151.4
12.5 168.8 149.8 146.8 154.5
13   174.9 151.6 150.8 157.7
13.5 181.2 153.8 155.1 162.2
14   186.3 156.3 159.5 167.4
14.5 189.6 159.2 163.3 172.5
15   191.3 163.3 166.8 176.3
15.5 192.1 167.7 167.8 178.5
16   192.8 171.5 168.8 179.8
16.5 193.2 174.3 169.8 180.7
```

```
17   193.8 176.1 170.9 181.4
17.5 194.3 177.4 171.2 181.6
18   195.1 178.7 171.5 181.8
> growth$hgtf[,1:4]
     girl01 girl02 girl03 girl04
1      76.2   74.6   78.2   77.7
1.25   80.4   78.0   81.8   80.5
1.5    83.3   82.0   85.4   83.3
1.75   85.7   86.9   87.9   87.0
2      87.7   90.0   89.6   90.3
3      96.0   94.9   97.1   98.6
4     103.8  102.1  109.2  106.3
5     110.7  109.2  116.2  113.9
6     116.8  115.6  122.9  120.7
7     122.2  122.7  128.0  125.7
8     127.4  128.2  134.2  131.0
8.5   130.6  131.1  138.0  134.1
9     133.4  134.8  141.5  137.1
9.5   135.9  138.2  145.2  140.2
10    138.6  140.9  148.8  143.0
10.5  142.4  143.4  152.3  145.7
11    146.8  146.1  155.6  148.5
11.5  150.3  149.4  158.2  151.5
12    153.1  152.9  159.6  154.8
12.5  155.0  156.4  160.1  158.4
13    156.2  159.5  160.3  161.2
13.5  157.1  161.6  160.8  163.5
14    157.7  162.6  161.6  165.2
14.5  158.0  163.6  161.8  166.0
15    158.2  165.0  161.7  166.5
15.5  158.4  165.3  161.8  166.9
16    158.6  165.6  161.9  167.2
16.5  158.7  165.9  161.9  167.4
17    158.7  166.1  161.7  167.4
17.5  158.8  166.0  161.9  167.6
18    158.9  166.0  162.2  167.8
> growth$age
 [1]  1.00  1.25  1.50  1.75  2.00  3.00  4.00  5.00  6.00
[10]  7.00  8.00  8.50  9.00  9.50 10.00 10.50 11.00 11.50
[19] 12.00 12.50 13.00 13.50 14.00 14.50 15.00 15.50 16.00
[28] 16.50 17.00 17.50 18.00
```

5.1.2 Descriptive statistics

Then we convert functional data into fdata class with `fdata` function. The upload of `fda.usc` package is necessary. We create three `fdata` objects, corresponding to "boys" functional data `fdata.m`, "girls" functional data `fdata.f` and both together `fdata.growth`, where `argvals` is the vector that collects the ages at which heights were measured. Following the indications in Section 4.2, we transpose the numeric data matrix: the curves must be by rows and the observation time points in columns.

```
library(fda.usc)

argvals<-growth$age

fdata.m=fdata(t(growth$hgtm),
              argvals = argvals,
              names=list(main="Boys",xlab="ages",ylab="heigth"))

fdata.f=fdata(t(growth$hgtf),
              argvals = argvals,
              names=list(main="Girls",xlab="ages",ylab="heigth"))

fdata.growth<-fdata(t(cbind(growth$hgtm,growth$hgtf)), argvals = argvals)
```

Now, the objective is to summarize numerically and graphically the created `fdata` object. We computed central and dispersion measures for functional data as we could see in Section 4.3. The measures of the central tendency by groups are shown in Figure 5.1.

```
fac<-factor(rep(c("boy","girl"),c(39,54)))

ldata<-list("df" = data.frame(fac), "fdataobj" = fdataobj)

a<-func.mean.formula(fdataobj~fac, data = ldata)

dev.new()
```

```
plot(a)
legend(2,160,c("boys mean","girls mean"),fill=c(1,2),border=0, bty="n")
```

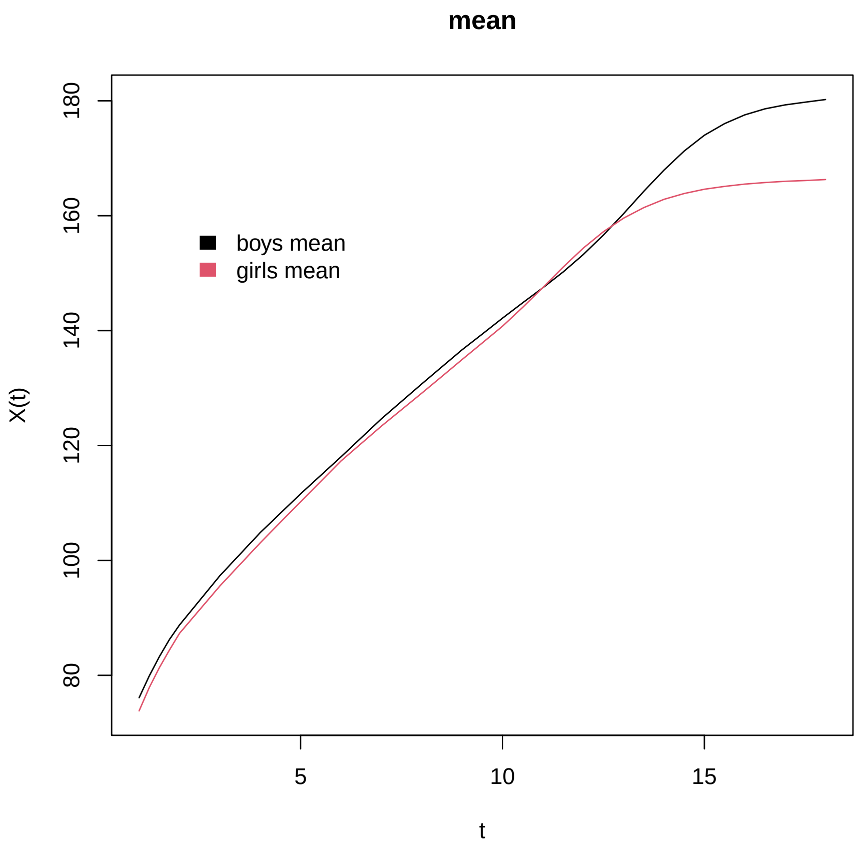


Figure 5.1: Measures of central tendency by groups

Other measures of central tendency, as we can see in Figure 5.2, are calculated for the whole dataset.

```
a1<-func.mean(fdataobj)
a2<-func.trim.FM(fdataobj)
a3<-func.trim.mode(fdataobj)
a4<-func.trim.RP(fdataobj)
a5<-func.med.FM(fdataobj)
a6<-func.med.mode(fdataobj)
```

```
a7<-func.med.RP(fdataobj)

dev.new()
par(mfrow=c(1,2))

plot(c(a1,a2,a3,a4),ylim=c(70,170),col=c(2,3,4,5),main="Central tendency:
trimmed mean")

legend(2,160,c("mean","trimmed mean FM criteria","trimmed mean mode criteria",
"trimmed mean RP criteria"),fill=c(2,3,4,5),border=0, bty="n")

plot(c(a1,a5,a6,a7),ylim=c(70,170),main="Central tendency: median",
col=c(2,3,4,5))

legend(2,160,c("mean","median FM criteria","median mode criteria",
"median RP criteria"),fill=c(2,3,4,5),border=0, bty="n")
```

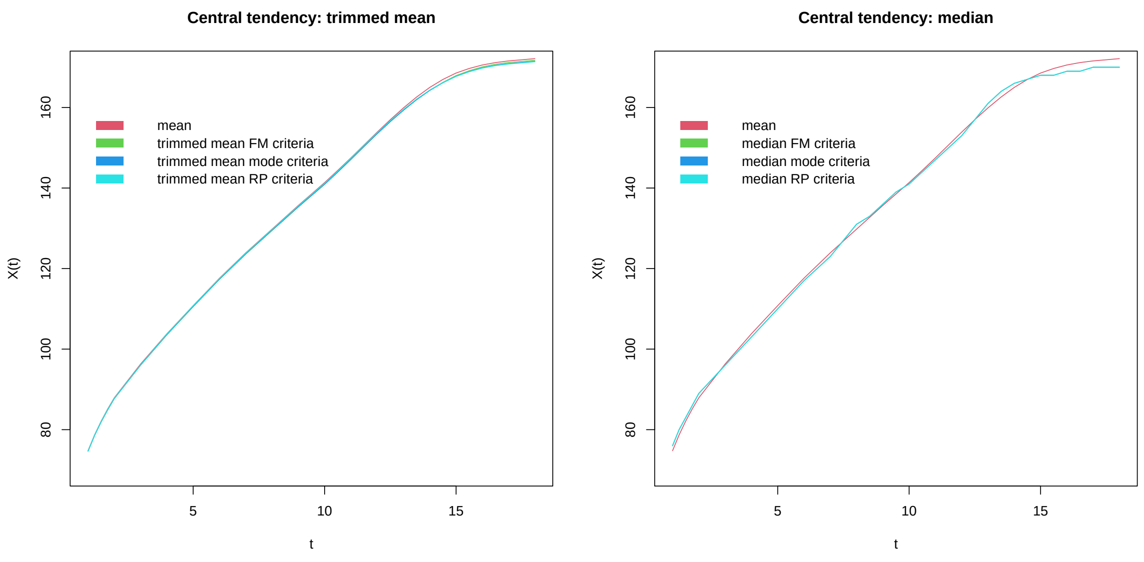


Figure 5.2: Mean, trimmed mean and median using FM criteria, mode criteria and RP criteria.

Measures of dispersion such as variance and trimmed variance using FM criteria, FM criteria with trimmed parameter `trim = 0.1`, mode criteria and RP criteria are being calculated, among others. A graphical representation of these dispersion measures is shown in Figure 5.3.

