Functional Data Analysis

Faculty of Sciences. PCEO Mathematics and Statistics

Pablo Borrego Ramos

Junio 2025

Julian Ramajo Hernández, professor at the Department of Economics of the University of Extremadura,

CERTIFIES:

That Mr. Pablo Borrego Ramos has completed under his supervision the Bachelor's Thesis and considers that the report meets the necessary requirements for its evaluation.

Badajoz, June 6, 2025.

Signed: Julian Ramajo Hernández

Contents

Introduction

Currently, the growth in both the volume and complexity of data has created the need for new statistical tools capable of handling them more appropriately. In this context, functional data analysis (FDA) emerges as a natural extension of classical multivariate analysis, allowing the study of observations that take the form of continuous functions. The aim of this work is to explore both the theoretical foundations and practical applications of FDA, with special emphasis on its application to the analysis of financial time series, such as the returns of European bonds.

As in the statistical analysis of discrete observations, the main objectives of functional analysis are data description, classification, and inference. However, we will encounter different challenges, such as the cardinality of the number of observations, functional misalignment, etc.

Unlike other methods that reduce functions to a discrete set of values, functional data analysis preserves the information of the entire function, allowing for a more concise analysis.

Functional data analysis is used in many fields, including medicine, biophysics, neuroscience, and chemistry, where technologies such as imaging techniques, electroencephalographs and electrocardiographs, gyroscopes, or accelerometers allow the continuous collection of functional data.

This work is divided into two parts: a theoretical part, where the descriptive and inferential theory of functional data analysis is developed, and a second part where this theory is applied to a specific case, the returns of European bonds over their maturity period.

In Chapter 1, the fundamental concepts of functional data analysis are introduced, with an emphasis on various descriptive procedures that allow studying the nature of functions. Additionally, the transformation of a functional analysis problem from a practical approach to a theoretical one is discussed.

Chapter 2 studies one of the most common descriptive techniques, clustering, in this case of functions, and presents the general framework of the three most common clustering methods, along with a specific example for each method.

Chapter 3 presents different regression methods applied to functions, both linear and nonlinear. Functional regression is classified according to the location of the functional data, whether in the independent variables, the dependent variable, or both.

Finally, Chapter 4 applies all the developed theory to a real functional dataset: the returns of European bonds over their maturity period. Returns are analyzed over

655 business days between December 29, 2006, and July 24, 2009. When applying a functional regression model, the daily value of the IBEX35 index between these dates will be included as the dependent variable.

The preparation of this work was based on the review and study of specialized literature in functional data analysis, both books and academic articles. Key references include works such as *Functional Data Analysis* by Ramsay and Silverman, as well as recent articles published in statistical and econometric journals. For the practical part, public data on European bonds and historical IBEX35 series were used. The analysis was primarily carried out using the R programming language.

Chapter 1

Fundamental Concepts

In this chapter, we introduce some basic concepts used in functional data analysis.

Generally, functional data are functions $f : \mathcal{T} \subset \mathbb{R}^k \mapsto \mathbb{R}^{k'}$, where $k, k' \in \mathbb{N}$, usually taking values $1, 2$, or 3.

Any statistical method combines information from different observations to make inferences about the populations from which they were sampled. Most basic methods involve replication, which consists of combining information across different sampled units, or regularization, which consists of combining information within each sampled unit. Functional data analysis involves both approaches.

We will begin by studying how to transform a discrete set of observations into a functional observation.

1.1 Data Handling

What does it mean that an observation $X : \mathcal{T} \mapsto \mathbb{R}$ is functional data? It does not mean that X is actually recorded for every value of t, as that would require storing an infinite number of values. Rather, it implies the assumed existence of a function X that gives rise to the observed data.

Furthermore, most of the time, we will seek smooth functional data; smoothness, in the sense of possessing a certain number of derivatives, is a property of the function X, and may not be immediately apparent from the data vector $(X(t_1), ..., X(t_n))$. In this case, smoothness can be interpreted as consecutive data points $X(t_j)$ and $X(t_{j+1})$ not differing too much from each other.

Given a functional observation $\{X(t_i)\}_{i \in I}$, a general model for the observations is of the form $X(t_i) = \hat{X}(t_i) + \epsilon_i$, where $X(t_i)$ is the observed value, $\hat{X}(t)$ is the smoothed function, and ϵ_i are the fitting errors.

Therefore, in order to apply much of the theory of functional data analysis, it is necessary to convert discrete observations into functions. The general idea is to express $\hat{X}(t)$ as a linear combination of basis functions $\sum_{i=1}^{K} c_i \phi(t_i)$ and estimate c_i using some smoothing method on the already observed discrete function values $\{X(t_i)\}_{i \in I}$.

We divide the process into two steps: determining the basis functions and estimating

the parameters.

1.1.1 Basis Functions

A system of basis functions is a set of known functions $\{\phi_k(t)\}$ that are mathematically independent of each other and have the property that any function can be approximated arbitrarily well by taking a weighted sum of a sufficiently large number K of these functions.
Given a function $\hat{X}(t)$, it can be represented by a linear expansion

$$\hat{X}(t) = \sum_{k=1}^{K} c_k \phi_k(t)$$

in terms of K known basis functions ϕ_k.
If we define $\mathbf{c}$ as the vector of length K containing the coefficients c_k and $\boldsymbol{\phi}$ as the functional vector whose elements are the basis functions ϕ_k, we can also express the above equation in matrix notation as

$$\hat{X}(t) = \mathbf{c}'\boldsymbol{\phi} = \boldsymbol{\phi}'\mathbf{c}.$$

This allows us, once a function basis is fixed, to associate each function with a vector of a given dimension K. As K increases, the approximation of $\hat{X}(t)$ becomes more precise. Therefore, K should be seen as a variable depending on the characteristics of $\hat{X}(t)$. Some examples of basis functions are:

- Power series:
  $$1, t, t^2, t^3, \ldots, t^k, \ldots.$$
  These approximate analytic and smooth functions well.

- Fourier series:
  $$1, \sin(\omega t), \cos(\omega t), \sin(2\omega t), \cos(2\omega t), \ldots, \sin(k\omega t), \cos(k\omega t), \ldots.$$
  These function series fit well to functions exhibiting periodicity, continuity, or smoothness.

- Wavelets:
  Starting from a so-called mother wavelet, which satisfies certain properties, such as being nonzero only in a small region, taking values of different signs, having zero mean, etc. By scaling and translating this function, a wavelet basis is obtained, of the form:
  $$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k)$$
  These functions allow capturing sudden changes in the function.

- B-Splines:

  A set of piecewise polynomial functions defined over a sequence of points $t_0, t_1, \ldots, t_m$, called knots.

  A B-Spline of degree p with knots $t_0, t_1, \ldots, t_m$ is constructed recursively:

$$B_{i,0}(t) = \begin{cases} 1, & t_i \leq t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

  For $p \geq 1$, the basis functions are recursively defined as:

$$B_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} B_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} B_{i+1,p-1}(t)$$

  They approximate well smooth functions and functions with continuous derivatives.

Once the structure of the function to be approximated is known, the basis functions that best fit this structure will be chosen.

1.1.2 Smoothing

Considering a general model for the observations $X(t_i) = \hat{X}(t_i) + \epsilon_i$ where $i \in I = \{1 \ldots n\}$.
Given a dataset $\{X(t_i), t_i\}_{i \in S}$, a smoothing spline is obtained as the smooth (twice differentiable) function $s(x)$ that minimizes the residual sum of squares plus a penalty measuring its roughness:

$$\sum_{i \in I} \left(X(t_i) - s(t_i)\right)^2 + \lambda \int \left(s''(x)\right)^2 dx$$

where $0 \leq \lambda < \infty$ is the smoothing hyperparameter.
Once we have expressed the function $\hat{X}(t)$ as a weighted sum of a basis $\{\phi_k(t)\}_k$ such that:

$$\hat{X}(t) = \sum_{k=1}^{K} c_k \phi_k(t)$$

the goal will be to determine the coefficients c_k so that they produce a smooth function. Some of the most common smoothing methods are:

- Least Squares Smoothing: The coefficients $\{c_k\}_{k=1,\ldots,K}$ are those that minimize the function

$$LS(X|c) = \sum_{i=1}^{n} \left(X(t_i) - \sum_{k=1}^{K} c_k \phi_k(t_i) \right)^2 .$$

  Let's express this in matrix form.

Defining $\mathbf{c} = (c_1, \ldots, c_K)^\top$, $\mathbf{X} = (X(t_1), \ldots, X(t_n))^\top$ and $\mathbf{\Phi}$ as the $n \times K$ matrix with entries $\phi_k(t_i)$, we can write:

$$LS(X|c) = (\mathbf{X} - \mathbf{\Phi c})^\top (\mathbf{X} - \mathbf{\Phi c}) = \|\mathbf{X} - \mathbf{\Phi c}\|^2.$$

Taking the derivative with respect to $\mathbf{c}$, we obtain:

$$2\mathbf{\Phi}'\mathbf{\Phi c} - 2\mathbf{\Phi}'\mathbf{X} = 0,$$

and solving this equation for $\mathbf{c}$ yields the estimate $\hat{\mathbf{c}}$ that minimizes the least squares criterion:

$$\hat{\mathbf{c}} = (\mathbf{\Phi}'\mathbf{\Phi})^{-1}\mathbf{\Phi}'\mathbf{X}.$$

- Weighted Least Squares Smoothing: In this case, the equation to minimize is:

$$WLS(X|c) = (\mathbf{X} - \mathbf{\Phi c})'\mathbf{W}(\mathbf{X} - \mathbf{\Phi c})$$

where $\mathbf{W}$ is a symmetric positive definite matrix that assigns different weights to the observations. Typically, $\mathbf{W} = \mathbf{\Sigma}_\epsilon^{-1}$, where $\mathbf{\Sigma}_\epsilon$ is the covariance matrix of the residuals. The corresponding estimate of $\hat{\mathbf{c}}$ is

$$\hat{\mathbf{c}} = (\mathbf{\Phi}'\mathbf{W}\mathbf{\Phi})^{-1}\mathbf{\Phi}'\mathbf{W}\mathbf{X}.$$

- Locally Weighted Least Squares Smoothing: Here, the fitting error is measured locally, so that observations closer to a point t have more influence on the function estimate at that point. The objective function to minimize is:

$$LWLS(X|c) = \sum_{i=1}^{n} w_i(t) \left(X(t_i) - \sum_{k=1}^{K} c_k \phi_k(t_i) \right)^2,$$

where the weights $w_i(t)$ are typically defined as

$$w_i(t) = \kappa \left(\frac{t_i - t}{h} \right),$$

with $\kappa(\cdot)$ being a kernel function and $h > 0$ a smoothing parameter.

Some commonly used kernel functions are:

  - Uniform kernel: $\kappa(u) = 0.5$ for $|u| \leq 1$, 0 otherwise,
  - Quadratic (Epanechnikov) kernel: $\kappa(u) = 0.75(1 - u^2)$ for $|u| \leq 1$, 0 otherwise,
  - Gaussian kernel: $\kappa(u) = (2\pi)^{-1/2} e^{-u^2/2}$ for all $u \in \mathbb{R}$.

The matrix form solution that minimizes $LWLS(X|c)$ is

$$\hat{\mathbf{c}}(t) = \left(\mathbf{\Phi}'\mathbf{W}(t)\mathbf{\Phi}\right)^{-1} \mathbf{\Phi}'\mathbf{W}(t)\mathbf{X},$$

where $\mathbf{W}(t)$ is the diagonal weight matrix with entries $w_i(t)$.

- Penalized Least Squares Smoothing: To avoid overfitting and obtain a smoother estimator, a penalty term can be introduced to control the roughness of the fitted function. Instead of penalizing the coefficients directly, the variability of the second derivative of the function is penalized.

  The penalized loss function is:

$$S_{\text{pen}}(X|\mathbf{c}) = \sum_{i=1}^{n} \left(X(t_i) - \sum_{k=1}^{K} c_k \phi_k(t_i) \right)^2 + \lambda J(\mathbf{c}),$$

  where the first term measures the squared error between the observations $X(t_i)$ and the prediction, and the second term $\lambda J(\mathbf{c})$ penalizes combinations of the basis functions that are not smooth enough. The parameter $\lambda > 0$ controls the intensity of the penalty.

  The penalty functional is:

$$J(\mathbf{c}) = \int \left(\sum_{k=1}^{K} c_k \phi_k''(t) \right)^2 dt,$$

  i.e., the integral of the square of the second derivative of the linear combination of basis functions.