```
b1<-func.var(fdataobj)
b2<-func.trimvar.FM(fdataobj)
b3<-func.trimvar.FM(fdataobj,trim=0.1)
b4<-func.trimvar.mode(fdataobj)
b5<-func.trimvar.RP(fdataobj)

dev.new()
par(mfrow=c(1,2))

plot(c(b1,b2,b3),col=c(2,3,4),ylim=c(0,90),main="Measures of dispersion I")

legend(2,80,fill=c(2,3,4),border=0, bty="n",c("variance","trimmed var FM
criteria", "trimmed var FM criteria 10%"))

plot(c(b1,b4,b5),col=c(2,3,4),ylim=c(0,90),main="Measures of dispersion II")

legend(2,80,fill=c(2,3,4),border=0, bty="n",c("variance", "trimmed var mode
criteria", "trimmed var RP criteria"))
```

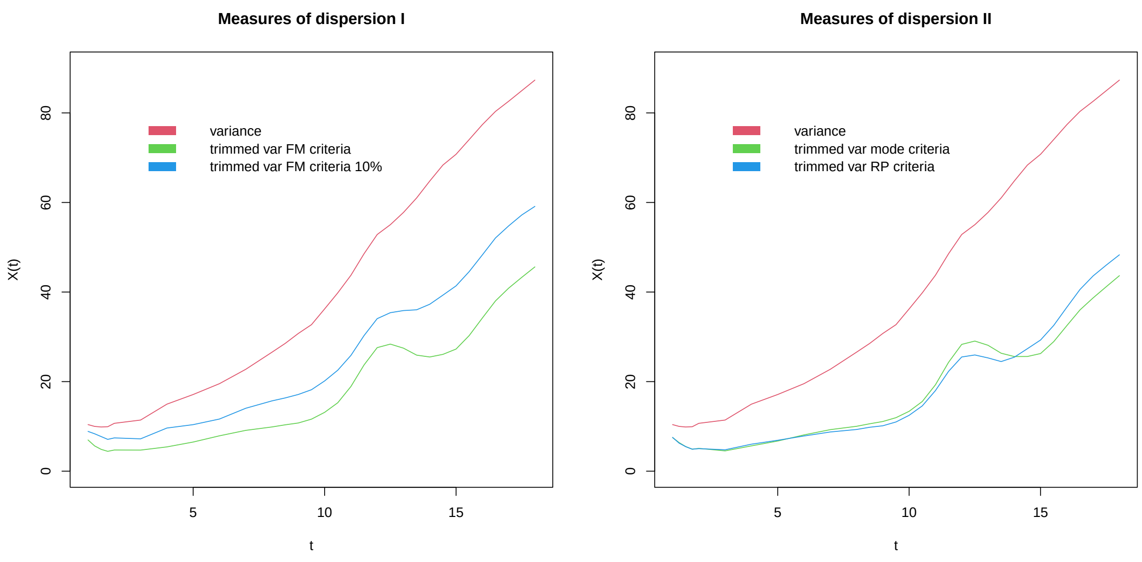


Figure 5.3: Measures of dispersion: variance, trimmed variance using FM criteria, FM criteria with trimmed parameter `trim = 0.1`, mode criteria and RP criteria.

Now, we compare the mean curve of each group, “boys” and “girls” with the rest of “boys” and “girls” curves, respectively. See Figure 5.4.

```
dev.new()
par(mfrow=c(1,2))

mean.m=func.mean(fdata.m)

matplot(growth$age,growth$hgtm,type="l",ylim = c(60,200))

lines(mean.m,col=2,lwd=6)

title("39 boys heights until 18",cex.sub=.1,col.main=2)

legend(0,200,"mean curve of heights",2,bty="n")

mean.f = func.mean(fdata.f)

matplot(growth$age,growth$hgtf,type="l",ylim = c(60,200))

lines(mean.f,col=2,lwd=6)

title("54 girls heights until 18",cex.sub=.1,col.main=2)

legend(0,200,"mean curve of heights",2,bty="n")
```

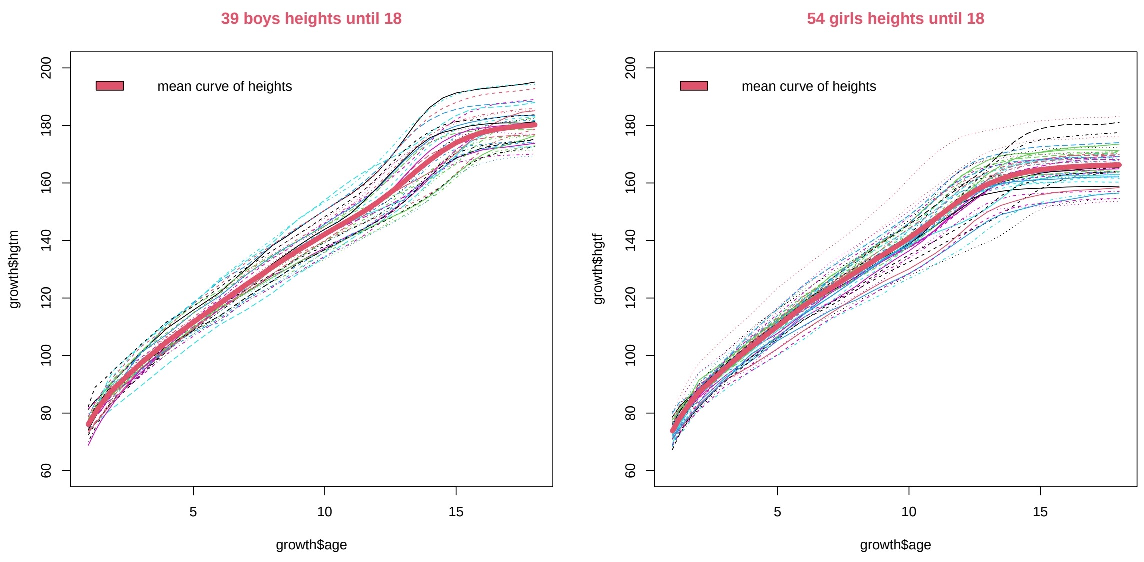


Figure 5.4: Comparing “boys” and “girls” curves with their mean curves.

One of the advantages of using functional data instead of the multivariate case is that derivatives of any order can be computed for the data, noted in Section 2.2. In Figure 5.5 and Figure 5.6, first and second order derivatives for each time series in the dataset are represented.

```
fdata.m.deriv<-fdata.deriv(fdata.m,nderiv=1,method = "bspline",
class.out = 'fdata')
fdata.m.deriv2<-fdata.deriv(fdata.m,nderiv=2,method = "bspline",
class.out = 'fdata')

dev.new()
par(mfrow=c(1,2))
plot(fdata.m.deriv)
plot(fdata.m.deriv2)

fdata.f.deriv<-fdata.deriv(fdata.f,nderiv=1,method = "bspline",
class.out = 'fdata')
fdata.f.deriv2<-fdata.deriv(fdata.f,nderiv=2,method = "bspline",
class.out = 'fdata')

dev.new()
par(mfrow=c(1,2))
plot(fdata.f.deriv)
plot(fdata.f.deriv2)
```

A fundamental part of classifying functional data is to compute the distances between every couple of time series that belong to the dataset. First, we check if the function `metric.dist` 4.5 computes the distance between two functional data according to the above-mentioned definition in Section 2.5. For that, the distance between the first boy in `fdata.m` and the first girl in `fdata.f`, `dist_boy01_girl01` is compare with the square root of the square difference of the corresponding rows "boy01" and "girl01", `diff_boy01_girl01`.

```
dist_boy01_girl01<-metric.dist(t(growth$hgtm),t(growth$hgtf),
method = "euclidean")[1,1]
```

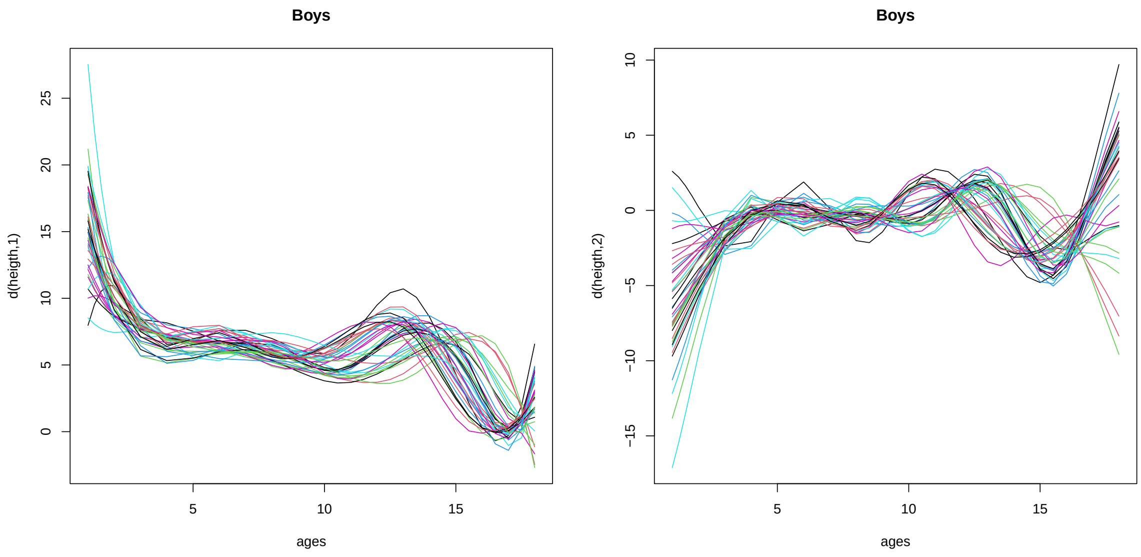


Figure 5.5: First and second order derivatives for "boys" functional data