  Expanding this expression, we can write:

$$J(\mathbf{c}) = \sum_{k=1}^{K} \sum_{l=1}^{K} c_k c_l \int \phi_k''(t) \phi_l''(t)\, dt,$$

  which in matrix form is:

$$J(\mathbf{c}) = \mathbf{c}^\top \mathbf{P} \mathbf{c},$$

  where the penalty matrix $\mathbf{P}$ has entries:

$$P_{kl} = \int \phi_k''(t) \phi_l''(t)\, dt, \quad 1 \le k, l \le K.$$

  Expressing the model in matrix form:

$$\mathbf{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_K \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} X(t_1) \\ X(t_2) \\ \vdots \\ X(t_n) \end{pmatrix}, \quad \mathbf{\Phi} = \begin{pmatrix} \phi_1(t_1) & \phi_2(t_1) & \dots & \phi_K(t_1) \\ \phi_1(t_2) & \phi_2(t_2) & \dots & \phi_K(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(t_n) & \phi_2(t_n) & \dots & \phi_K(t_n) \end{pmatrix}.$$

  Thus, the penalized loss function in matrix form is:

$$S_{\text{pen}}(X|\mathbf{c}) = \|\mathbf{X} - \mathbf{\Phi}\mathbf{c}\|^2 + \lambda\, \mathbf{c}^\top \mathbf{P} \mathbf{c}.$$

To find $\hat{\mathbf{c}}$, differentiate with respect to $\mathbf{c}$ and set to zero:

$$\frac{\partial}{\partial \mathbf{c}} \left(\|\mathbf{X} - \mathbf{\Phi c}\|^2 + \lambda \, \mathbf{c}^\top \mathbf{P c} \right) = 0,$$

leading to the system:

$$(\mathbf{\Phi}'\mathbf{\Phi} + \lambda \mathbf{P})\mathbf{c} = \mathbf{\Phi}'\mathbf{X}.$$

Finally, the estimated coefficients are:

$$\hat{\mathbf{c}} = (\mathbf{\Phi}'\mathbf{\Phi} + \lambda \mathbf{P})^{-1}\mathbf{\Phi}'\mathbf{X}.$$

- Locally Penalized Least Squares Smoothing: This approach aims to control model complexity by penalizing coefficients locally. It penalizes coefficients depending on their location or some specific property of the model, which is useful when certain features of the data may vary in different regions of the design space. The locally regularized loss function is:

$$S_{\text{ploc}}(X|\mathbf{c}, t) = \sum_{i=1}^{n} w_i(t) \left(X(t_i) - \sum_{k=1}^{K} c_k \phi_k(t_i) \right)^2 + \lambda J(\mathbf{c}),$$

where $w_i(t) = \kappa\left(\frac{t_i - t}{h}\right)$ is the weight assigned to observation $X(t_i)$ based on its proximity to t, $\lambda > 0$ is the smoothing parameter, and $J(\mathbf{c})$ measures the roughness of the linear combination of basis functions.

The penalty function is:

$$J(\mathbf{c}) = \int \left(\sum_{k=1}^{K} c_k \phi_k''(t) \right)^2 dt,$$

which can be written in matrix form as:

$$J(\mathbf{c}) = \mathbf{c}^\top \mathbf{P c},$$

where the matrix $\mathbf{P}$ has entries $P_{kl} = \int \phi_k''(t)\phi_l''(t)\, dt$, $1 \le k, l \le K$. Define $\mathbf{c} = (c_1, c_2, \ldots, c_K)'$, $\mathbf{X} = (X(t_1), X(t_2), \ldots, X(t_n))'$, $\mathbf{\Phi}$ as the design matrix with rows given by the basis evaluations at t_i, and $\mathbf{W}(t)$ as the diagonal weight matrix with entries $w_i(t)$.

The loss function in matrix form is:

$$S_{\text{plocal}}(X|\mathbf{c}, t) = (\mathbf{X} - \mathbf{\Phi c})'\mathbf{W}(t)(\mathbf{X} - \mathbf{\Phi c}) + \lambda \mathbf{c}'\mathbf{P c}.$$

To minimize $S_{\text{plocal}}(X|\mathbf{c}, t)$, differentiate with respect to $\mathbf{c}$ and set to zero:

$$-2\mathbf{\Phi}'\mathbf{W}(t)(\mathbf{X} - \mathbf{\Phi c}) + 2\lambda \mathbf{P c} = 0,$$

leading to:

$$(\mathbf{\Phi}'\mathbf{W}(t)\mathbf{\Phi} + \lambda \mathbf{P})\mathbf{c} = \mathbf{\Phi}'\mathbf{W}(t)\mathbf{X}.$$

Finally, the estimated coefficients are:

$$\hat{\mathbf{c}}(t) = (\mathbf{\Phi}'\mathbf{W}(t)\mathbf{\Phi} + \lambda\mathbf{P})^{-1}\mathbf{\Phi}'\mathbf{W}(t)\mathbf{X}.$$

Note that in this case, the coefficients depend on the value $t \in \mathcal{T}$ of $X(t)$ being approximated.

As we can see, to obtain a functional datum from observations, we must make two choices: the basis to use and the smoothing method to apply in order to determine the coefficients.

1.1.3 Alternatives

There are other ways to fit a smooth function to a set of observations $\{X(t_i), t_i\}_{i\in I}$. These methods, instead of assuming that the function can be expressed as a weighted sum of a set of basis functions, compute the smoothing function directly through a fitting procedure. The most common methods are:

- Kernel Smoothing: This can only be applied in the particular case where the functional data $\hat{X}(t)$ is a scalar function, i.e., $\hat{X}(t) : \mathbb{R} \mapsto \mathbb{R}$. This method is a special type of weighted moving average, where the weights are defined by the kernel, giving more weight to closer values. Given a dataset $\{X(t_i), t_i\}_{i\in S}$, the smoothed function is estimated as:

$$\hat{X}(t) = \frac{\sum_{i\in I} K_h^i(t) X(t_i)}{\sum_{i\in I} K_h^i(t)}$$

  where $K_h^i(t)$ is a kernel defined by

$$K_h^i(t) = \kappa\left(\frac{t - t_i}{h}\right),$$

  with:

  - h the smoothing parameter,
  - $\kappa(\cdot)$ typically a positive function on the real numbers, whose value decreases as the distance between t_i and t increases.

  ——— The most popular kernels used for smoothing include the parabolic, tricube, and Gaussian kernels.

- Local Polynomial Regression: This is a nonparametric smoothing method that fits a low-degree polynomial in a neighborhood around the point to be estimated. Given a dataset $\{X(t_i), t_i\}_{i\in S}$ and a point t_0 at which we want to estimate the value, we calculate $X(t_0)$ using the data near t_0 by performing a Taylor expansion of X centered at t_0:

$$X(t) \approx \beta_0 + \beta_1(t - t_0) + \beta_2(t - t_0)^2 + \cdots + \beta_p(t - t_0)^p.$$

Thus, it suffices to compute the coefficients β_i for all $i \in \{0, \ldots, p\}$; these coefficients are obtained by minimizing the following weighted sum of squares:

$$\sum_{i \in I} K_h^i(t_0) \left(X(t_i) - \sum_{j=0}^{p} \beta_j (t_i - t_0)^j \right)^2,$$

where $K_h^i(t_0) = K\left(\frac{t_i - t_0}{h}\right)$ is a weight assigned to each observation, typically with $K(u) = (1 - |u|^3)^3$. Once the coefficients $\beta_0, \ldots, \beta_p$ are estimated via weighted least squares, the value of the function at t_0 is simply

$$X(t_0) = \hat{\beta}_0.$$

Note that this is a local method, meaning it allows approximation of the true function only at specific points. This method is useful when we are interested in approximating only certain values of the true function.

Once the smoothed functions are obtained, they are treated as functional observations, allowing the application of functional data theory and methods. On the other hand, the approximation error propagates through the subsequent analyses.

1.2 Outliers

Functional data analysis often begins with a descriptive study of the data, which reduces to visualizing the data and detecting outliers.

In the case of discrete observations, outliers are typically identified using a box plot. For functional data, it is necessary to define a notion of outlyingness or centrality, which can be done using so-called functional data depths.

The functional depth of a function is defined as the proportion of times the function lies between any two other functions. From these proportions, a "functional box plot" can be constructed.

Formally, given a set of functional data $\{X_i\}_{i \in I}$ with a common domain $\mathcal{T}$, the functional depth of observation $X_k(t)$ is defined as $\#\{(i, j) \text{ such that } X_i(t) < X_k(t) < X_j(t)\ \forall t \in \mathcal{T}\}$, where $i < j$.

Thus, the sample median is the function with the greatest functional depth, while the 50% central region corresponds to the 50% of curves with the highest functional depth.

Example 1. Below, we present an example through a graphical representation based on fertility data of Australian women aged 15 to 49 between 1921 and 2006. These data are available in the `rainbow` R package.

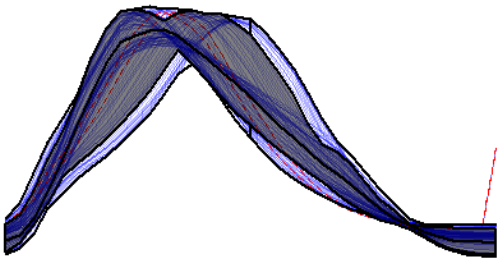

Figure 1.1: Functional dataset represented by the set of blue lines; the gray area represents the 50% central region.

1.3 Phase and Amplitude Variation

Functional data may exhibit two types of variation: phase variation and amplitude variation. Phase variation refers to differences in the location of certain events or patterns within the domain of X, while amplitude variation refers to differences in the magnitude or intensity of the function at different points in the domain, regardless of position. In the common case where $X : \mathcal{T} \subset \mathbb{R} \mapsto \mathbb{R}$, phase variation corresponds to changes along the x-axis, and amplitude variation corresponds to changes along the y-axis.

Ignoring phase variation in functional analysis may confound the effects of phase and amplitude variation, leading to less precise results.

There are two main techniques to reduce variation in functional data. Registration is one, aiming to remove phase variation and focus solely on amplitude variation. This process aligns the functions so that their main features are matched.

Suppose we have a set of functions $\{X_i(t)\}_{i\in I}$. To align these functions, a transformation $\gamma_i : \mathcal{T} \to \mathcal{T}$ is introduced for each $X_i(t)$, aligning $X_i(t)$ with the mean function $\mu(t)$, defined as the mean of $\{X_i(t)\}_{i\in I}$. We then study the set $\{(\gamma_i \circ X_i)(t)\}_{i\in I}$.

Generally, γ_i is chosen to align key features of the function, such as peaks, valleys, and other landmarks. More sophisticated approaches consider pairs of functions $X_i(t)$ and $X_j(t)$ and seek a transformation γ_{ij} that minimizes the L^2-norm distance between them:

$$\inf_{\gamma} \|X_i \circ \gamma_{ij} - X_j\|_{L^2}^2 = \inf_{\gamma} \int_T (X_i(\gamma_{ij}(t)) - X_j(t))^2 dt.$$

The second approach separates phase and amplitude variation to analyze each independently. This method is less commonly used.

1.4 Functional Principal Component Analysis

The main goal of this section is to show a method for reducing the dimensionality of functional data by finding principal components, which represent the main modes of variation of the function. This is achieved using the Karhunen-Loève theorem, which allows us to express a random function $X(t)$ as an infinite series of orthonormal basis functions:

$$X(t) = \mu(t) + \sum_{m=1}^{\infty} \xi_m \phi_m(t), \quad t \in \mathcal{T},$$

where $\mu(t)$ is the mean of $X(t)$ and $\phi_m(t)$ are orthonormal eigenfunctions of the covariance operator obtained via Mercer's theorem [3]:

$$\mathrm{Cov}(X(s), X(t)) = \sum_{m=1}^{\infty} \nu_m \phi_m(s) \phi_m(t),$$

with decreasing eigenvalues $\nu_1 \geq \nu_2 \geq \cdots \geq 0$, and ξ_m are uncorrelated random variables with mean zero and variance ν_m. The orthonormality condition is $\int_{\mathcal{T}} \phi_m(t)\phi_k(t)dt = 1$ if $k = m$ and 0 otherwise.

To approximate $X(t)$ using M principal components, it suffices to truncate the series at the first M terms.

Chapter 2

Clustering

Functional data clustering is an unsupervised learning method that aims to segment functional data into distinct groups based on certain patterns, in order to gain insight into the structure of the data. There are different approaches to clustering functional data:

- **Filtering Method:** These methods first approximate the curves using a basis of functions or principal components, and then perform clustering using the expansion coefficients, applying classical clustering methods.
- **Adaptive Method:** Similar to the previous method, dimensionality reduction is performed via basis functions or functional principal components, but clustering is done assuming a certain distribution for the expansion coefficients.
- **Distance-based Methods:** These replicate the techniques used for clustering multivariate observations, applying clustering algorithms based on distances specifically defined for functional data.

2.1 Filtering Method

This method consists of reducing the dimension of functional data to a finite dimension, either using functional principal components or a basis of functions adapted to the characteristics of the data. Once the dimension is reduced, classical clustering methods for discrete observations, such as K-means or hierarchical clustering, are applied. We illustrate this method using a B-spline basis for dimensionality reduction and K-means for clustering. Given a set of functions $(X_i(t))_{i\in I}$, represented using a B-spline basis $\{\phi_p(t)\}_p$, we have:

$$X_i(t) = \sum_{j=1}^{N} \beta_{ij}\phi_j(t).$$

To estimate the coefficients $\{\beta_{ij}\}_{i\in I,\, j=1,\dots,N}$, we solve the following linear systems for each $i \in I$:

$$\begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_N(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_N(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_m) & \phi_2(x_m) & \dots & \phi_N(x_m) \end{bmatrix} \begin{bmatrix} \beta_{i1} \\ \beta_{i2} \\ \vdots \\ \beta_{iN} \end{bmatrix} = \begin{bmatrix} X(x_1) \\ X(x_2) \\ \vdots \\ X(x_m) \end{bmatrix}.$$

We then classify the N-dimensional vectors of coefficients $\{\beta_i = (\beta_{i1}, \dots, \beta_{iN})\}_{i \in I}$ using the K-means method.

The goal of the algorithm is to partition the points into K groups $\{C_1, C_2, \dots, C_K\}$ such that the sum of squared distances between the coefficients in each group and their respective means is minimized. The procedure is as follows:

Choose the number of groups K and initialize the group means randomly: $\{\mu_1, \mu_2, \dots, \mu_K\}$.

For each data point β_i, assign it to the group C_k whose mean μ_k is closest to β_i:

$$C_k = \{X_i(t) : \|\beta_i - \mu_k\|^2 \leq \|\beta_i - \mu_j\|^2, \forall j = 1, 2, \dots, K\}.$$

After assignment, update the means:

$$\mu_k = \frac{1}{|C_k|} \sum_{\beta_i \in C_k} \beta_i,$$

where $|C_k|$ is the number of points in group C_k.

Repeat the assignment and mean update steps iteratively until the group means stabilize. The objective of K-means is to produce groups as homogeneous as possible by minimizing the within-cluster sum of squares (WCSS):

$$WCSS = \sum_{k=1}^{K} \sum_{\beta_i \in C_k} \|\beta_i - \mu_k\|^2.$$

Lower $WCSS$ values indicate lower within-group variability.