```
diff_boy01_girl01<-growth$hgtm[,1]-growth$hgtf[,1]

round(sqrt(sum(diff_boy01_girl01^2))-dist_boy01_girl01,2)
```

Now, we calculate the distances between "girls" and "boys". Arguments of the function `metric.dist` must be a data matrix. For this reason, the data matrix `data` is created, adding a grouping variable ("boys" and "girls") which will serve us later. A representative submatrix of the distance matrix is shown in the bellow frame 5.1.2.

Figure 5.7 represents graphically the Euclidean distances between the whole dataset. Red labels or characters make mention to functional data corresponding to "boys", and green ones to "girls". The time series in `data` are represented in the horizontal axis (note that we have 93 curves). On the vertical axis, the distances between two functional data are shown. For example, for the first curve `boy01` (labeled as "1"), the functional data nearest to it (in the sense of Euclidean distances) are those labeled as "v", "0" or "y". These labels are all red, so they correspond to other "boys" functional data. In the positive sense of the horizontal axis, the label "0" appears in the 10 -th place. We conclude that `boy10` is very similar to `boy01` in height.

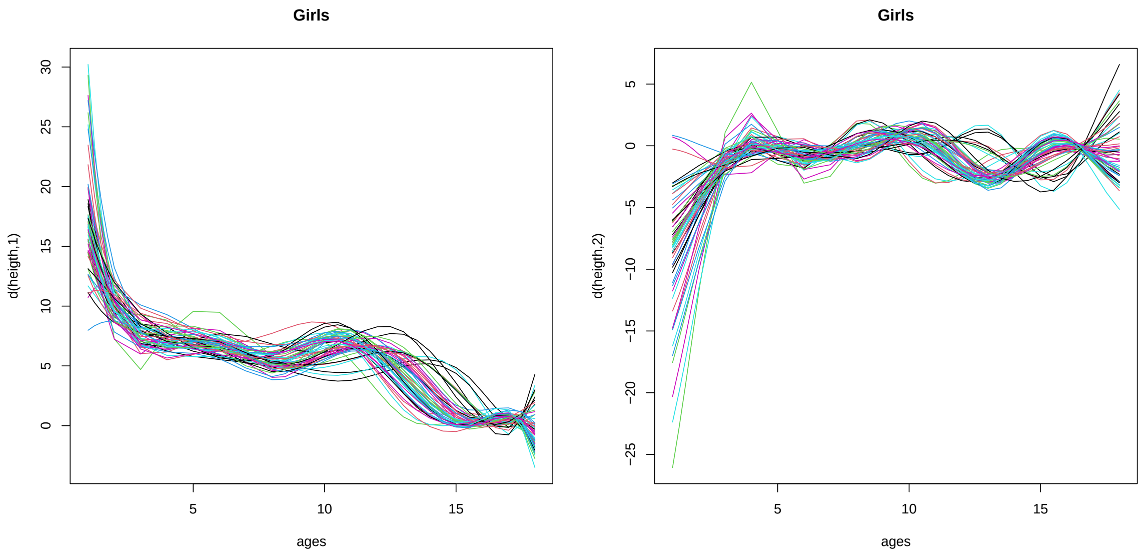


Figure 5.6: First and second order derivatives for “girls” functional data

Observation 5.1 As is known, every element satisfies that the distance with itself is 0. This is the reason why all the labels appear in the horizontal line `dist = 0`.

```
data=rbind(cbind(growth$hgtm,growth$hgtf),rep(c("boy","girl"),c(39,54)))

d<-metric.dist(t(data[1:31,]))

d[c(1:4,40:43),c(1:4,40:43)]

dev.new()

matplot(d,type="p",col=1+(data[32,]=="boy")+2*(data[32,]=="girl"))

title("Distances between girls and boys")
```

```
> d[c(1:4,40:43),c(1:4,40:43)]
          [,1]  [,2]   [,3]  [,4]   [,5]  [,6]  [,7]  [,8]
boy01     0.00 88.47 101.00 61.76 112.30 92.50 95.31 83.35
boy02    88.47  0.00  25.65 30.57  41.37 28.87 45.61 30.06
boy03   101.00 25.65   0.00 40.91  37.89 29.12 55.55 37.76
```

```
boy04    61.76 30.57  40.91  0.00  60.14 40.00 56.20 36.44
girl01 112.30 41.37  37.89 60.14   0.00 21.93 30.79 29.62
girl02  92.50 28.87  29.12 40.00  21.93  0.00 29.67 13.99
girl03  95.31 45.61  55.55 56.20  30.79 29.67  0.00 23.33
girl04  83.35 30.06  37.76 36.44  29.62 13.99 23.33  0.00
```

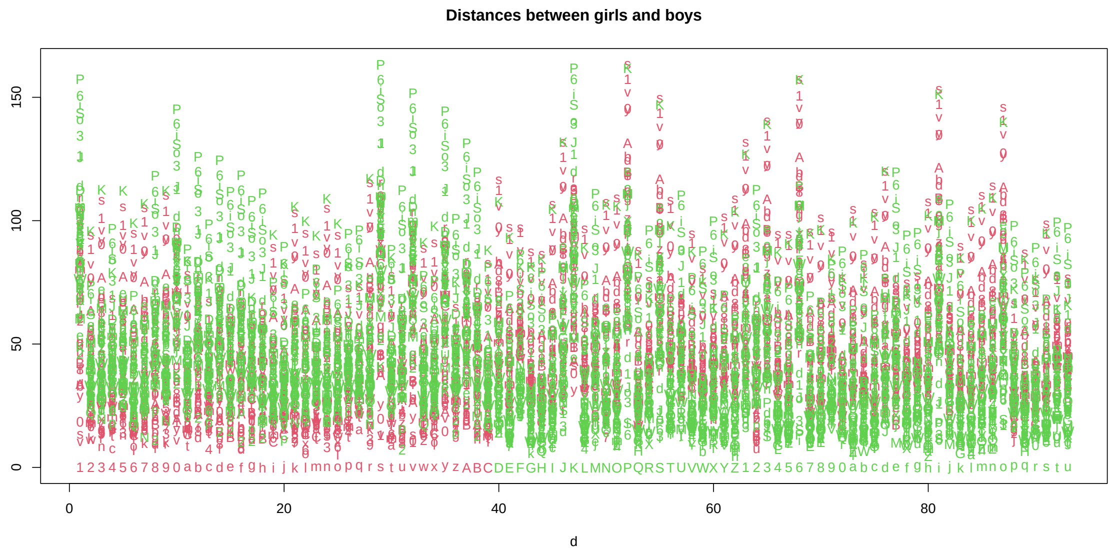


Figure 5.7: Euclidean distance between "boys" and "girls"

Then we do the same with the DTW distance. By default, the local cost (2.12) associated to compute the DTW distance between two functional data is the L^p-metric, approximated by Simpson's rule. We choose $p = 2$. Figure 5.8 follows the same idea that Figure 5.7 but with the new distance. Also, Figure 5.9 is created expanding Figure 5.8 to get a more detailed view.

```
dtw<-metric.DTW(fdata.growth,p=2)

dtw[c(1:4,40:43),c(1:4,40:43)]

dev.new()

matplot(dtw,type="p",col=1+(data[32,]=="boy")+2*(data[32,]=="girl"))
```

```
dev.new()

matplot(dtw,col=1+(data[32,]=="boy")+2*(data[32,]=="girl"),
xlim=c(0,20),ylim=c(0,1000))
```

```
> dtw[c(1:4,40:43),c(1:4,40:43)]
         [,1]    [,2]    [,3]    [,4]     [,5]    [,6]    [,7]    [,8]
[1,]     0.00 1813.09 4081.19 1149.59 10933.81 6555.96 8743.18 5694.87
[2,]  1813.09    0.00  166.57   70.41  1564.16  574.89 1055.91  384.92
[3,]  4081.19  166.57    0.00  616.90   897.10  182.22  500.32   95.25
[4,]  1149.59   70.41  616.90    0.00  3453.58 1492.78 2456.19 1166.30
[5,] 10933.81 1564.16  897.10 3453.58     0.00  413.29  150.94  670.89
[6,]  6555.96  574.89  182.22 1492.78   413.29    0.00  155.13   82.68
[7,]  8743.18 1055.91  500.32 2456.19   150.94  155.13    0.00  260.93
[8,]  5694.87  384.92   95.25 1166.30   670.89   82.68  260.93    0.00
```

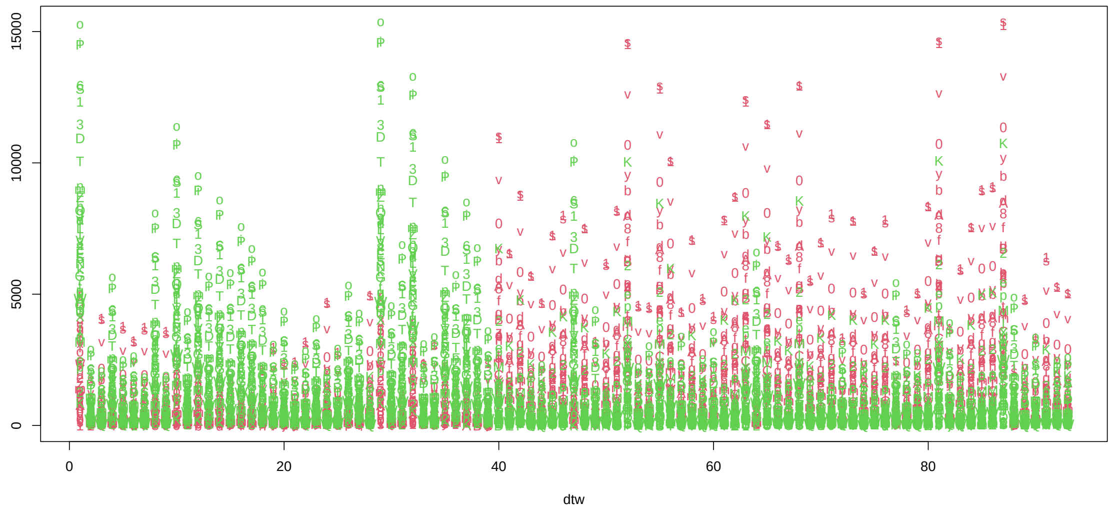


Figure 5.8: DTW distances between "boys" and "girls

We now calculate the DTW distance between the first boy in `fdata.m` and the first girl in `fdata.f`. The accumulated cost matrix is computed following Theorem 2.12. The function `findPath` computes the optimal warping path between these two

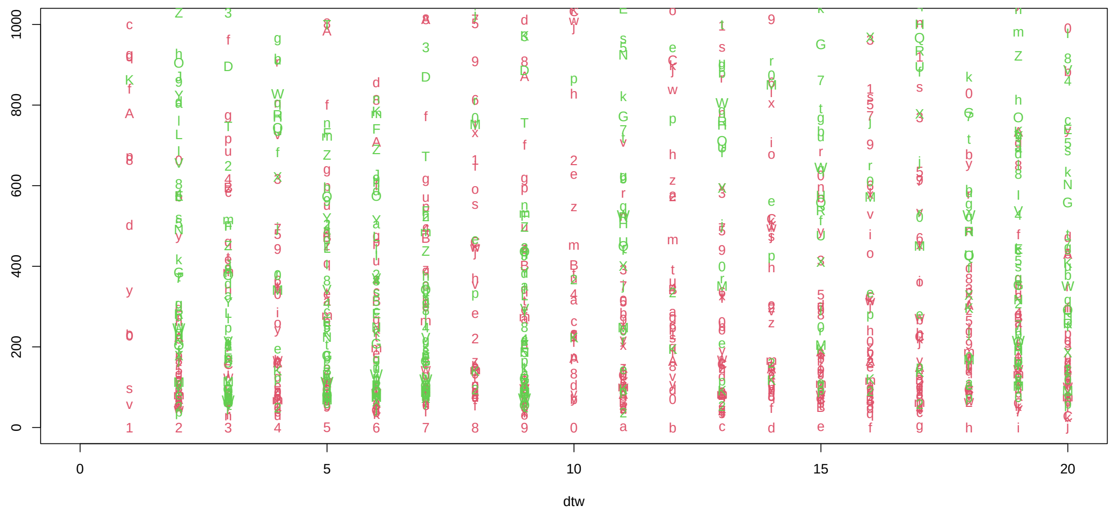


Figure 5.9: Figure 5.8 expanded in [0,20]×[0,1000].

time series using the Algorithm 2.5. We can see in Figure 5.10 how this warping between these two functional data has been executed.