2.2 Adaptive Method

In filtering methods, curves are approximated using a finite basis and then identified with their expansion coefficients. In adaptive methods, the coefficients are treated as random variables with a probability distribution specific to the group to which they belong. These models depend on the choice of representation method (basis or functional principal components) and the probabilistic model of the coefficients. Given a set of data $\{X_i(t)\}_{i \in I}$, we can represent them as

$$X_i(t) \approx \sum_{j=1}^{P} \beta_{ij} \phi_j(t),$$

where $\{\phi_j(t)\}_j$ is a set of basis functions. Denote $\beta_i = (\beta_{i1}, \dots, \beta_{iP})$.

Alternatively, we can represent the data as

$$X_i(t) \approx \bar{X}(t) + \sum_{j=1}^{P} \xi_{ij}\psi_j(t),$$

where $\bar{X}(t)$ is the mean function and $\psi_j(t)$ are the eigenfunctions from the covariance decomposition.

Once the number of groups K is fixed, we assume that each group follows a predetermined distribution. For example, we may assume Gaussian distributions. Each observation $X_i(t)$ is assigned to a group k based on the probability that β_i belongs to the group's distribution. Using Bayes' theorem:

$$P(Z_i = k \mid \beta_i) = \frac{\pi_k \mathcal{N}(\beta_i \mid \mu_k, \Sigma_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(\beta_i \mid \mu_l, \Sigma_l)},$$

where Z_i is the group assignment for observation i, π_k is the prior probability of group k, and $\mathcal{N}(\beta_i \mid \mu_k, \Sigma_k)$ is the Gaussian density of β_i under group k. Parameters π_k, μ_k, and Σ_k are estimated using the EM algorithm.

Initialize parameters $\pi_k^{(0)}$, $\mu_k^{(0)}$, $\Sigma_k^{(0)}$, and compute the posterior probability at iteration s:

$$P(Z_i = k \mid \beta_i)^{(s)} = \frac{\pi_k^{(s)} \mathcal{N}(\beta_i \mid \mu_k^{(s)}, \Sigma_k^{(s)})}{\sum_{l=1}^{K} \pi_l^{(s)} \mathcal{N}(\beta_i \mid \mu_l^{(s)}, \Sigma_l^{(s)})}.$$

Update the parameters:

$$\pi_k^{(s+1)} = \frac{1}{n} \sum_{i=1}^{n} P(Z_i = k \mid \beta_i)^{(s)},$$

$$\mu_k^{(s+1)} = \frac{\sum_{i=1}^{n} P(Z_i = k \mid \beta_i)^{(s)} \beta_i}{\sum_{i=1}^{n} P(Z_i = k \mid \beta_i)^{(s)}},$$

$$\Sigma_k^{(s+1)} = \frac{\sum_{i=1}^{n} P(Z_i = k \mid \beta_i)^{(s)} (\beta_i - \mu_k^{(s)})(\beta_i - \mu_k^{(s)})^\top}{\sum_{i=1}^{n} P(Z_i = k \mid \beta_i)^{(s)}}.$$

Iterate until convergence or stability criteria are met. Finally, assign each observation β_i to the group maximizing the posterior probability.

The Bayesian Information Criterion (BIC) is recommended for model selection:

$$BIC = -2 \log L + p \log N,$$

where L is the likelihood, p is the number of estimated parameters, and N is the total number of observations:

$$L = \prod_{i=1}^{n} \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\beta_i \mid \mu_k, \Sigma_k) \right).$$

Lower BIC indicates a better balance between fit and model simplicity.

2.3 Distance-Based Methods

These methods apply clustering algorithms developed for multivariate observations to functional data. A distance between functions must be defined to represent the proximity between two functions. Consider the family of distances:

$$d_l(X_i(t), X_j(t)) = \left(\int_{\mathcal{T}} (X_i^{(l)}(t) - X_j^{(l)}(t))^2 \, dt \right)^{1/2},$$

for $l \in \mathbb{N}$. The choice of l depends on the clustering algorithm: for hierarchical clustering $l = 2$ is common, while for K-means a combination of $l = 0, 1$ such as $(d_0^2 + d_1^2)^{1/2}$ is used.

The most common hierarchical clustering method is divisive, starting with all data in a single group and iteratively partitioning. The process faces two challenges: how to partition and when to partition.

To decide, a heterogeneity index is defined. Let S be a group and $\mathcal{M}_{1,S}(t)$, $\mathcal{M}_{2,S}(t)$, $\mathcal{M}_{3,S}(t)$ be central measures (mean, median, mode). Define

$$HI(S) = \frac{d(\mathcal{M}_{i,S}(t), \mathcal{M}_{j,S}(t))}{d(\mathcal{M}_{i,S}(t), 0) + d(\mathcal{M}_{j,S}(t), 0)},$$

with $i, j \in \{1, 2, 3\}, i \neq j$, and $d(\cdot, \cdot)$ the Euclidean distance. For robustness, consider L subgroups $S^{(l)} \subset S$ of equal size:

$$SHI(S) = \frac{1}{L} \sum_{l=1}^{L} HI(S^{(l)}).$$

For a partition $S_1, \ldots, S_K$ of S:

$$PHI(S_1, \ldots, S_K) = \frac{1}{|S|} \sum_{k=1}^{K} |S_k| \times SHI(S),$$

and the partition criterion is

$$SC(S; S_1, \ldots, S_K) = SHI(S) - PHI(S; S_1, \ldots, S_K).$$

If $SC > 0$, split S; if $SC < 0$, do not split.

To perform the split, a granularity parameter h is introduced. Define

$$p_h(i) = \hat{P}(X(t) \in B(X_i(t), h)) = \frac{1}{n} \#\{j \in J \,|\, d(X_i(t), X_j(t)) < h\},$$

where $B(X_0(t), h) = \{X(t) \in L^2 | d(X(t), X_0(t)) < h\}$. Estimating the density f_h of $\{p_h(i)\}_{i \in J}$, modes correspond to distinct groups.

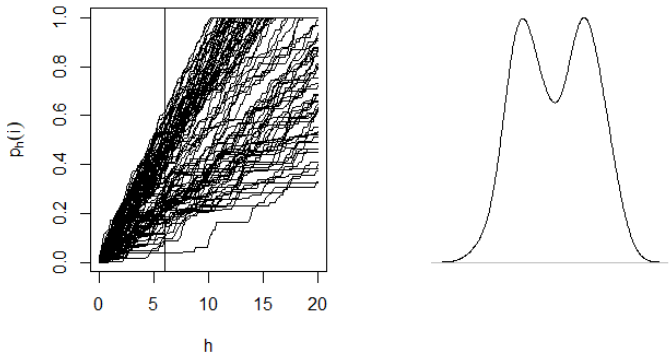

Figure 2.1: Left: evolution of $p_h(i)$ values for different h. Right: density of $\{p_h(i)\}_{i\in J}$ when $h = 6$.

Suppose that $f_h(\cdot)$ has K modes, and let $f_h(m_1), \ldots, f_h(m_K)$ be the corresponding $K-1$ local minima of $f_h(\cdot)$. Assuming that $m_1 < \cdots < m_K$, we can construct

$$\mathcal{I}_j = \{i \in J,\, m_{j-1} < \hat{p}_h(i) \leq m_j\},$$

where $m_{0,S} = 0$ and $m_{K,S} = 1$. Additionally, we have

$$\{1, \ldots, n\} = \bigcup_{k=1}^{K} \mathcal{I}_k \quad \text{and} \quad \forall k \neq k',\, \mathcal{I}_k \cap \mathcal{I}_{k'} = \emptyset.$$

This allows us to obtain a partition of $\{X_i(t)\}_{i\in J}$ as

$$C_k = \{X_i(t),\, i \in \mathcal{I}_k\}, \quad k = 1, \ldots, K.$$

Finally, we need to estimate the value of h that exhibits the greatest heterogeneity among the functions. The optimal value is usually obtained by minimizing the entropy of $f_h(\cdot)$ over a range of possible values H:

$$h_{opt} = \arg_{h\in H} \inf \int_0^1 f_h(t) \log(f_h(t))\, dt.$$

Therefore, once $f_h(\cdot)$ is estimated from $\{\hat{p}_h(i)\}_{i\in I}$ and taking a finite subset of H, denoted $\hat{H}$, we have

$$\hat{h}_{opt} = \arg_{h\in \hat{H}} \inf \int_0^1 \hat{f}_h(t) \log(\hat{f}_h(t))\, dt.$$

$f_h(t)$ is typically estimated using a symmetric kernel:

$$\hat{f}_h(\cdot) = \frac{1}{nb} \sum_{i=1}^{n} K(b^{-1}(t - p_h(i))).$$

In summary, the method works as follows: first, we consider a group formed by the set of all observations $S = \{X_i(t)\}_{i\in I}$, take a random partition of the subgroup $\{S_k\}_{k\in K}$, and calculate the coefficient $SC(S; S_1, \ldots, S_K)$. If it is negative,

the method stops; otherwise, we proceed to study the most efficient way to perform the partition. We start by selecting an interval H of possible h values (usually $H = [0, 100]$), then discretize the interval to obtain a finite set of candidate values, and for each value, estimate $\hat{f}_h(\cdot)$ to obtain $\hat{h}_{opt}$. Once the optimal bandwidth is determined, we calculate $\hat{p}_{\hat{h}_{opt}}(i)$, extract the modes of the density function $f_{\hat{h}_{opt}}(\cdot)$, and calculate the sets $\{\mathcal{I}_k\}_{k\in\{1\dots K\}}$, which allows us to obtain a partition of S into sets $\{C_k\}_{k\in\{1\dots K\}}$. Next, for each set C_k, the process is repeated: we consider a random partition of C_k, $S_{C_{1,k}}, \dots, S_{C_{n_k,k}}$, calculate the coefficient $SC(C_k; S_{C_{1,k}}, \dots, S_{C_{n_k,k}})$, and continue in the same manner as before.

Another method consists of applying the K-means algorithm to $\{X_i(t)\}$, but unlike the algorithm used in 2.1, the data dimensionality is not reduced; the algorithm is applied directly to the functions using the distances d_0. The algorithm has two iterative steps: a group assignment step, where each function is assigned to the closest group based on d_0, and a mean calculation step. Moreover, the functional K-means algorithm requires the number of groups K as input in advance and the selection of K functional samples as the initial set of group means, denoted $\{\mu_1(t), \dots, \mu_K(t)\}$. Each function $X_i(t)$ is initially assigned to the functional mean that minimizes the distance d_0, i.e.,

$$k_i = \arg\min_{j=1\dots K} d_0(X_i(t), \mu_j(t)),$$

where $k_i^{(s)}$ represents the assignment of function $X_i(t)$ to group k. Therefore, we can define each group as

$$C_k = \{X_i(t) : d(X_i(t), \mu_k(t)) \leq d(X_i(t), \mu_j(t)),\ \forall j = 1, 2, \dots, K\}.$$

After assigning all points to their respective groups, the means are updated:

$$\mu_j(t) = \frac{1}{|C_j|} \sum_{X_i(t)\in C_j} X_i(t).$$

The algorithm is repeated until the group means no longer change significantly. The objective of the K-means algorithm is to minimize the function

$$\sum_{k=1}^{K} \sum_{X_i(t)\in C_k} d(X_i(t), \mu_k(t)).$$

We conclude the descriptive study of the functional data and proceed to study the different regression methods that can be applied to a set of functional data.

Chapter 3

Functional Regression

In this chapter, we study how to perform inference on functional data. For a regression to be considered functional, at least one of the variables (dependent or independent) must be a function. Depending on the role of functional data in model construction, three types of regressions can be distinguished: scalar-on-function regression, where the independent variable is functional but the dependent variable is not; function-on-scalar regression, where the dependent variable is functional but the independents are not; and function-on-function regression, where both types of variables are functions.

3.1 Scalar-on-function regression

Scalar-on-function regression is a particular case of regression in which at least one of the explanatory variables is functional, while the response variable is scalar. The most popular scalar-on-function regression model is the functional linear model.

3.1.1 Functional linear scalar-on-function regression

As an example, consider a functional linear model with a single regressor. Given a dataset $\{Y_i, X_i(t)\}_{i=1\dots N}$ where all variables $\{X_i(t)\}_{i=1\dots N}$ share a common domain $\mathcal{T}$, the model is constructed as

$$Y_i = \alpha + \int_{\mathcal{T}} X_i(t)\beta(t)\,dt + \epsilon_i, \quad i = 1, \dots, n,$$

where Y_i, $i = 1 \dots N$ is a scalar variable, $X_i(t)$ is a predictor function, $\beta(t)$ is the functional coefficient of the model, and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$.
We present three different methods to estimate the coefficient $\beta(t)$.
The most direct way to obtain $\beta(t)$ is to expand it using basis functions:

$$\beta(t) = \sum_{i=1}^{K} c_k \phi_k(t).$$

We can rewrite the linear model as

$$\int_{\mathcal{T}} \beta(t)X_i(t)dt = \sum_{k=1}^{K} c_k \int_{\mathcal{T}} \phi_k(t)X_i(t)dt = \sum_{k=1}^{K} x_{ik}c_k,$$

where $x_{ik} = \int_{\mathcal{T}} \phi_k(t)X_i(t)dt$. Thus, we can reduce the model to a multiple scalar linear regression:

$$\mathbf{Y} = \mathbf{X}\mathbf{c} + \epsilon,$$

where $\mathbf{Y} = (Y_1, \ldots, Y_N)$, $\mathbf{c} = (\alpha, c_1, \ldots, c_K)^\top$, and

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1K} \\ 1 & x_{21} & x_{22} & \cdots & x_{2K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{NK} \end{bmatrix}.$$

When $\mathbf{X}$ is nonsingular, the solution is $\hat{\mathbf{c}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{Y}$. Hence,

$$\hat{\beta}(t) = \sum_{i=1}^{K} \hat{c}_k\phi_k(t).$$

The choice of basis functions affects estimation accuracy. Typically, K is chosen larger than the number of observations of all $X_i(t)$ variables, and it is recommended to use the same basis as that used to smooth the functions $X_i(t)$.

Since $\epsilon \sim \mathcal{N}(0, \sigma^2)$, confidence intervals for $\beta(t)$ can be constructed:

$$\epsilon_i = Y_i - \hat{\alpha} - \sum_{i=1}^{K} \hat{c}_k\phi_k(t).$$

Thus, the variance of $\hat{c}_k$ corresponds to the k-th diagonal element of $\hat{\sigma}(X^\top X)^{-1}$, denoted $\hat{\sigma}_k^2$. The 95% confidence interval for $\beta(t)$ is

$$\sum_{i=1}^{K} \hat{c}_k\phi_k(t) \pm 1.96 \sum_{i=1}^{K} \hat{\sigma}_k^2\phi_k(t).$$

The value of K plays an important role in smoothing the estimator of $\beta(t)$; as K increases, its influence on smoothing decreases.

Another way to estimate $\beta(t)$ is via principal component analysis. As seen in Section 1.4, any random variable $X_i(t)$ in L^2 can be expressed as

$$X_i(t) = \mu_i(t) + \sum_{j=1}^{\infty} \xi_j v_j(t),$$

where $\mu_i(t)$ is the trend of $X_i(t)$ and $v_j(t)$ are the functional principal components, i.e., the eigenfunctions of the covariance operator, which can be estimated empirically and denoted by $\hat{v}_j(t)$. Without loss of generality, we can assume that all $X_i(t)$ share the same functional principal components.

Hence, we can approximate $X_i(t)$ as

$$X_i(t) \approx \hat{\mu}(t) + \sum_{j=1}^{P} \hat{\xi}_{ij}\hat{v}_j(t), \quad \hat{\xi}_{ij} = \int [X_i(t) - \hat{\mu}(t)]\hat{v}_j(t)dt,$$

where $\hat{\mu}(t) = \frac{1}{n}\sum_{i=1}^{n} \mu_i(t)$. Substituting into the model,

$$Y_i = \alpha + \int \beta(t) \left(\hat{\mu}(t) + \sum_{j=1}^{P} \hat{\xi}_{ij}\hat{v}_j(t) \right) dt + \epsilon_i = \beta_0 + \sum_{j=1}^{P} \hat{\xi}_{ij}\beta_j + \epsilon_i,$$

where

$$\beta_0 = \alpha + \int \beta(t)\hat{\mu}(t)dt, \quad \beta_j = \int \beta(t)\hat{v}_j(t)dt.$$

We again have a multiple scalar linear regression problem. Writing the model in matrix form, define the matrix Ξ as

$$\Xi = \begin{bmatrix} 1 & \hat{\xi}_{11} & \hat{\xi}_{12} & \dots & \hat{\xi}_{1K} \\ 1 & \hat{\xi}_{21} & \hat{\xi}_{22} & \dots & \hat{\xi}_{2K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \hat{\xi}_{N1} & \hat{\xi}_{N2} & \dots & \hat{\xi}_{NK} \end{bmatrix},$$

$$Y = [Y_0, Y_1, \dots, Y_P]^\top, \quad \beta' = [\beta_0, \beta_1, \dots, \beta_P]^\top.$$

The model can be expressed as $Y = \Xi\beta + \epsilon$. Let $\hat{\beta} = [\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta_P}]^\top$ denote the least squares estimator. The estimates of the functional regression parameters are

$$\hat{\beta}(t) = \sum_{j=1}^{P} \hat{\beta}_j\hat{v}_j(t), \quad \hat{\alpha} = \hat{\beta}_0 - \sum_{j=1}^{P} \hat{\beta}_j \int \hat{v}_j(t)\mu(t)dt.$$

This follows from noting that $\beta(t) = \sum_{j=1}^{P} \beta_j v_j(t)$ since $\beta(t)$ admits an expansion in the basis $\{v_j(t)\}_{j=1}^{\infty}$, so $\beta(t) = \sum_{j=1}^{P} c_j v_j(t)$. Then, $\beta_j = \int \sum_{i=1}^{P} c_i v_i(t) v_j(t)dt$, and by orthogonality of $\{v_j(t)\}_{j=1}^{\infty}$, we have $c_j = \beta_j$ for all $j = 1, \dots, \infty$.

The third approach to estimate $\beta(t)$ is roughness penalization, where the smoothness of $\hat{\beta}(t)$ depends on λ and $L(t)$:

$$P_\lambda(\alpha, \beta) = \sum_{i=1}^{N} \left\{ Y_i - \alpha - \int \beta(t)X_i(t)dt \right\}^2 + \lambda \int L(t)^2 dt.$$

Typically, $L(t) = \beta''(t)$. As in the first method, expanding $\beta(t)$ in a basis $\{\phi_k(t)\}_k$:

$$\beta(t) = \sum_{i=1}^{K} c_k \phi_k(t),$$

we obtain

$$P_\lambda(\alpha, \beta) = P_\lambda(\alpha, c_1, c_2, \dots, c_K)$$

$$= \sum_{i=1}^{N} \left(Y_i - \alpha - \sum_{k=1}^{K} x_{ik} c_k \right)^2 + \lambda \int \sum_{k=1}^{K} c_k \phi_k''(t)\, dt$$
$$= \sum_{i=1}^{N} \left(Y_i - \alpha - \sum_{k=1}^{K} x_{ik} c_k \right)^2 + \lambda \sum_{k,k'=1}^{K} c_k c_{k'} R_{kk'},$$

where

$$R_{kk'} = \int \phi_k''(t) \phi_{k'}''(t) dt.$$

The parameter vector $\mathbf{c} = [\alpha, c_1, c_2, \ldots, c_K]^\top$ that minimizes this expression is

$$\hat{\mathbf{c}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{R})^{-1} \mathbf{X}^\top \mathbf{Y},$$

where

$$\mathbf{R} = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & R_{11} & R_{12} & \ldots & R_{1K} \\ 0 & R_{21} & R_{22} & \ldots & R_{2K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & R_{K1} & R_{K2} & \ldots & R_{KK} \end{bmatrix}.$$

As discussed at the beginning of this section, we have studied the simplest linear scalar-on-function regression model. There exist several generalizations and extensions; for instance, given a scalar variable $\{Y_i\}_{i=1\ldots n}$, a set of functions $\{X_{i1}(t)\}_{i=1\ldots n}, \ldots, \{X_{ip}(t)\}_{i=1\ldots n}$ with domains $\mathcal{T}_1, \ldots, \mathcal{T}_p$, and a set of scalar variables $Z_{i1}, \ldots, Z_{iq}$, we can construct the model

$$Y_i = \alpha + \sum_{j=1}^{p} \int_{T_j} X_{ij}(t) \beta_j(t)\, dt + \sum_{r=1}^{q} Z_{ir} \gamma_r + \epsilon_i, \quad i = 1, \ldots, n,$$

which only has solutions in very particular cases (see [6]).

In summary, a functional linear model with a single regressor can be estimated by several methods. First, the coefficient $\beta(t)$ can be expanded in a function basis, resulting in a multiple linear regression model. Second, principal component analysis can be used, approximating $X_i(t)$ as a linear combination of its principal components, also reducing the model to linear regression. Finally, roughness penalization can be applied, where the smoothness of $\hat{\beta}(t)$ depends on a parameter λ and a smoothing operator $L(t)$. All these methods provide estimates of $\beta(t)$, but the choice of method and associated parameters, such as the number of components K or the penalty λ, affects the precision and smoothness of the estimation.

3.1.2 Nonlinear scalar-on-function regression

Let us start by studying one of the simplest nonlinear models, quadratic regression. Given a dataset $\{Y_i, X_i(t)\}_{i=1\ldots N}$, consider the model

$$Y_i = \alpha + \int_{\mathcal{T}} X_i(t) \beta(t)\, dt + \int_{\mathcal{T}} X_i(t) \gamma(t, s)\, dt + \epsilon_i, \quad i = 1, \ldots, n,$$

where Y_i, $i = 1 \dots N$ is a scalar variable, $X_i(t)$ is a predictor function, $\beta(t)$ is the functional coefficient, and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. As in the previous subsection, we consider the expansion

$$X_i(t) = \mu(t) + \sum_{j=1}^{\infty} \xi_{ij} v_j(t),$$

so we can approximate $\beta(t)$ and $\gamma(t, s)$ as

$$\hat{\beta}(t) = \sum_{j=1}^{K} \hat{\beta}_j \hat{v}_j(t), \quad \hat{\gamma}(t, s) = \sum_{j,l=1}^{K} \hat{\gamma}_{jl} \hat{v}_j(s) \hat{v}_l(t).$$

To approximate

$$\hat{\beta} = \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_P \end{bmatrix}, \quad \hat{\gamma} = \begin{bmatrix} \hat{\gamma}_{11} & \hat{\gamma}_{12} & \cdots & \hat{\gamma}_{1K} \\ \hat{\gamma}_{21} & \hat{\gamma}_{22} & \cdots & \hat{\gamma}_{2K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{\gamma}_{K1} & \hat{\gamma}_{K2} & \cdots & \hat{\gamma}_{KK} \end{bmatrix},$$

we reduce the functional model to a scalar model using the same method as in the previous section. First, we construct the model matrix, composed of two submatrices:

$$\Xi_{\hat{\beta}} = \begin{bmatrix} \hat{\xi}_{11} & \hat{\xi}_{12} & \cdots & \hat{\xi}_{1K} \\ \hat{\xi}_{21} & \hat{\xi}_{22} & \cdots & \hat{\xi}_{2K} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{\xi}_{N1} & \hat{\xi}_{N2} & \cdots & \hat{\xi}_{NK} \end{bmatrix},$$

and

$$\Xi_{\hat{\gamma}} = \begin{bmatrix} \hat{\xi}_{11}\hat{\xi}_{11} & \hat{\xi}_{11}\hat{\xi}_{12} & \cdots & \hat{\xi}_{1K}\hat{\xi}_{1K} \\ \hat{\xi}_{21}\hat{\xi}_{21} & \hat{\xi}_{21}\hat{\xi}_{22} & \cdots & \hat{\xi}_{2K}\hat{\xi}_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\xi}_{N1}\hat{\xi}_{N1} & \hat{\xi}_{N1}\hat{\xi}_{N2} & \cdots & \hat{\xi}_{NK}\hat{\xi}_{NK} \end{bmatrix}.$$

Thus, $\Xi = (\Xi_{\hat{\beta}} | \Xi_{\hat{\gamma}})$, which allows us to reduce the functional regression model to a linear scalar regression model $Y = \Xi\beta + \epsilon$. Solving by least squares, we obtain $(\hat{\beta}|\hat{\gamma}) = (\Xi^\top \Xi)^{-1} \Xi^\top Y$.

For higher-degree polynomial regression, the same procedure applies.

A more general model is the functional generalized additive model (FGAM), given by

$$Y_i = \alpha + \int f(X_i(t), t)\, dt + \epsilon_i,$$

where the function $f(x, t)$ is smooth. It is usually approximated as

$$f(x, t) \approx \sum_{i=1}^{I} \sum_{l=1}^{L} f_{il} B_i^*(x) B_l(t),$$

where $B_i^*(x)$ and $B_l(t)$ are spline bases. The coefficients are typically estimated using penalized least squares; the penalty term is added to avoid overfitting and to

obtain a smoother approximation of $f(x,t)$. The fitting procedure usually involves minimizing

$$\sum_{i=1}^{n}\left(Y_i-\alpha-\int_{\mathcal{T}_i}\int_{X_i(\mathcal{T}_i)}\sum_{i=1}^{I}\sum_{\ell=1}^{L}f_{i\ell}B_i(x)B_\ell(t)\,dxdt\right)^2+\lambda\int\left(\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial t^2}\right)^2 dx\,dt,$$

where the free parameter λ controls the smoothness of $f(x,t)$.
Another approach for estimating the model coefficients is nonparametric regression, i.e.,

$$Y_i=m(X_i)+\epsilon,$$

where $m: L^2\mapsto\mathbb{R}$, $X\mapsto m(X)=\alpha+\int f(t,X)\,dt+\epsilon$. To estimate $m(X)$ for $X\in L^2$, we define

$$\hat{m}(x)=\sum_{i=1}^{N}w_i(x)Y_i\quad\text{where}\quad\sum_{i=1}^{N}w_i(x)=1.$$

The estimation is based on the idea that the prediction of $m(X)$ should be the weighted average of observed values Y_i, where weights $w_i(x)$ give higher importance to Y_i corresponding to regressors close to $X(t)$. The weights are estimated as

$$w_i(x)=\frac{K\left(h^{-1}d(x,X_i)\right)}{\sum_{i=1}^{N}K\left(h^{-1}d(x,X_i)\right)},$$

where $d(\cdot,\cdot)$ is a distance metric between functions, typically the L^2 distance. The kernel $K(t):\mathbb{R}^+\mapsto\mathbb{R}$ is a decreasing function, with $K(t)$ approximately zero for large t. The bandwidth h determines which Y_i are effectively included: for small h, few Y_i are included; as h increases, more Y_i contribute to the estimate.

In summary, nonparametric and functional models aim to approximate the relationship between the dependent variable Y_i and the predictor functions $X_i(t)$ using flexible techniques such as functional bases, splines, and penalized smoothing. In FGAMs, splines are used to smooth $f(x,t)$, whereas in nonparametric approaches, weights based on the proximity of predictors to $X(t)$ are used to estimate $m(X)$. These models provide flexibility and adaptability without overfitting, and careful selection of smoothing parameters, such as h or λ, is crucial for accurate and robust estimation.