```
D1=fda.usc:::DTW(fdata.m$data[1,],fdata.f$data[1,],p=2)

aa1=fda.usc:::findPath(D1$D)

dev.new()

plot(c(fdata.m[1,],fdata.f[1,]),col=c("blue","red"))

segments(fdata.m$argvals[aa1[,1]],
fdata.m[1]$data[aa1[,1]],fdata.f$argvals[aa1[,2]],
fdata.f[1]$data[aa1[,2]],lwd = 0.5,col=8)
```

In general, the results obtained about central and dispersion measures suggest that boys in Berkeley (California) are taller than girls. In addition, while examining the smoothness of the derivative curves, boys grow more sharply than girls, where the stages of the most rapid growth are childhood and adolescence (over 13 years). In Figure 5.7 and clearer in Figure 5.8, we can see that the most similar curves to the boys curves are themselves, and the same for girls functional data. More

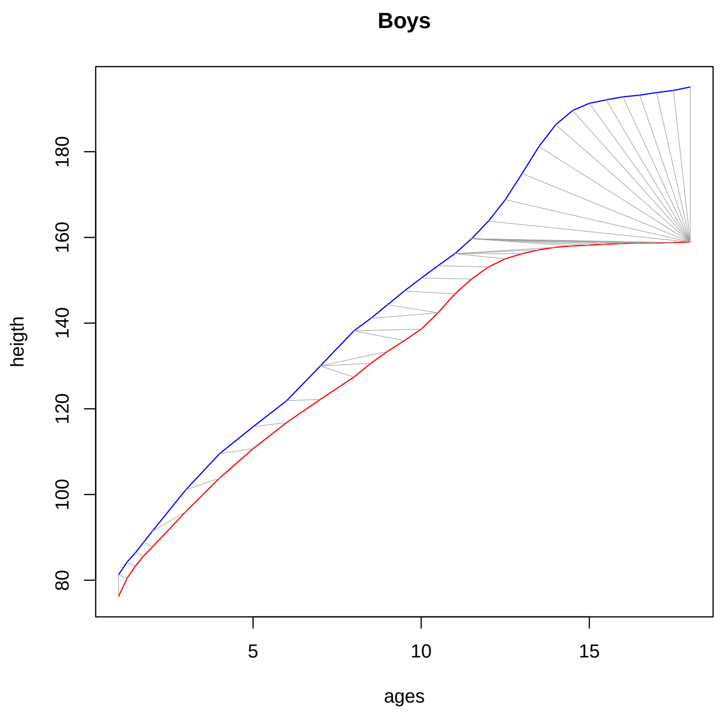


Figure 5.10: Warping path between the first boy in `fdata.m` and the first girl in `fdata.f`.

extreme values are observed in Figure 5.8 than in Figure 5.7, i.e., two time series are very similar or very different for the DTW distance, while for the Euclidean distance the distances are more homogeneous. This phenomenon can be explained by Figure 5.10, where the ages at which `boy01` grows more (13 years of age or older) are warped to the same point in the curve corresponding to `girl01`, which is experiencing stagnation of growth about 12 years old.

5.1.3 Unsupervised classification

We can consider the unsupervised classification problem. For that, K-means clustering method is introduced. Set `ncl = 2` the number of clusters and `method="exact"` since the sample size is not too large. Figure 5.11 shows the output of the evaluated function `kmeans.fd`. All functional data are represented and colored in red or green depending on the group in which they have been classified (red for "boys" and green for "girls"). In the right plot, we can see the center curves in the last step of the iterative K-means process. Furthermore, the confusion matrix is provided. Then we calculate the estimate of the total probability of correct classification.

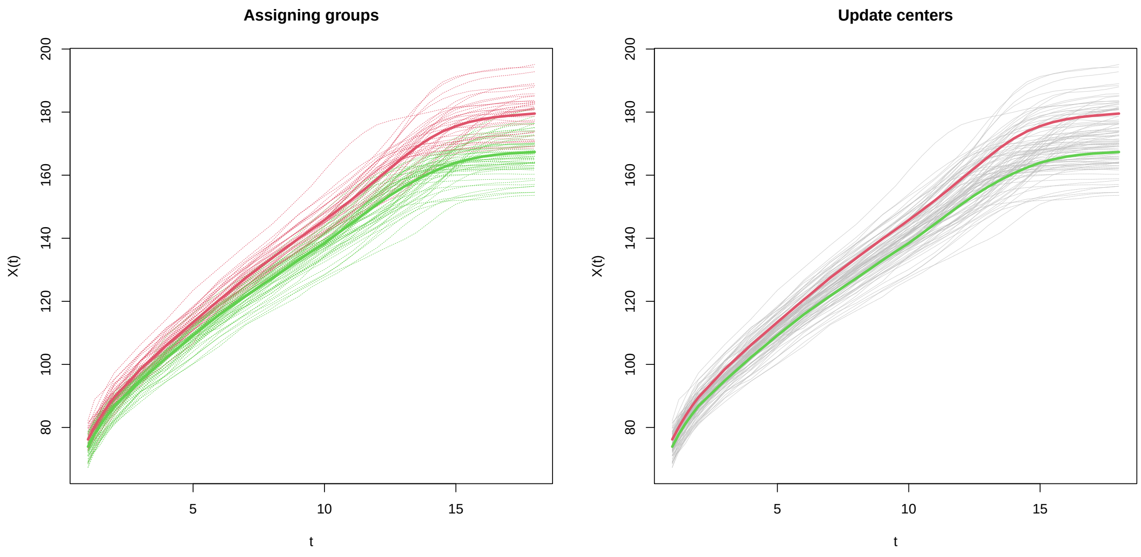


Figure 5.11: Red lines correspond to those curves classified as “boy” and green lines to “girls”.

```
dev.new()

unsu.classif=kmeans.fd(fdata.growth,ncl=2,draw=TRUE,method="exact")

table(unsu.classif$cluster,groups)

sum(diag(table(unsu.classif$cluster,groups)))/nrow.fdata(fdata.growth)
```

```
> table(unsu.classif$cluster,groups)
   groups
    boy girl
  1  27   27
  2  12   27

> sum(diag(table(unsu.classif$cluster,groups)))/nrow.fdata(fdata.growth)
[1] 0.5806452
```

5.1.4 Supervised classification

Every functional data in `fdata.growth` belongs to one of the two groups: "boys" or "girls" (2 populations). Then, we should consider the supervised classification problem with `growth` dataset. The final aim is to classify a new time series X into one of these populations, using the information provided by the training sample.

We will analyze the previous survey by the k-Nearest neighbors method, first by hand calculating in `R` and then using automatic procedures already implemented. We choose an arbitrary functional data in `fdata.growth`, for example the first observation, `boy01`. We shall forget this functional data belongs to "boys" group for classify it using the information of the rest of the dataset (the training sample). Implementation of k-Nearest neighbors method needs a chosen distance, for example the Euclidean distance. Now, we compute the k training data closest to `boy01` and assign it to P_j ("boys" or "girls") when the majority of these k data belong to P_j. See Section 2.1. Let $k = 5$, Figure 5.7 shows that the closest training data to `boy01` are those labeled as "v", "s", "0", "y", "A", which corresponds to the functional training data `boy32, boy29, boy10, boy35` and `boy37`. All of them have been classified as "boys", so we predict that `boy01` is a "boy" for $k = 5$. The distances among them are calculated bellow.

```
> metric.dist(t(growth$hgtm[,1]), t(growth$hgtm[,c(32,29,10,35,37)]))
        boy32    boy29    boy10    boy35    boy37
[1,] 10.13262 13.10458 18.38614 28.09609 33.81272
```

Also, the function `predict` could classify the functional data `boy01` by k-Nearest neighbors method as follows.