3.2 Function-on-scalar regression

Function-on-scalar regression is a regression model in which the response variable is a function, while the predictor variables are scalars. As in the previous section, we distinguish two cases: linear and nonlinear.

3.2.1 Linear function-on-scalar regression

The linear function-on-scalar regression model is of the form

$$Y_i(t)=\alpha(t)+\sum_{j=1}^{q}x_{ij}\beta_j(t)+\epsilon_i(t),\quad i=1,2,\ldots,N.$$

The functional regression parameters β_j are often called functional effects. Defining

$$Y(t) = \begin{bmatrix} Y_1(t) \\ Y_2(t) \\ \vdots \\ Y_N(t) \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1q} \\ 1 & x_{21} & x_{22} & \dots & x_{2q} \\ 1 & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \dots & x_{Nq} \end{bmatrix}, \quad \gamma(t) = \begin{bmatrix} \alpha(t) \\ \beta_1(t) \\ \beta_2(t) \\ \vdots \\ \beta_q(t) \end{bmatrix}, \quad \epsilon(t) = \begin{bmatrix} \epsilon_1(t) \\ \epsilon_2(t) \\ \vdots \\ \epsilon_N(t) \end{bmatrix},$$

we can rewrite the model as

$$Y(t) = X\gamma(t) + \epsilon(t).$$

The least squares estimator of the functional parameter $\gamma(t)$ is defined as the value that minimizes

$$\sum_{i=1}^{N} \left\| Y_i(t) - \sum_{k=1}^{q+1} x_{ik}\gamma_k(t) \right\|^2 dt = \sum_{i=1}^{N} e_i^2(\gamma, t)\, dt,$$

where

$$e_i(\boldsymbol{\gamma}, t)^2 = \left\| Y_i(t) - \sum_{k=1}^{q+1} x_{ik}\gamma_k(t) \right\|^2.$$

For each fixed t, the function that minimizes $\sum_{i=1}^{N} e_i^2(\gamma, t)$ is

$$\hat{\gamma}(t) = (\alpha(t), \beta_1(t), \dots, \beta_q(t)) = \left(X^\top X\right)^{-1} X^\top Y(t),$$

giving the approximation of the response variable at $t \in \mathcal{T}$:

$$\hat{Y}(t) = X\hat{\gamma}(t).$$

In summary, this generalizes the idea used in scalar-on-scalar regression by transforming the functional model into a multivariate model. In this case, the dependent variable is a function, so the model is solved locally for each $t \in \mathcal{T}$.

3.2.2 Nonlinear function-on-scalar regression

Nonlinear function-on-scalar regression models usually have very complex solutions. In this section, we consider one of the simplest models: an additive function-on-scalar regression.

$$Y_i(t) = \alpha(t) + \sum_{j=1}^{p} \gamma_j(x_{ij}, t) + \epsilon_i(t), \quad i = 1, \dots, n,\ t \in \mathcal{T}.$$

Here, γ_j is a nonlinear function. To estimate the model parameters, suppose that $\gamma_j(x_{ij}, t)$ can be expressed as

$$\gamma_j(x_{ij}, t) \approx \sum_{k=1}^{K_j} \beta_{jk} b_{jk}(x_{ij}) \phi_k(t),$$

where $b_{jk}(x_{ij})$ are basis functions modeling the nonlinear effect of x_{ij} and $\phi_k(t)$ are basis functions modeling smooth time-dependent variation. The model becomes

$$Y_i(t) = \alpha(t) + \sum_{j=1}^{p} \sum_{k=1}^{K_j} \beta_{jk}\, b_{jk}(x_{ij})\phi_k(t) + \epsilon_i(t).$$

The goal is to estimate the coefficients β_{jk}, typically using penalized ordinary least squares. To avoid overfitting, a penalty is imposed on the smoothness of $\gamma_j(x_{ij}, t)$, often by penalizing the second derivative of the basis function coefficients. The estimation problem becomes minimizing

$$\sum_{i=1}^{n} \int_{\mathcal{T}} \left(Y_i(t) - \alpha(t) + \sum_{j=1}^{p} \sum_{k=1}^{K_j} \beta_{jk}\, b_{jk}(x_{ij})\phi_k(t) \right)^2 dt + \lambda_j \int \left(\frac{\partial^2 \gamma_j(x_{ij}, t)}{\partial t^2} \right)^2 dt,$$

where λ_j is the smoothing parameter controlling the trade-off between model fit and smoothness. The derivation of the parameter estimates can be found in [4].

3.3 Function-on-function regression

In these models, both the dependent variable and all predictor variables are functions, which makes practical estimation challenging.

3.3.1 Linear function-on-function regression

We begin with the linear function-on-function regression model, also known as the concurrent model. If all variables have the same domain, the model is

$$Y_i(t) = \alpha(t) + \sum_{j=1}^{P} X_{ij}(t)\beta_j(t) + \epsilon_i(t), \quad i = 1 \dots n,\, t \in \mathcal{T}.$$

If the variables $X_{ij}(t)$ have different domains for different i, then

$$Y_i(t) = \alpha(t) + \sum_{j=1}^{P} \int_{\mathcal{S}_j} X_{ij}(s)\beta_j(t, s)\, ds + \epsilon_i(t), \quad i = 1 \dots n,\, t \in \mathcal{T},$$

where $\mathcal{S}_j$ denotes the domain of $X_{ij}(t)$ for all i.
To simplify notation, assume $\alpha(t) = 0$. We perform a basis expansion of all functions in the model:

- $Y_i(t) = \sum_{k=1}^{K} c_{ik}\phi_k(t), \quad \forall t \in \mathcal{T}$
- $X_{ij}(s) = \sum_{p=1}^{M} d_{ijp}\psi_{pj}(s), \quad \forall s \in \mathcal{S}_i$
- $\beta_j(t, s) = \sum_{p=1}^{M} \sum_{k=1}^{N} b_{jpk}\psi_{pj}(s)\phi_k(t), \quad \forall t \in \mathcal{T}, \forall s \in \mathcal{S}_j$

Where $\{\psi_l(s)\}$ and $\{\phi_n(t)\}$ are basis functions. Denoting

$$\mathbf{\Phi}(t) = \begin{pmatrix} \phi_1(t) \\ \phi_2(t) \\ \vdots \\ \phi_N(t) \end{pmatrix}, \quad \mathbf{\Psi}_m(t) = \begin{pmatrix} \psi_1(t) \\ \psi_2(t) \\ \vdots \\ \psi_N(t) \end{pmatrix}, \quad \mathbf{d}_{im} = \begin{pmatrix} d_{im1} \\ d_{im2} \\ \vdots \\ d_{imP} \end{pmatrix},$$

$$\mathbf{B}_j = \begin{bmatrix} b_{j11} & b_{j12} & \cdots & b_{j1N} \\ b_{j21} & b_{j22} & \cdots & b_{j2N} \\ \vdots & \vdots & \ddots & \vdots \\ b_{jM1} & b_{jM2} & \cdots & b_{jMN} \end{bmatrix}, \quad \mathbf{B} = \begin{pmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \vdots \\ \mathbf{B}_P \end{pmatrix}, \quad \mathbf{c}_i = \begin{pmatrix} c_{i1} \\ c_{i2} \\ \vdots \\ c_{iK_y} \end{pmatrix}.$$

The model can be rewritten as

$$\mathbf{c}_i\mathbf{\Phi}(t) = \sum_{m=1}^{M} \mathbf{d}_{im}\zeta_{\psi_m}\mathbf{B}_m\mathbf{\Phi}(t) + \epsilon_i(t) = z_i\mathbf{B}\mathbf{\Phi}(t) + \epsilon_i(t),$$

where $\zeta_{\psi_m} = \int_{\mathcal{S}} \mathbf{\Psi}_m(t)\mathbf{\Psi}_m^\top(t)dt$ is an $M \times M$ cross-product matrix, and $z_i = (\mathbf{d}_{i1}^\top\zeta_{\psi_1}, \mathbf{d}_{i2}^\top\zeta_{\psi_2}, \ldots, \mathbf{d}_{iM}^\top\zeta_{\psi_M})$. Denoting

$$\mathbf{C} = \begin{pmatrix} \mathbf{c}_1^\top \\ \mathbf{c}_2^\top \\ \vdots \\ \mathbf{c}_N^\top \end{pmatrix}, \quad \mathbf{Z} = \begin{pmatrix} z_1^\top \\ z_2^\top \\ \vdots \\ z_N^\top \end{pmatrix},$$

the model in matrix form is

$$\mathbf{C}\mathbf{\Phi}(t) = \mathbf{Z}\mathbf{B}\mathbf{\Phi}(t) + \mathbf{e}(t).$$

Assuming the error function $\boldsymbol{\epsilon}(t)$ can also be expanded in basis functions, $\boldsymbol{\epsilon}(t) = \mathbf{e}\mathbf{\Phi}(t)$ with $\mathbf{e} = (\mathbf{e}_1, \ldots, \mathbf{e}_N)^\top$, where $\mathbf{e}_i$ are i.i.d. random variables. By orthogonality of the basis functions, multiplying on the right by $\mathbf{\Phi}^\top(t)$ and integrating over $\mathcal{T}$ yields:

$$\mathbf{C} = \mathbf{Z}\mathbf{B} + \mathbf{e}.$$

The coefficients of $\mathbf{C}$ and $\mathbf{Z}$ come from the basis expansions and can be estimated parametrically, allowing the solution of the matrix equation and estimation of $\mathbf{B}$ as

$$\hat{\mathbf{B}} = (\mathbf{Z}^\top\mathbf{Z})^{-1}\mathbf{Z}^\top\mathbf{C},$$

giving the coefficients $\{b_{jpk}\}$ of the basis expansion of $\beta_j(t,s)$ and approximating the parametric function of the model.

3.3.2 Nonlinear function-on-function regression

The general formula for a nonlinear function-on-function regression model is

$$Y_i(t) = \alpha(t) + \sum_{j=1}^{p} \int_{S_j} F_j(X_{ij}(s), s, t)\, ds + \epsilon_i(t), \quad i = 1, \ldots, n,$$

where $F_j(X_{ij}(s), s, t)$ is a nonlinear function. Estimation of parameters in these models is beyond the scope of this work due to their complexity and the advanced techniques required; some proposed algorithms can be found in [11, 12].

Chapter 4

Study of European Bonds through Functional Data

In this chapter we will apply the theory developed in previous chapters to a set of European bond yield curves where $X_i(t)$ represents the yield of the set of bonds issued by European banks in month number t, where t varies from 1 to 350. In total we have 655 curves, each of which represents the yield of the bonds from a business day between 29/12/2006 and 24/07/2009. Therefore, the dataset is of the form $\{X_i(t_{ij}) \mid i = 1, \ldots, 655;\ j = 1, \ldots, 350\}$. The data come from `ECBYieldcurve` from the `fda` package. We will start the chapter by visualizing the data.

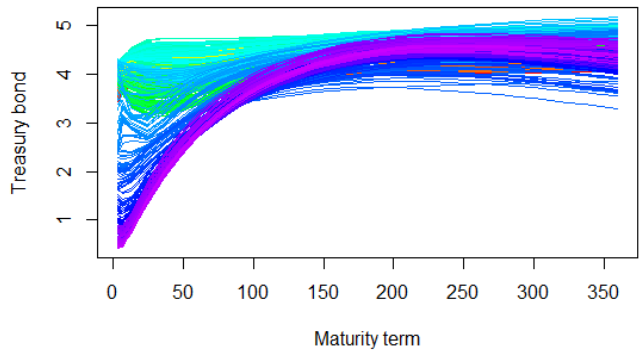

Figure 4.1: Graphical representation of the dataset `ECBYieldcurve`

On the Y-axis the bond return is represented in percentage, while on the X-axis the maturity date is observed, so each curve represents the yield of the set of bonds over different months. It should be noted that maturity dates are usually fixed, i.e., they occur in discrete time intervals such as 3 months, 12 months, 48 months, etc. Therefore, we will only have discrete observations of each curve; specifically, for each

i we have

$$\mathcal{T}_i = \left\{ \begin{array}{cccccccc} 3, & 6, & 12, & 24, & 36, & 48, & 60, & 72, \\ 84, & 96, & 108, & 120, & 132, & 144, & 156, & 168, \\ 180, & 192, & 204, & 216, & 228, & 240, & 252, & 264, \\ 276, & 288, & 300, & 312, & 324, & 336, & 348, & 360 \end{array} \right\}$$

We now study the distribution of the functional data, for which we will represent a functional box plot,

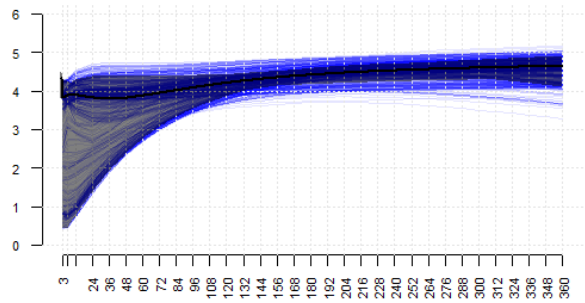

Figure 4.2: Representation of the dataset `ECBYieldcurve` together with a functional box-plot, the grey area represents the central 50% region

where the grey region represents the central functional region from 25% to 75% calculated from the proportions described in 1.2, and the black curve represents the most central function, based on the proportions mentioned in the same section.

4.1 Functional Data Approximation

As reflected in 1.1.1, when applying functional data analysis theory to real datasets, we lack continuous observations; therefore, it is necessary to apply some method that allows obtaining a smoothed function. Generally, it is assumed that the function can be expressed as a linear combination of basis functions, the basis being predefined, and then the coefficients of the linear combination are obtained through some smoothing method.

Below we show functions programmed in `R` to approximate the 655 functions using different bases and smoothing methods.

4.1.1 Splines

We start by using B-Splines as basis functions, applying different parametric estimation methods in `RStudio` to approximate the coefficients of the linear combination.

Estimation by Ordinary Least Squares (OLS):

The following code in `R` approximates the functions from a spline basis and with coefficients estimated by ordinary least squares:

```
b_splin_MCO <- function(i, num_pun) {

  x <- datos$x
  y <- datos$y[,i]

  modelo <- lm(y ~ bs(x, df = length(x) - 1))
  x_nuevo <- seq(min(x), max(x), length.out = num_pun)

  y_pred <- predict(modelo, newdata = data.frame(x = x_nuevo))
  plot(x, y, main = paste("B-Splines by OLS for function", i),
       col = "red", pch = 16, xlab = "Maturity", ylab = "Return")
  lines(x_new, y_pred, col = "blue", lwd = 2, type = "l")
}
```

We will plot a particular function $i = 200$, corresponding to the bond yields starting from 07/10/2007 over their maturity period. To evaluate the results, we take a sample of 1000 values within $\mathcal{T}_i$ and substitute them into the approximated function; the estimated points are connected by linear interpolation.

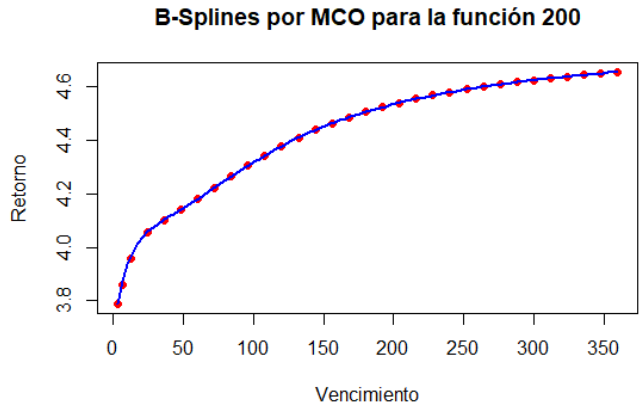

Figure 4.3: Approximation of the functions in the discrete dataset `ECBYieldcurve` using B-splines and OLS

A good fit is observed, although knowing only 32 discrete values of the function makes it difficult to obtain a completely precise method.

Estimation by Penalized Ordinary Least Squares (POLS):

Unlike the previous case, we add a penalization coefficient (automatically optimized by `gam` via cross-validation) that allows obtaining a smoother function.

```
b_splin_MCO <- function(i, num_pun) {

  x <- datos$x
```

```
  y <- datos$y[,i]

  df <- data.frame(x = datos$x, y = datos$y[, i])

  modelo_penalizado <- gam(y ~ s(x, bs = "ps"), data = df)

  x_nuevo <- seq(min(x), max(x), length.out = num_pun)

  y_pred <- predict(modelo_penalizado, newdata = data.frame(x = x_nuevo))
  plot(x, y, main = paste("B-Splines by Penalized OLS for function", i),
       col = "red", pch = 16, xlab = "Maturity", ylab = "Return")
  lines(x_new, y_pred, col = "blue", lwd = 2, type = "l")
}
```

We will plot the results in the same way as in the previous case.

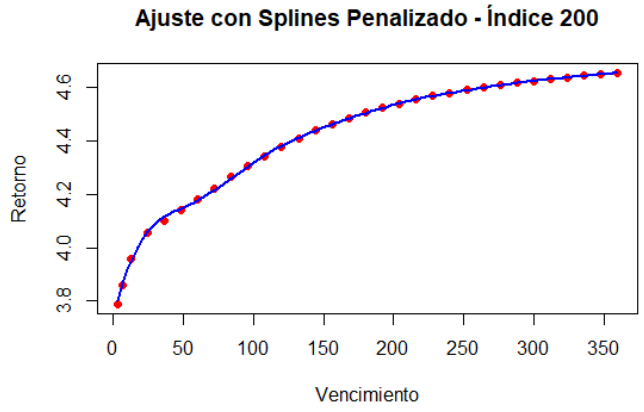

Figure 4.4: Approximation of the functions in the discrete dataset `ECBYieldcurve` using B-splines and Penalized OLS

No significant change is observed compared to the previous method, as most discrete observations do not present abrupt variations. Nevertheless, smoothing methods will continue to be applied for illustrative purposes.

Estimation by Locally Penalized OLS:

```
b_splines_MCO_penalizados_localizados <- function(i, num_points =
    100,window_size = 10) {
  x <- datos$x
  y <- datos$y[, i]
  n <- length(x)

  spl_basis <- bs(x, df = n - 1)

  calc_varianza_local <- function(x, y, window_size = 10) {
    n <- length(x)
    var_local <- rep(NA, n)
```

```
  for (j in 1:n) {
    indices <- which(abs(x - x[j]) <= window_size / n * diff(range(x)))
    var_local[j] <- var(y[indices], na.rm = TRUE)
  }

  return(var_local)
}

var_local <- calc_varianza_local(x, y, window_size)

var_local[var_local < 1e-4] <- 1e-4
weights <- 1 / var_local

fit <- lm.wfit(x = spl_basis, y = y, w = weights)

c_k <- fit$coefficients

x_new <- seq(min(x), max(x), length.out = num_points)
spl_basis_new <- predict(bs(x, df = n - 1), newx = x_new)

y_est_new <- spl_basis_new %*% c_k

plot(x, y, col = "red", pch = 16, main = paste("B-Splines Locally
    Penalized OLS - Index", i), xlab = "Maturity", ylab = "Return")
lines(x_new, y_est_new, col = "blue", lwd = 2)
}
```

We will plot the results in the same way as before.

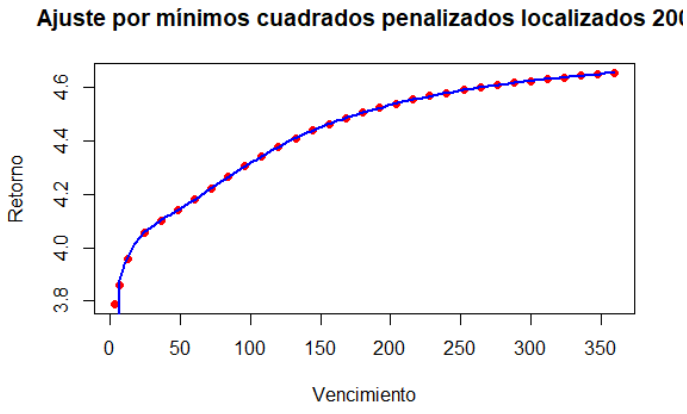

Figure 4.5: Approximation of the functions in the discrete dataset `ECBYieldcurve` using B-splines and Locally Penalized OLS

The fit is again quite good, showing some deviation in the first discrete observation. We continue the section by exploring another type of basis functions.

4.1.2 Wavelets

We move on to study another basis of functions, the Wavelets, described in 1.1.1. As in the previous section, we will apply three parametric estimation methods in `RStudio`.

Estimation by OLS:

```
wavelet_MCO <- function(i, num_pun , niveles ) {
  x <- datos$x
  y <- datos$y[, i]

  wavelet_ajustado <- dwt(y, n.levels = niveles)

  y_wavelet_rec <- idwt(wavelet_ajustado)

  x_new <- seq(min(x), max(x), length.out = num_pun)
  y_wavelet_new <- approx(x, y_wavelet_rec, xout = x_new)$y

  plot(x, y, col = "red", pch = 16, main = paste("OLS Fit - Index", i),
      xlab = "Maturity", ylab = "Return")
  lines(x_new, y_wavelet_new, col = "blue", lwd = 2)
}
```

We will plot a particular function $i = 200$ to evaluate the results, taking a sample of 1000 values within $\mathcal{T}_i$ and substituting them into the approximated function; the estimated points are connected by linear interpolation.

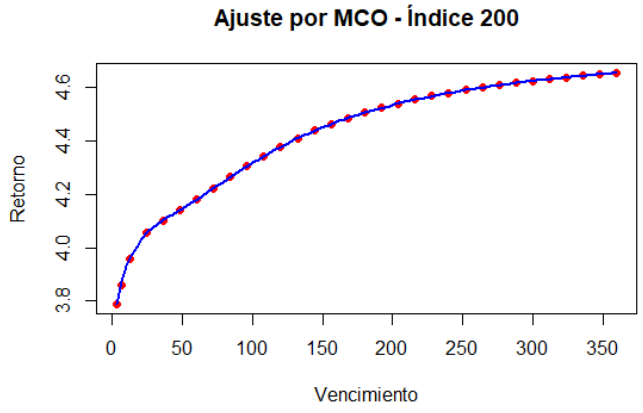

Figure 4.6: Approximation of the functions in the discrete dataset `ECBYieldcurve` using Wavelets and OLS

As with the B-Spline basis, the OLS method gives a good fit with wavelets.

Estimation by Penalized OLS:

Unlike the previous case, we add a penalization coefficient, determined empirically, that allows obtaining a smoother function.

```
wavelet_MCO_penalizado <- function(i, num_pun , numero_filtros , niveles)
    {
  x <- datos$x
  y <- datos$y[, i]

  wavelet_descomp <- wd(y, filter.number = numero_filtros, family =
      "DaubExPhase")

  nivel_wavelet <- threshold(wavelet_descomp, levels = niveles)

  y_wavelet_suave <- wr(nivel_wavelet)

  x_new <- seq(min(x), max(x), length.out = num_pun)
  y_wavelet_new <- approx(x, y_wavelet_suave, xout = x_new)$y

  plot(x, y, col = "red", pch = 16, main = paste("Penalized Wavelet Fit -
      Index", i), xlab = "Maturity", ylab = "Return")
  lines(x_new, y_wavelet_new, col = "blue", lwd = 2)
}
```

We will plot the results in the same way as before.

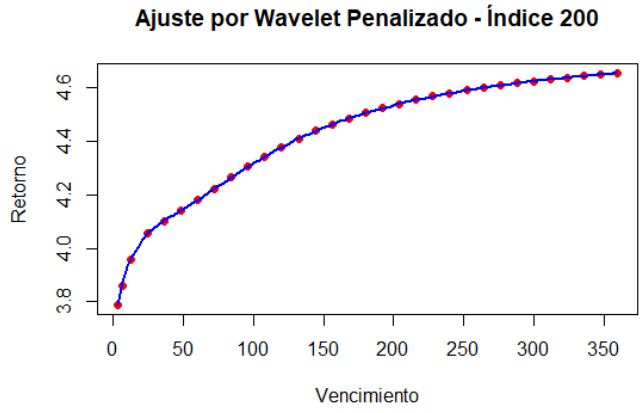

Figure 4.7: Approximation of the functions in the discrete dataset `ECBYieldcurve` using Penalized Wavelets

Note that although the fit is similar to the OLS method, this method is more robust against overfitting.

Estimation by Locally Penalized OLS:

```
wavelet_MCO_ponderada_local <- function(i, num_pun , niveles) {
  x <- datos$x
  y <- datos$y[, i]

  wavelet_fit <- dwt(y, n.levels = niveles)
```

```
  y_wavelet_rec <- idwt(wavelet_fit)

  x_new <- seq(min(x), max(x), length.out = num_pun)
  y_wavelet_rec_new <- approx(x, y_wavelet_rec, xout = x_new)$y

  calc_varianza_local <- function(x, y, ventana) {
    n <- length(x)
    var_local <- rep(NA, n)

    for (j in 1:n) {
      indices <- which(abs(x - x[j]) <= ventana / n * diff(range(x)))
      var_local[j] <- var(y[indices], na.rm = TRUE)
    }

    return(var_local)
  }

  var_local <- calc_varianza_local(x, y, 10)
  var_local[var_local < 1e-4] <- 1e-4
  pesos <- 1 / var_local

  fit <- lm.wfit(x = cbind(rep(1, length(x)), y_wavelet_rec), y = y, w =
      pesos)

  y_est <- cbind(rep(1, length(x)), y_wavelet_rec) %*% fit$coefficients
  y_est_new <- cbind(rep(1, length(x_new)), y_wavelet_rec_new) %*%
      fit$coefficients

  plot(x, y, col = "red", pch = 16, main = paste("Locally Penalized
      Wavelet Fit - Index", i) , xlab = "Maturity", ylab = "Return")
  lines(x_new, y_est_new, col = "blue", lwd = 2)
}
```

We will plot the results in the same way as before.

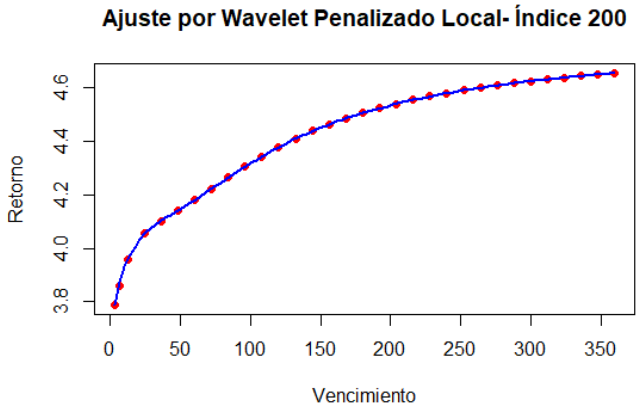

Figure 4.8: Approximation of the functions in the discrete dataset `ECBYieldcurve` using Locally Penalized Wavelets

As with the previous smoothing method, it is not necessary to apply this method when the discrete observations do not present evident roughness; nevertheless, it does not worsen the OLS estimation.

Next, we will explore other types of methods to smooth the functions.

4.1.3 Alternatives

We will apply the different methods shown in 1.1.3 that allow obtaining a smooth function directly.

Kernel Smoothing:

```
suavizado_nucleo <- function(i, num_pun) {
  x <- datos$x
  y <- datos$y[, i]

  bw <- bw.SJ(x)

  x_new <- seq(min(x), max(x), length.out = num_pun)
  y_kernel_new <- ksmooth(x, y, kernel = "normal", bandwidth = bw,
      x.points = x_new)

  plot(x, y, col = "red", pch = 16, main = paste("Kernel Smoothing (",
      "normal", ") - Index", i, sep = ""), xlab = "Maturity", ylab =
      "Return")
  lines(y_kernel_new$x, y_kernel_new$y, col = "blue", lwd = 2)
}
```

We will plot the results.