```
train.sample<-fdata.growth[c(2:93),]

groups<-data[32,]

train.groups<-groups[c(2:93)]
```

```
out<-classif.knn(train.groups,train.sample,knn=5)

summary(out)

test.sample<-fdata.growth[1,]

pred<-predict(out,test.sample)

pred
```

Our training data `train.sample` consists of `fdata.growth` but excluding the first functional data `boy01` since it is the time series of which we want to know the group. To construct the `classif` object using the k-Nearest neighbors method, a training sample, their corresponding groups and the neighbors number `knn` are needed. This is the reason why the character vector `groups` is created. The `summary` of the `classif` object and the resulting prediction follows.

```
> summary(out)
     - SUMMARY -

-Probability of correct classification by group (prob.classification):
y
      boy      girl
1.0000000 0.9444444

-Confusion matrix between the theoretical groups (by rows)
  and estimated groups (by column)

       boy girl
  boy   38    0
  girl   3   51

-Vector of probability of correct classification
   by number of neighbors (knn):
     5
0.9674
```

```
-Optimal number of neighbors: knn.opt= 5
with highest probability of correct classification max.prob=  0.9673913

-Probability of correct classification:  0.9674

> pred
[1] boy
Levels: boy girl
```

These results are in line with those obtained by hand calculations. Furthermore, `summary` of `classif` object, give us information about how good is the classification model with the training sample providing the confusion matrix. Moreover, an estimation of the probability of correct classification is displayed (properly classified number divided by sample size).

5.2 `CanadianWeather` dataset

5.2.1 Data description

`CanadianWeather` collects information about Canadian average annual weather cycle. This dataset included in `fda` package [12], contains the daily temperature and precipitation at 35 different locations in Canada averaged over 1960 to 1994. `CanadianWeather` is a list with the following components:

- `dailyAv` a three dimensional array c(365, 35, 3) summarizing data collected at 35 different weather stations in Canada on the following:
  - `dailyAv[,,1]=[,, 'Temperature.C']`: average daily temperature for each day of the year.
  - `dailyAv[,,2]=[,, 'Precipitation.mm']`: average daily rainfall for each day of the year rounded to 0.1 mm.
  - `dailyAv[,,3]=[,, 'log10precip']`: base 10 logarithm of `dailyAv[,,2]` after first replacing 27 zeros by 0.05 mm [11].

- `place` Names of the 35 different weather stations in Canada whose data are summarized in `dailyAv`. These names vary between 6 and 11 characters in length.
- `province` names of the Canadian province that contains each place.
- `coordinates` a numeric matrix giving N.latitude and W.longitude for each place.
- `region` Which of the 4 climate zones contain each place: Atlantic, Pacific, Continental and Arctic.
- `monthlyTemp` A matrix of dimensions (12×35) giving the average temperature in degrees Celsius for each month of the year.
- `monthlyPrecip` A matrix of dimensions (12×35) giving the average daily precipitation in millimeters for each month of the year.
- `geogindex` Order the weather stations from East to West to North.

First, we load the `CanadianWeather` data. Also, the `fda` [12] library must be uploaded if we had not done it before in Section 5.1. Let us then the different components of the list, displaying it in the first four places. Also, Figure 5.12, Figure 5.13 and Figure 5.14 show dataset graphical descriptions. Figure 5.12 shows that temperatures and precipitations waver much since there are many discrete time points of observation. A map of Canada is sensed in Figure 5.13. Places names are colored according to the climate zone (Atlantic, Pacific, Continental and Arctic) to which they belong (left plot) and the point shape changes depending on the province (right plot). Monthly temperature and precipitation curves are represented in Figure 5.14.

```
library(fda)

data("CanadianWeather")

x<-CanadianWeather$dailyAv

x[100:103,1:4,1]
x[100:103,1:4,2]
```

```
x[100:103,1:4,3]

dev.new()
par(mfrow=c(1,2))
matplot(x[,,1], type="l")
matplot(x[,,2], type="l")

place<-CanadianWeather$place
place[1:4]

province<-CanadianWeather$province
province[1:4]

coordinates<-CanadianWeather$coordinates
coordinates[1:4,]

region<-CanadianWeather$region
region[1:4]

dev.new()
par(mfrow=c(1,2))
plot(-coordinates[,2],coordinates[,1],type="n")
text(-coordinates[,2],coordinates[,1],place,cex=0.7,
col=1+(region=="Atlantic")+2*(region=="Continental")+
3*(region=="Artic")+4*(region=="Pacific"))
plot(-coordinates[,2],coordinates[,1],
col=1+(region=="Atlantic")+2*(region=="Continental")+
3*(region=="Artic")+4*(region=="Pacific"),
pch=1+(province=="Quebec")+2*(province=="British
Colombia")+3*(province=="Northwest Territories"))

y<-CanadianWeather$monthlyTemp
y[,1:4]

z<-CanadianWeather$monthlyPrecip
z[,1:4]

dev.new()
par(mfrow=c(1,2))
matplot(y,type="l")
matplot(z,type="l")
```

```
geo<-CanadianWeather$geogindex
```

```
> x[100:103,1:4,1]
      St. Johns Halifax Sydney Yarmouth
apr10       1.0     2.4    1.6      3.7
apr11       1.1     2.8    1.4      3.8
apr12       0.7     2.0    0.6      3.3
apr13       0.2     2.2    0.5      3.5

> x[100:103,1:4,2]
      St. Johns Halifax Sydney Yarmouth
apr10       3.3     6.1    6.8      4.8
apr11       6.6     5.0    6.0      3.1
apr12       5.3     2.2    3.8      1.4
apr13       2.8     2.5    1.3      3.0

> x[100:103,1:4,3]
      St. Johns   Halifax    Sydney  Yarmouth
apr10 0.5185139 0.7853298 0.8325089 0.6812412
apr11 0.8195439 0.6989700 0.7781513 0.4913617
apr12 0.7242759 0.3424227 0.5797836 0.1461280
apr13 0.4471580 0.3979400 0.1139434 0.4771213

> place[1:4]
[1] "St. Johns" "Halifax"   "Sydney"    "Yarmouth"

> province[1:4]
     St. Johns        Halifax         Sydney       Yarmouth
"Newfoundland"  "Nova Scotia"  "Nova Scotia"  "Nova Scotia"

> coordinates[1:4,]
          N.latitude W.longitude
St. Johns      47.34       52.43
Halifax        44.39       63.36
Sydney         46.09       60.11
Yarmouth       43.50       66.07

> region[1:4]
 St. Johns    Halifax     Sydney   Yarmouth
```

```
"Atlantic" "Atlantic" "Atlantic" "Atlantic"

> y[,1:4]
    St. Johns   Halifax    Sydney   Yarmouth
Jan -4.654839 -6.158065 -5.722581 -3.2161290
Feb -5.325000 -6.182143 -6.796429 -3.4892857
Mar -2.532258 -1.738710 -2.935484  0.1516129
Apr  1.256667  3.623333  1.850000  4.6866667
May  5.793548  9.441935  7.496774  9.3419355
Jun 10.786667 14.776667 13.140000 13.4033333
Jul 15.206452 18.380645 17.487097 16.2935484
Aug 15.280645 18.203226 17.635484 16.5967742
Sep 11.623333 13.866667 13.306667 13.5933333
Oct  7.019355  8.490323  8.274194  9.2451613
Nov  2.953333  3.240000  3.533333  4.9000000
Dec -1.845161 -2.987097 -2.029032 -0.4645161

> z[,1:4]
    St. Johns  Halifax   Sydney Yarmouth
Jan  4.651613 4.632258 4.764516 4.022581
Feb  4.735714 4.146429 4.446429 3.639286
Mar  4.235484 4.135484 4.422581 3.309677
Apr  3.616667 4.010000 4.076667 3.373333
May  3.251613 3.529032 3.180645 3.093548
Jun  3.270000 3.126667 2.976667 3.046667
Jul  2.651613 3.022581 2.754839 2.638710
Aug  3.822581 3.403226 3.116129 2.780645
Sep  4.126667 3.203333 3.506667 2.953333
Oct  4.909677 4.122581 4.551613 3.509677
Nov  4.783333 5.140000 5.193333 4.516667
Dec  4.680645 5.364516 5.509677 4.641935
```

Using the function `summary`, we can see the highest temperature or millimeters of precipitation achieved at each location during the year, as well as the minimum or the mean. A temperature summary of the first four location appearing in dataset follows.

```
> summary(x[,,1])[,1:4]
   St. Johns         Halifax          Sydney          Yarmouth
```

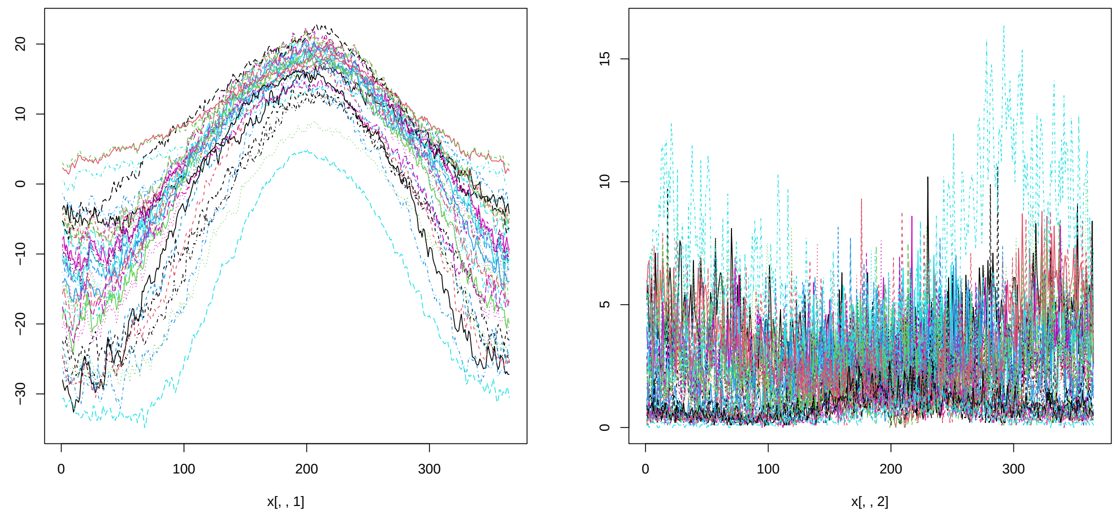


Figure 5.12: Average daily temperature for each day of the year at 35 different weather stations in Canada (left plot) and average daily rainfall for each day of the year rounded to 0.1 mm at 35 different weather stations in Canada.