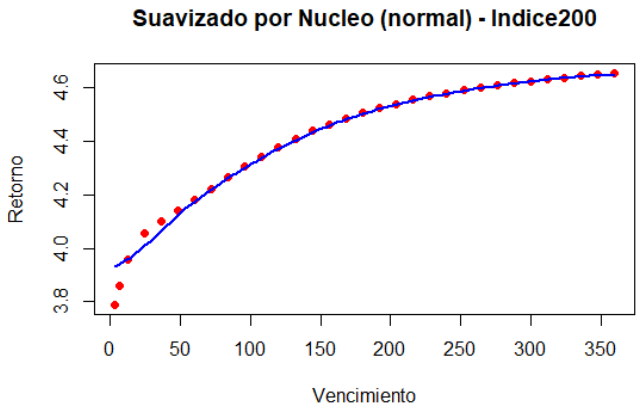

Figure 4.9: Approximation of the functions in the discrete dataset `ECBYieldcurve` using kernel smoothing

We observe that this estimation does not produce good results on the initial values of the function due to excessive smoothing; exploring different values of h produces similar results.

Local Polynomial Regression:

```
regresion_polinomica_local <- function(i, num_pun , grado ) {
  x <- datos$x
  y <- datos$y[, i]

  locfit_model <- locfit(y ~ lp(x, deg = 3))

  x_new <- seq(min(x), max(x), length.out = 100)
  y_locfit_new <- predict(locfit_model, newdata = data.frame(x = x_new))

  plot(x, y, col = "red", pch = 16, main = paste("Local Polynomial
      Regression - Index", i), xlab = "Maturity", ylab = "Return")
  lines(x_new, y_locfit_new, col = "blue", lwd = 2)
}
```

We will plot the results.

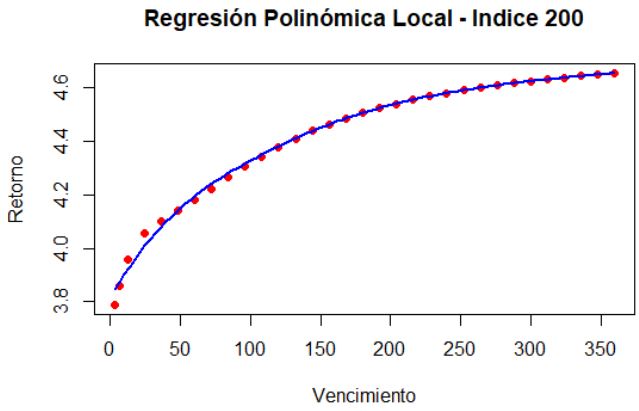

Figure 4.10: Approximation of the functions in the discrete dataset `ECBYieldcurve` using local polynomial regression

Like the previous method, it does not fit the observations well, although its deviation is smaller.

Spline Smoothing:

```
spline_suavizado <- function(i, num_pun) {
  x <- datos$x
  y <- datos$y[, i]

  spline_model <- smooth.spline(x, y)

  x_new <- seq(min(x), max(x), length.out = num_pun)
  y_pred <- predict(spline_model, x = x_new)$y

  plot(x, y, col = "red", pch = 16, main = paste("Spline Smoothing -
      Index", i), xlab = "Maturity", ylab = "Return")
  lines(x_new, y_pred, col = "blue", lwd = 2)
}
```

We will plot the results.

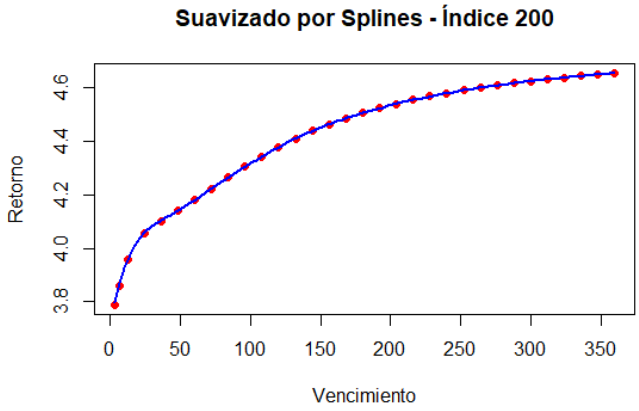

Figure 4.11: Approximation of the functions in the discrete dataset `ECBYieldcurve` using splines

It is impossible to measure the error committed in each approximation, since we only know discrete observations of the real functions. Having used these observations to approximate the coefficients, we cannot use them when measuring the error, as it will always be much smaller than the true error.

As a second step in the descriptive method, we study hidden structures in the dataset through clustering methods.

4.2 Clustering

Next, we will apply the three clustering methods shown in 2 to the functional dataset `ECBYieldcurve`.

We proceed to divide the functions into groups. We begin by studying the optimal number of groups, for which we apply the elbow method, a graphical method that allows determining the optimal number of groups through an error metric that measures the variation within each group. The two most popular metrics are,

$$WCSS = \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 \quad \text{and} \quad BIC = p\ln(n) - 2\ln(L)$$

which have already been shown in 2.1 and 2.2.

Next, the variation of the error metric with respect to k, the number of groups, is plotted, and the value of k is taken from which the variation becomes less significant; graphically, this appears as an elbow.

In each subsection, we will start by applying the elbow method; once the optimal number of groups is determined, the method will be applied and the results plotted. The code for the corresponding methods can be found in the appendix.

4.2.1 Filtering Method

We begin by studying the optimal number of groups, and then apply the filtering method using B-Spline bases to reduce the dimension of the functional observations and the K-means algorithm to cluster the observations once the dimension is reduced, using $WCSS$ as the error metric.

To see the optimal number of groups in which to divide the functional data, we apply the elbow method:

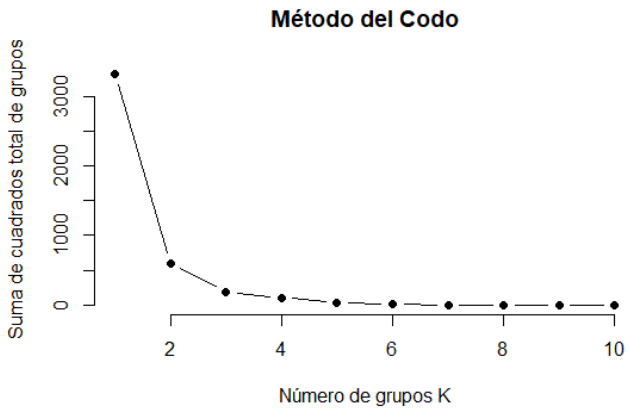

Figure 4.12: Elbow method on the filtering method using B-Splines and K-Means

We observe that the optimal number of groups is $k = 2$, so we will apply the filtering method with 2 groups, obtaining:

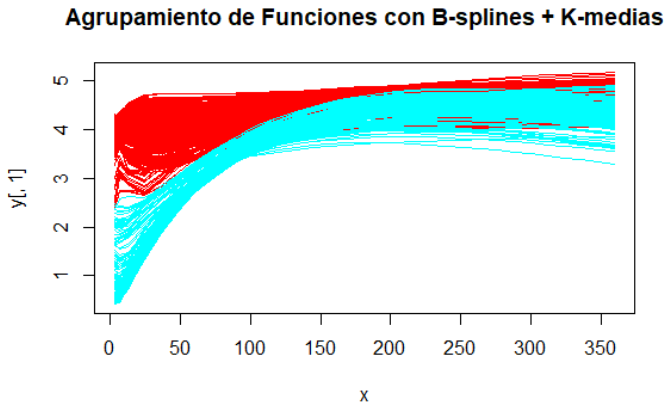

Figure 4.13: Function clustering provided by the filtering method using B-Splines and K-Means

The first group (red) contains 477 functions, while the second group (cyan) contains 178 functions. We can observe that the criterion applied by the method to divide the functions is based on the values taken by the function in the first 50 months.

4.2.2 Adaptive Method

We follow the same procedure as in the previous section, starting by studying the optimal number of groups, and then apply the adaptive method through a B-Spline decomposition to reduce the dimension of the functional observations and the EM algorithm to cluster the observations once the dimension is reduced. In this case, we use BIC as the error metric to avoid overfitting.

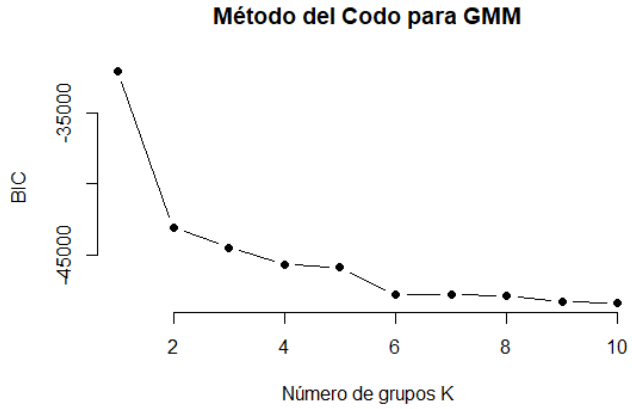

Figure 4.14: Elbow method on the adaptive method using B-Splines and the EM algorithm

In this case, the graph seems to present two elbows, one at $k = 2$ and another at $k = 6$; we opt for $k = 2$ as it has a more pronounced elbow.

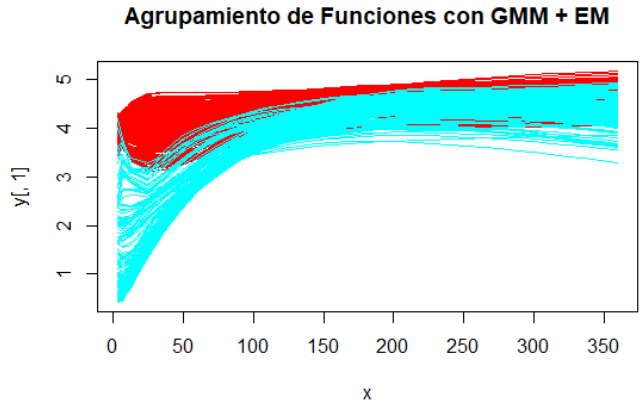

Figure 4.15: Function clustering provided by the adaptive method using B-Splines and the EM algorithm

The second group (cyan) contains 207 functions, while the first group (red) contains 448 functions. Both the size and distribution of the groups are quite similar to the previous method.

4.2.3 Distance Method

Finally, we apply the distance method. The functions will be estimated from discrete observations using a B-spline basis, and we will use the L^2 distance as the metric, measuring the clustering error through $WCSS$.

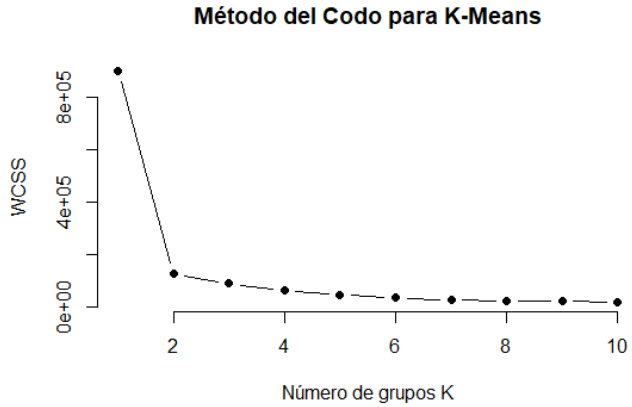

Figure 4.16: Elbow method on the distance method using K-Means

The elbow method indicates that the most efficient choice is $k = 2$, and the clustering is as follows:

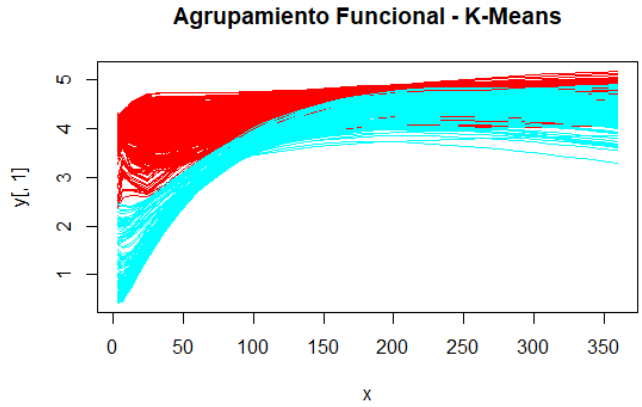

Figure 4.17: Functional dataset represented by the set of blue lines, the grey area represents the central 50% region

The first group (red) contains 478 functions, while the second group (cyan) includes 177 functions. From these results, we can conclude that the three clustering methods produce fairly similar results. In particular, the distance method and the filtering method show an almost identical distribution for this dataset, while the adaptive method generates slightly more balanced groups.

With this section, we conclude the descriptive study and move on to the inferential study, which involves setting up a regression model.

4.3 Scalar-on-Function Regression

In this section, we will set up a functional linear model where the independent variables are functions, while the dependent variables are scalars, i.e., the case treated in subsection 3.1.1.

The dependent variables, Y_i, represent the daily closing value of the IBEX35 index from 29/12/2006 to 24/07/2009 on business days, i.e., 655 days. The functions $X_i(t)$ represent the evolution of European bond returns over time. We want to study the relationship of the IBEX35 with European bond returns and see if we can predict the IBEX35 values from the bond returns.

We start by transforming the data provided by `ECBYieldcurve` into functions representing the evolution of returns over time. Assuming the functions can be approximated by B-Splines, we apply the techniques developed in 4.1. Next, we fit the functional regression model using different parametric estimation methods, and finally study the fit.