```
Min.   :-7.00   Min.   :-8.10   Min.   :-8.40   Min.   :-5.300
1st Qu.:-2.10   1st Qu.:-2.60   1st Qu.:-2.40   1st Qu.:-0.100
Median : 4.50   Median : 6.40   Median : 5.40   Median : 7.400
Mean   : 4.69   Mean   : 6.15   Mean   : 5.51   Mean   : 6.812
3rd Qu.:11.40   3rd Qu.:14.00   3rd Qu.:13.10   3rd Qu.:13.500
Max.   :17.10   Max.   :19.80   Max.   :19.20   Max.   :17.700
```

5.2.2 Descriptive statistics

To address the classification problem, let us turn our attention to `monthlyTemp` and `monthlyPrecip` since `monthlyTemp` is itself a summary of `dailyAv[,,1]` and `monthlyPrecip` of `dailyAv[,,2]` (we are keeping with the averaged temperature per month). Because `dailyAv[,,3]` is a log transformation of `dailyAv[,,2]`, we will not analyze it either.

Next, we convert the functional data into `fdata` class. Uploading the `fda.usc` package is necessary. Two `fdata` objects, corresponding to "Temperature" `fdata.t` and "Precipitation.mm" `fdata.p` are created. Following the indications in Section

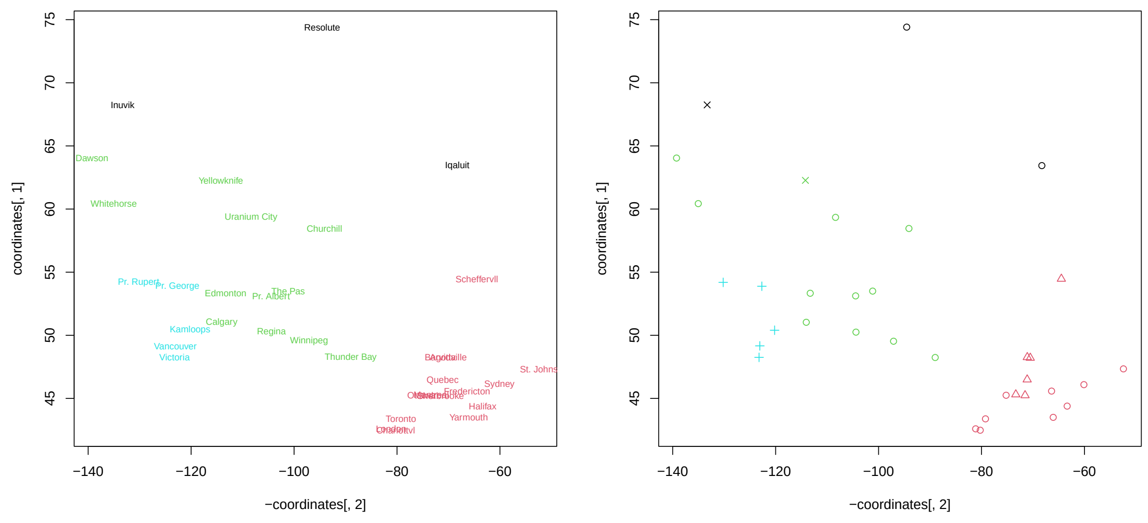


Figure 5.13: Coordinates of the 35 different weather stations in Canada. In red those that belong to ”Atlantic” climate zone, green corresponds to ”Continental” climate zone, black is for ”Arctic” region and blue, those that belong to ”Pacific” zone (left plot). At right the point shapes vary depending on the province: the places belong to ”Quebec” are represented with △, + correspond with those that are in ”British Colombia” province, × match with ”Northwest Territories”. The rest of provinces are assigned with ○.

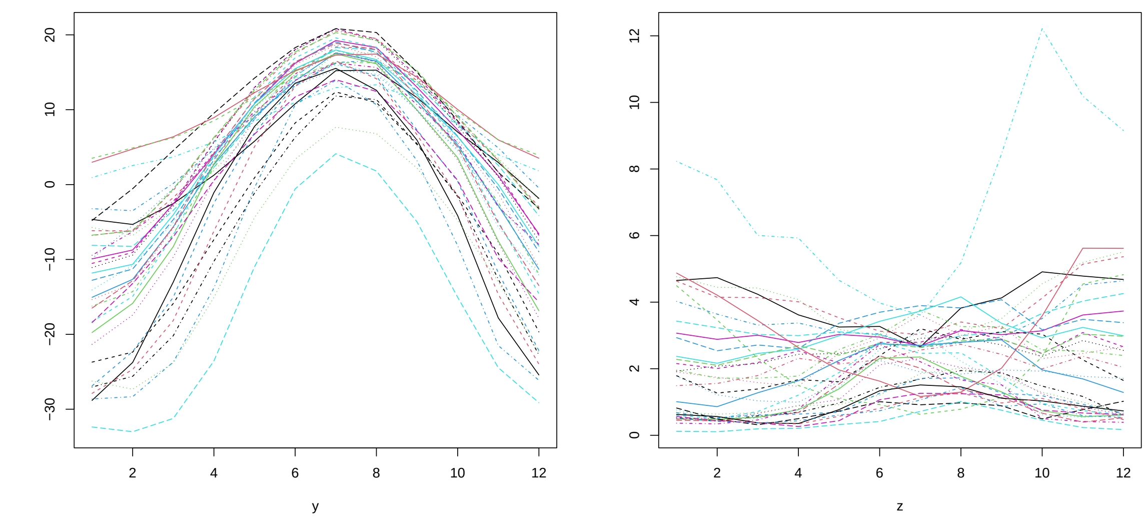


Figure 5.14: Average temperature in degrees Celsius for each month of the year (left plot) and average daily precipitation in millimeters for each month of the year (right plot).

4.2, we transpose the numeric data matrix: the curves must be by rows and the observation time points in columns.

```
library(fda.usc)

fdata.t=fdata(t(y))

fdata.p=fdata(t(z))
```

Now, the objective is to summarize numerically and graphically the created `fdata` object. We computed central and dispersion measures for functional data, as we could see in Section 4.3. If we present the problem of unsupervised classification for temperature or precipitation, the intuition suggests that there exists a good way to create groups or clusters, by region (Arctic, Atlantic, Continental and Pacific). For this reason, we will keep in mind the possible existence of these groups all through this Section 5.2.2. Simultaneously, we will study both `fdata` objects.

```
fac<-factor(region)

ldata.a<-list("df" = data.frame(fac), "fdataobj" = fdata.t)
ldata.b<-list("df" = data.frame(fac), "fdataobj" = fdata.p)

a<-func.mean.formula(fdataobj~fac, data = ldata.a)
b<-func.mean.formula(fdataobj~fac, data = ldata.b)

dev.new()
par(mfrow=c(1,2))
plot(a)
legend(5,-10,levels(fac),fill=1:4,border=0, bty="n")
plot(b)
legend(4,4,levels(fac),fill=1:4,border=0, bty="n")
```

In Figure 5.15 we can see the average monthly temperature by region (Arctic, Atlantic, Continental and Pacific) and the average monthly temperature by region. Higher temperatures are concentrated in the Summer months from June

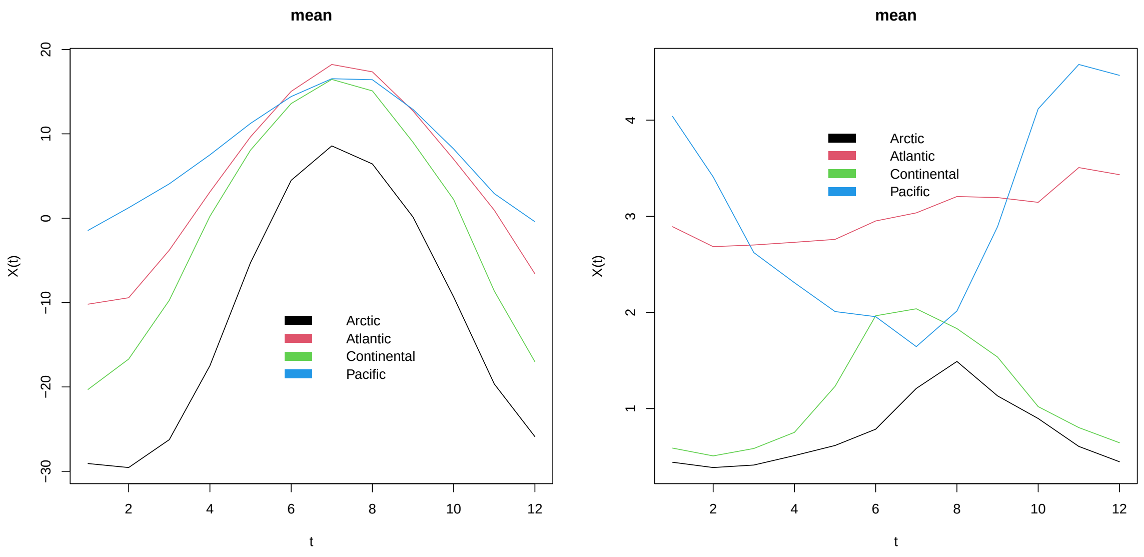


Figure 5.15: Average monthly temperature by region (left plot) and average monthly precipitation in millimeters by region (right plot).

until September. Pacific region is notorious as the hottest region of Canada and the Arctic region is the coldest. In the Arctic and Continental regions, rainfall is concentrated in the Summer months of June through September, whereas in the Pacific region, the Winter is very rainy (it rains throughout the year). Precipitation has remained constant through all the year in Atlantic region.

One of the advantages of using functional data instead of the multivariate case is that derivatives of any order can be computed for the data, noted in Section 2.3. In Figure 5.16 , first derivative for each time series in both dataset are represented.

```
fdata.t.deriv<-fdata.deriv(fdata.t, class.out = 'fdata', nbasis = 4)
fdata.p.deriv<-fdata.deriv(fdata.p, class.out = 'fdata', nbasis = 4)

dev.new()
par(mfrow=c(1,2))
plot(fdata.t.deriv)
plot(fdata.p.deriv)
```

Functional data classification (both, supervised and unsupervised) requires measures of proximity between two curves. For that, we are going to calculate the distances (Euclidean and DTW) between the “Temperature” curves, and on the

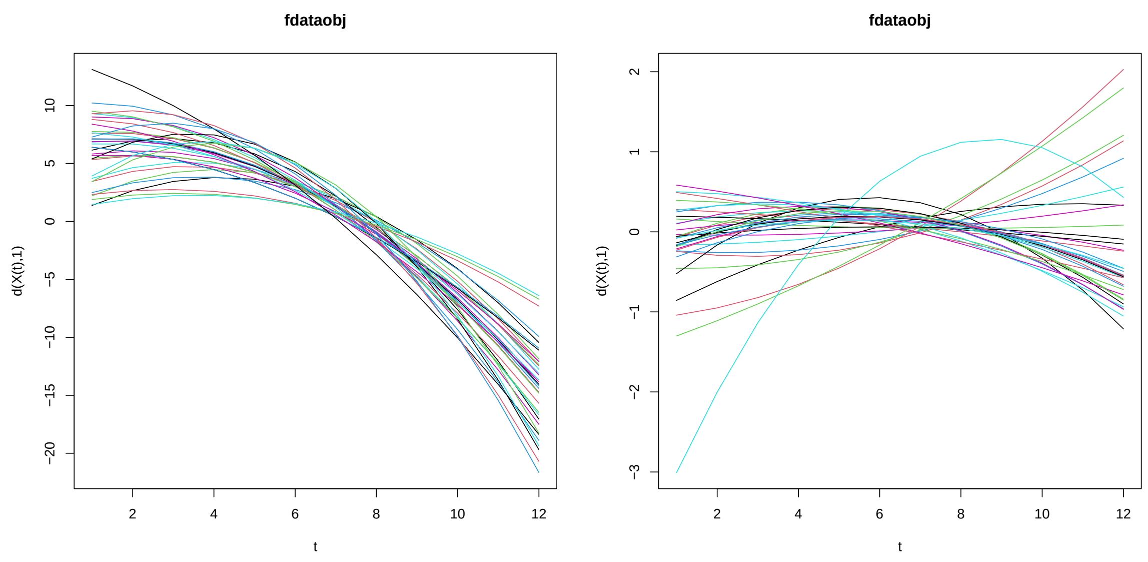


Figure 5.16: Monthly temperatures first derivatives (left plot) and monthly precipitations first derivatives.

other hand between "Precipitation.mm" curves. Moreover, we are going to study how equal or different are the 35 locations belong to Canada, depending on their temperatures or their precipitations.

```
dy<-metric.dist(t(y))
dy[1:4,1:4]



dz<-metric.dist(t(z))
dz[1:4,1:4]

dev.new()
par(mfrow=c(1,2))
matplot(dy,col=1+(region=="Atlantic")+2*(region=="Continental")+
3*(region=="Artic")+4*(region=="Pacific"))
legend(0,100,levels(fac),fill=c(1,2,3,5),border=0, bty="n")

matplot(dz,col=1+(region=="Atlantic")+2*(region=="Continental")+
3*(region=="Artic")+4*(region=="Pacific"))
legend(0,25,levels(fac),fill=c(1,2,3,5),border=0, bty="n")

dtwy<-metric.DTW(fdata.t,p=2)
```

```
dtwy[1:4,1:4]

dtwz<-metric.DTW(fdata.p,p=2)
dtwz[1:4,1:4]

dev.new()
par(mfrow=c(1,2))
matplot(dtwy,col=1+(region=="Atlantic")+2*(region=="Continental")+
3*(region=="Artic")+4*(region=="Pacific"))
legend(0,8000,levels(fac),fill=c(1,2,3,5),border=0, bty="n")

matplot(dtwz,col=1+(region=="Atlantic")+2*(region=="Continental")+
3*(region=="Artic")+4*(region=="Pacific"))
legend(0,8000,levels(fac),fill=c(1,2,3,5),border=0, bty="n")
```

```
>dy[1:4,1:4]
              [,1]     [,2]     [,3]     [,4]
St. Johns 0.000000 8.102809 5.271054 7.828830
Halifax   8.102809 0.000000 3.753839 6.261523
Sydney    5.271054 3.753839 0.000000 6.790745
Yarmouth  7.828830 6.261523 6.790745 0.000000

>dz[1:4,1:4]
              [,1]      [,2]      [,3]     [,4]
St. Johns 0.000000 1.7292430 1.5204781 2.661606
Halifax   1.729243 0.0000000 0.8901354 1.948340
Sydney    1.520478 0.8901354 0.0000000 2.382512
Yarmouth  2.661606 1.9483399 2.3825116 0.000000

>dtwy[1:4,1:4]
         [,1]     [,2]     [,3]     [,4]
[1,]  0.00000 42.66202 27.78401 31.42942
[2,] 42.66202  0.00000 14.09131 39.20667
[3,] 27.78401 14.09131  0.00000 39.62530
[4,] 31.42942 39.20667 39.62530  0.00000

>dtwz[1:4,1:4]
         [,1]      [,2]      [,3]     [,4]
[1,] 0.000000 1.2156991 1.8384273 1.494167
```

```
[2,] 1.215699 0.0000000 0.6262885 1.727500
[3,] 1.838427 0.6262885 0.0000000 2.246582
[4,] 1.494167 1.7275001 2.2465817 0.000000
```

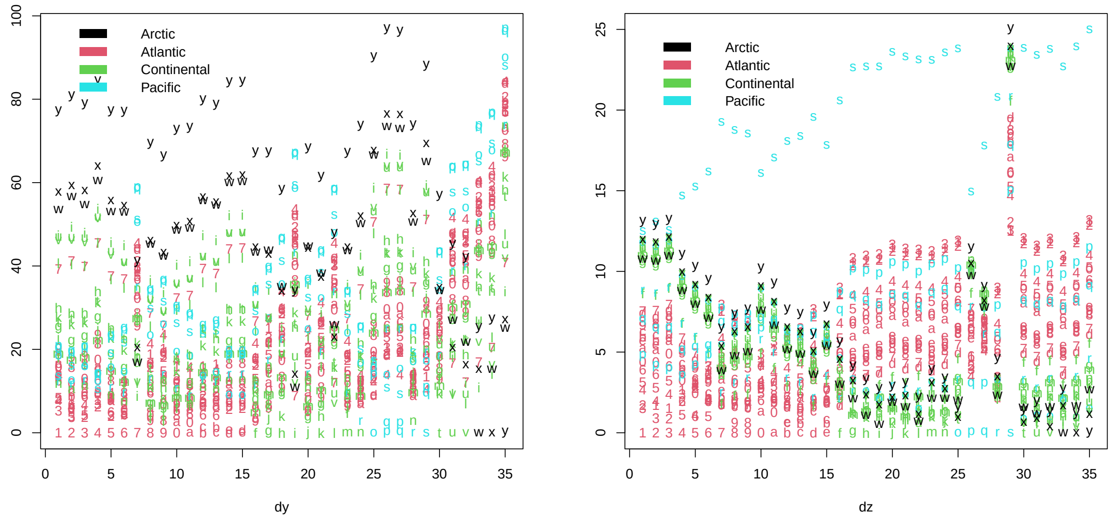


Figure 5.17: Euclidean distances between 35 places in Canada according to temperature (left plot) and their precipitations (right plot).

In Figure 5.17, we can see that with the Euclidean distance and considering the temperature, there is clearly a region which differs from the other, the Arctic region (the coldest region). The most distinctive one in terms of the precipitations are the Pacific, the most rainy region (as we also saw in Figure 5.15). We see in Figure 5.18 that the results are equivalent.

5.2.3 Unsupervised classification

The unsupervised classification problem is considered, which we will resolve using automatic procedures implemented in `R`. For that, `kmeans.fd` function is introduced in Section 4.10. Set `ncl = 4`, the number of clusters and `method="exact"` since the sample size is not too large. We choose the numbers of groups equals to 4 for same the idea above in 5.2.2.

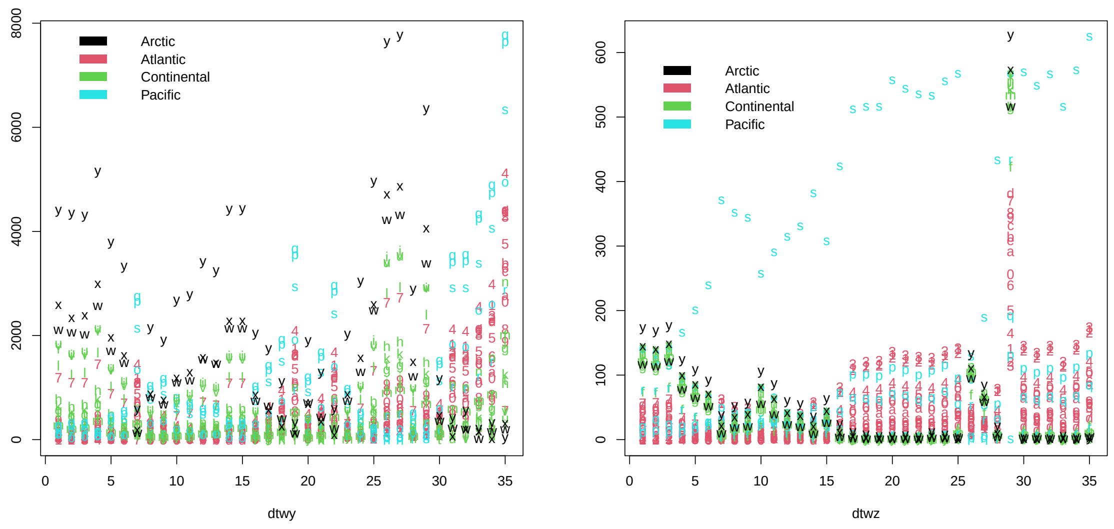


Figure 5.18: DTW distances between 35 places in Canada according to temperature (left plot) and their precipitations (right plot).

```
dev.new()
temp.cluster=kmeans.fd(fdata.t,ncl=4,draw=TRUE,method="sample",cluster.size=1)

dev.new()
prec.cluster=kmeans.fd(fdata.p,ncl=4,draw=TRUE,method="sample",cluster.size=1)
```

In both, Figure 5.19 and Figure 5.20 all functional data are represented and colored depending on the group in which they have been classified. On the right, we can see the center curves in the last step of the iterative K-means process.

5.2.4 Supervised classification

We could consider the supervised classification problem with `monthlyPrecip` dataset since every functional data in `fdata.p` belongs to a region (Arctic, Atlantic, Continental, Pacific). The final aim is to classify the first curve in `fdata.p` by taking the rest of dataset as a training sample. For this, we are going to create a classification model using k-Nearest neighbors method implemented by `classif.knn` function.

To construct the `classif` object using the k-Nearest neighbors method, a training sample, their corresponding groups and the neighbors number `knn` are needed.

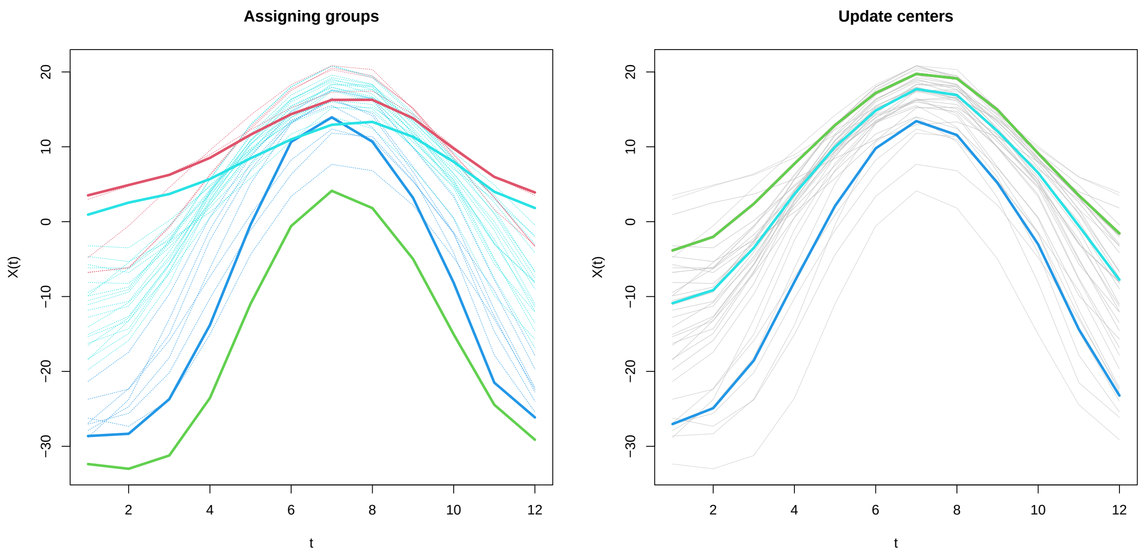


Figure 5.19: K-means. Average monthly temperature curves colored depending on the group in which they are classified.

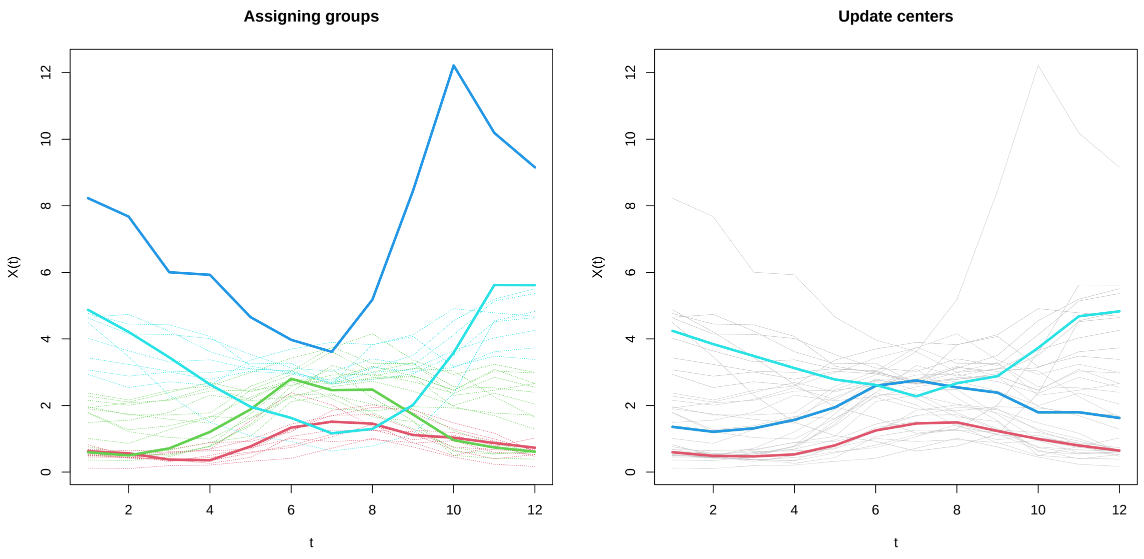


Figure 5.20: K-means. Average monthly precipitation curves colored depending on the group in which they are classified.

Training data `train.sample` consists of `fdata.p` but excluding “St. Johns” row, since it is the time series of which we want to know the group. The associated vector of the groups `train.groups` is the character vector `region`, but converted from “character” to “factor”. The summary of the `classif` object and the resulting prediction follows.

```
train.sample<-fdata.p[2:35,]

train.groups<-fac[2:35]

out<-classif.knn(train.groups,train.sample,knn=3)

summary(out)

test.sample<-fdata.p[1,]

pred<-predict(out,test.sample)
pred
```

```
>summary(out)
     - SUMMARY -

-Probability of correct classification by group (prob.classification):
y
     Arctic     Atlantic Continental     Pacific
  0.0000000   1.0000000   0.9166667   0.0000000

-Confusion matrix between the theoretical groups (by rows)
  and estimated groups (by column)

              Arctic Atlantic Continental Pacific
  Arctic           0        0           3       0
  Atlantic         0       14           0       0
  Continental      0        1          11       0
  Pacific          0        3           2       0

-Vector of probability of correct classification
```

```
   by number of neighbors (knn):
     3
0.7353

-Optimal number of neighbors: knn.opt= 3
with highest probability of correct classification max.prob=  0.7352941

-Probability of correct classification:  0.7353

> pred
[1] Atlantic
Levels: Arctic Atlantic Continental Pacific
```

The prediction is correct since "St. Johns" belongs to the Atlantic region. But looking at the confusion matrix, we realize that this model is not very reliable. In the dataset, three of the curves within the Arctic region, and none of them are classified as such. The same happens for the five functional data that belong to "Pacific", three of them are classified as "Atlantic" and the other two as "Continental". This problem may arise from a lack of functional data, in general, or from a lack of data within the Arctic and Pacific regions compared to the other two regions ("Atlantic" and "Continental" data are three times more than "Arctic" and "Pacific" data).

Bibliography

[1] Pablo León Alcaide. Ag-costane: Algoritmos genéticos para el cálculo del centroide de un grupo de series temporales mediante una distancia no euclídea. 2017.

[2] Amparo Baíllo, Antonio Cuevas, and Ricardo Fraiman. Classification methods for functional data. *The Oxford handbook of functional data analysis*, 2011.

[3] Michael R. Berthold and Frank Höppner. On clustering time series using euclidean distance and pearson correlation. *CoRR*, abs/1601.02213, 2016.

[4] Antonio Cuevas. A partial overview of the theory of statistics with functional data. *Journal of Statistical Planning and Inference*, 147:1–23, 2014.

[5] Chotirat Ann Ratanamahatana Eamonn Keogh. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7:358–386, 2005.

[6] Manuel Febrero-Bande and Manuel Oviedo de la Fuente. Statistical computing in functional data analysis: The r package fda.usc. *Journal of Statistical Software*, 51(4):1–28, 2012.

[7] Hesam Izakian, Witold Pedrycz, and Iqbal Jamal. Fuzzy clustering of time series data using dynamic time warping distance. *Engineering Applications of Artificial Intelligence*, 39:235–244, 2015.

[8] Piotr Kokoszka and Matthew Reimherr. *Introduction to functional data analysis*. Chapman and Hall/CRC, 2017.

[9] Carlos Arturo Lesmes. Como hacer un reporte estadístico con sweave. r y latex via sweave. *Comunicaciones en Estadística*, 6(1):9–19, 2013.

[10] Meinard Müller. *Dynamic Time Warping*, pages 69–84. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[11] James O Ramsay and Bernard W Silverman. *Fitting differential equations to functional data: Principal differential analysis*. Springer, 2005.

[12] JO Ramsay, Hadley Wickham, Maintainer JO Ramsay, and Suggests deSolve. Package 'fda', 2021.

[13] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning*. Springer, 2001.

[14] Jane-Ling Wang, Jeng-Min Chiou, and Hans-Georg Müller. Functional data analysis. *Annual Review of Statistics and Its Application*, 3(1):257–295, 2016.

[15] Wenjie Wang and Jun Yan. Shape-restricted regression splines with R package splines2. *Journal of Data Science*, 19(3):498–517, 2021.

[16] Wenjie Wang and Jun Yan. *splines2: Regression Spline Functions and Classes*, 2021. R package version 0.4.5.