4.3.1 Ordinary Least Squares

In this subsection, we will estimate the parameter $\beta(t)$ using OLS.

```
nbasis <- 12
basis <- create.bspline.basis(rangeval = range(datos$x), nbasis = nbasis)

fdX <- Data2fd(argvals = datos$x, y = datos$y, basisobj = basis)

Y <- ibex$Último
model <- fRegress(Y ~ fdX)

Y_pred_fd <- model$yhatfdobj
Y_pred <- predict(model)
ECM <- mean((Y - Y_pred)^2)
print(paste("Mean Squared Error (MSE):", ECM))
print(paste("Relative Mean Squared Error:", ECM/var(Y)))

R2 <- cor(Y, Y_pred)^2
print(paste("R-squared:", R2))

plot(Y, Y_pred, main = "Observed vs Predicted Values",
     xlab = "Observed", ylab = "Predicted", ylim = c(8000,18000))
abline(0,1, col = "red")

beta <- model$betaestlist$fdX$fd
plot(beta, main = "Beta function", xlab = "Maturity", ylab = "Beta value")
```

We obtain an R^2 of 0.8133, i.e., 81.33% of the variability of the IBEX35 can be explained by the variation of European bond returns. Furthermore, the fit of each

individual value is as follows:

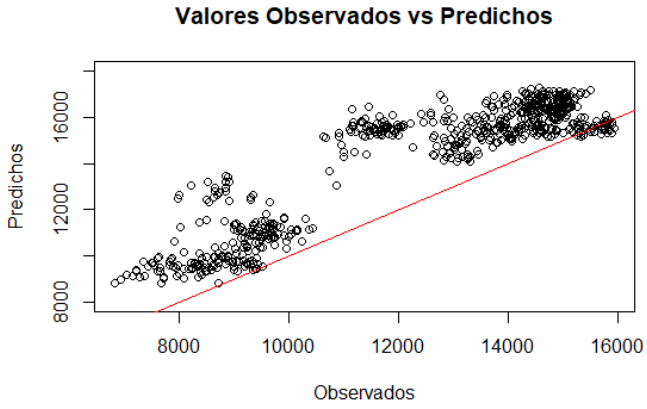

Figure 4.18: Graphical representation of observed vs fitted values

The model shows a tendency to overestimation, as most fitted values are higher than their respective observations. This is also reflected in the fact that the vast majority of points lie above the identity function.

We now represent the model's parameter function, which throughout the document we have denoted as $\beta(t)$.

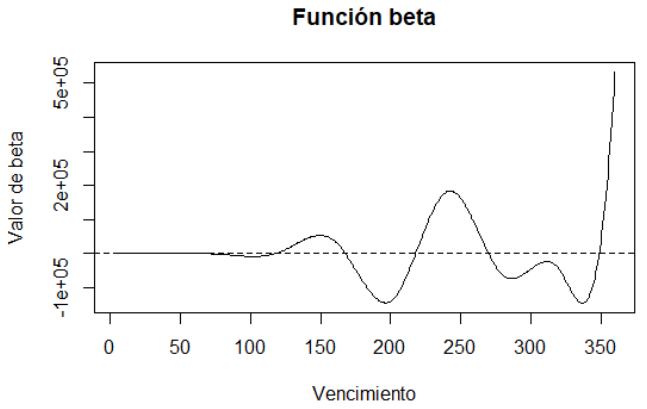

Figure 4.19: Representation of the function $\beta(t)$ over the bond maturities

We can observe that the IBEX 35 is not influenced by bond returns during the first 100 months. From that point onward, its impact begins to increase progressively. In most cases, an increase in bond returns is associated with an increase in the IBEX 35, except for two periods of decline around months 200 and 325, the second being more prolonged. In the final months, the effect of bond returns on the IBEX 35 is notably positive.

The model MSE is 4685565.3 while the relative MSE is 0.688. Note that these values are artificially low since we are calculating predictions on the same values

used to train the model. Nevertheless, the values are quite high, indicating that although there is a strong relationship between the variables, the model does not predict IBEX35 values well. This motivates the application of some penalization when estimating the model parameter $\beta(t)$ to reduce potential overfitting. Note that this will decrease R^2, but likely the MSE will also decrease.

4.3.2 Penalized Ordinary Least Squares

In this subsection, we will estimate the parameter $\beta(t)$ of the functional regression model using penalized OLS to reduce the MSE.

```
nbasis <- 12
basis <- create.bspline.basis(rangeval = range(datos$x), nbasis = nbasis)

fdX <- Data2fd(argvals = datos$x, y = datos$y, basisobj = basis)

Y <- ibex$Último

lambda <- 100
Lfdobj <- int2Lfd(2)

betaPar <- fdPar(basis, Lfdobj, lambda)

model <- fRegress(Y, list(fdX), list(betaPar))

Y_pred <- model$yhatfdobj

ECM <- mean((Y - Y_pred)^2)
print(paste("Mean Squared Error (MSE):", ECM))
print(paste("Relative Mean Squared Error:", ECM/var(Y)))

R2 <- cor(Y, Y_pred)^2
print(paste("R-squared:", R2))

plot(Y, Y_pred, main = "Observed vs Predicted Values",
     xlab = "Observed", ylab = "Predicted", ylim = c(8000,18000))
abline(0,1, col = "red")

beta <- model$betaestlist[[1]]$fd
plot(beta, main = "Coefficient Function with Penalization",
     xlab = "Maturity Term", ylab = "Coefficient")
```

The parameter λ has been optimized through cross-validation.

We plot the new graph of fitted vs predicted values:

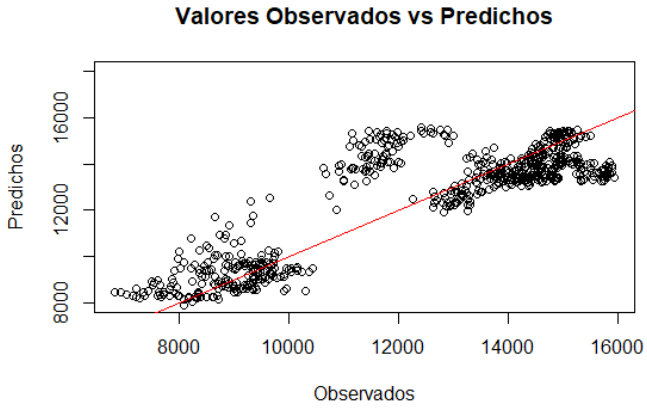

Figure 4.20: Graphical representation of observed vs fitted values

The fitted values are closer to the identity function, i.e., the estimation of $\beta(t)$ by penalized OLS provides a better fit, which will be reflected in the MSE calculation. Next, we plot the function $\beta(t)$.

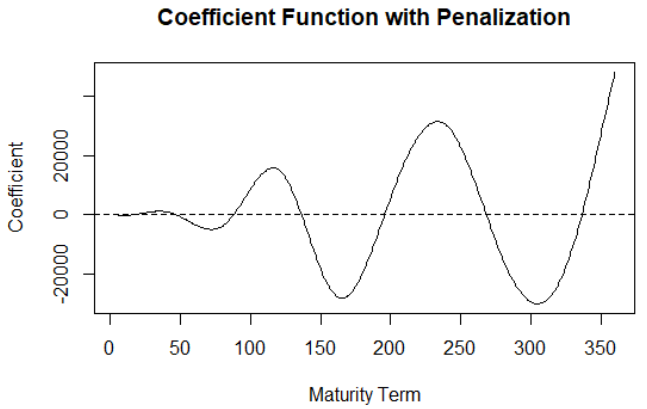

Figure 4.21: Representation of the function $\beta(t)$ over bond maturities

We observe that $\beta(t)$ takes much smaller values than when estimated by standard OLS, and, as expected, it appears smoother; in contrast, the R^2 has decreased to 0.7258. Meanwhile, the MSE has decreased to 1889493.9 and the relative MSE is 0.277. This indicates that although this model achieves a better approximation of individual observations, its overall fit is less precise.

In summary, in this chapter we have first approximated the functional data from discrete observations using different methods, then clustered the set of functions using various clustering methods, and finally, we set up a scalar-on-function linear regression model, obtaining the functional coefficient of the model in different ways.

4.4 Appendix

4.4.1 Filtering Method

```
agrupamiento_bspline_kmedias <- function(num_bases, num_grupos , semilla
    = 123) {

  x <- datos$x
  y <- datos$y

  base_bspline <- create.bspline.basis(rangeval = range(x), nbasis =
      num_bases)
  datos_fd <- Data2fd(argvals = x, y = y, basisobj = base_bspline)
  matriz_coef <- datos_fd$coefs

  set.seed(semilla)
  resultado_kmedias <- kmeans(t(matriz_coef), centers = num_grupos,
      nstart = 10)
  grupos <- resultado_kmedias$grupo
  colores_grupos <- rainbow(num_grupos)[grupos]

  plot(x, y[,1], type = "n", main = "Function Clustering with B-splines +
      K-means", ylim = range(y))
  for (i in 1:ncol(y)) {
    lines(x, y[,i], col = colores_grupos[i])
  }

  return(list(
    grupos = grupos,
    resultado_kmedias = resultado_kmedias,
    matriz_coef = matriz_coef
  ))
}

coef_matrix = agrupamiento_bspline_kmedias(num_bases = 10 , num_grupos =
    2)
summary(coef_matrix)
conteo_grupos <- table(coef_matrix$grupos)
print(conteo_grupos)

wcss <- numeric(10)
for (k in 1:10) {
  kmeans_result <- kmeans(t(coef_matrix$matriz_coef), centers = k, nstart
      = 10)
  wcss[k] <- kmeans_result$tot.withinss
}

plot(1:10, wcss, type = "b", pch = 19, frame = FALSE,
```

```
    xlab = "Number of Groups K", ylab = "Total Within-Cluster Sum of
       Squares",
    main = "Elbow Method")
```

4.4.2 Adaptive Method

```
agrupamiento_bspline_gmm <- function(num_bases, num_grupos, semilla =
    123) {

  x <- datos$x
  y <- datos$y

  base_bspline <- create.bspline.basis(rangeval = range(x), nbasis =
      num_bases)
  datos_fd <- Data2fd(argvals = x, y = y, basisobj = base_bspline)
  matriz_coef <- t(datos_fd$coefs)

  set.seed(semilla)
  resultado_gmm <- Mclust(matriz_coef, G = num_grupos)
  grupos <- resultado_gmm$classification

  colores_rainbow <- rainbow(num_grupos)

  if (num_grupos >= 2) {
    colores_rainbow[1] <- "cyan"
    colores_rainbow[2] <- "red"
  }

  colores_grupos <- colores_rainbow[grupos]

  plot(x, y[,1], type = "n", main = "Function Clustering with GMM + EM",
      ylim = range(y))
  for (i in 1:ncol(y)) {
    lines(x, y[,i], col = colores_grupos[i])
  }

  return(list(
    grupos = grupos,
    resultado_gmm = resultado_gmm,
    matriz_coef = matriz_coef
  ))
}

coef_matrix2 <- agrupamiento_bspline_gmm(num_bases = 10, num_grupos = 2)
summary(coef_matrix2$resultado_gmm)
```

```
bic_values <- numeric(10)
for (k in 1:10) {
  resultado_gmm <- Mclust(coef_matrix2$matriz_coef, G = k)
  bic_values[k] <- resultado_gmm$bic
}
plot(1:10, -bic_values, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Groups K", ylab = "BIC",
     main = "Elbow Method for GMM")
```

4.4.3 Distance Method

```
agrupamiento_funcional <- function(num_bases, num_grupos, metodo =
    "kmeans", semilla = 123) {

  x <- datos$x
  y <- datos$y

  base_bspline <- create.bspline.basis(rangeval = range(x), nbasis =
      num_bases)

  datos_fd <- Data2fd(argvals = x, y = y, basisobj = base_bspline)

  matriz_coef <- datos_fd$coefs

  interpolate_function <- function(coefs, basis) {
    return(function(x) {
      basis_values <- eval.basis(x, basis)
      basis_values <- as.matrix(basis_values)
      coefs <- as.vector(coefs)
      result <- basis_values %*% coefs
      return(result)
    })
  }

  interpolated_functions <- apply(matriz_coef, 2, function(coefs)
      interpolate_function(coefs, base_bspline))

  dist_matrix <- matrix(0, ncol = ncol(y), nrow = ncol(y))
  for (i in 1:(ncol(y)-1)) {
    for (j in (i+1):ncol(y)) {
      dist_matrix[i, j] <- compute_distance(interpolated_functions[[i]],
          interpolated_functions[[j]], min(x), max(x))
      dist_matrix[j, i] <- dist_matrix[i, j]
    }
  }
```

```
  set.seed(semilla)

  if (metodo == "kmeans") {
    resultado <- kmeans(dist_matrix, centers = num_grupos, nstart = 10)
    grupos <- resultado$cluster
    titulo <- "Functional Clustering - K-Means"
  } else if (metodo == "hierarchical") {
    resultado <- hclust(as.dist(dist_matrix), method = "ward.D2")
    grupos <- cutree(resultado, num_grupos)
    titulo <- "Functional Clustering - Hierarchical"
    plot(resultado, main = "Dendrogram - Hierarchical Clustering", xlab =
        "Functions", ylab = "Distance")
  }

  colores_rainbow <- rainbow(num_grupos)

  if (num_grupos >= 2) {
    colores_rainbow[1] <- "red"
    colores_rainbow[2] <- "cyan"
  }

  colores_grupos <- colores_rainbow[grupos]
  plot(x, y[,1], type = "n", main = titulo, xlim = range(x), ylim =
      range(y))
  for (i in 1:ncol(y)) {
    lines(x, y[,i], col = colores_grupos[i])
  }

  heterogeneity_index <- function(cluster_assignments, dist_matrix) {
    unique_groups <- unique(cluster_assignments)
    hi_values <- sapply(unique_groups, function(g) {
      indices <- which(cluster_assignments == g)
      cluster_dist <- dist_matrix[indices, indices, drop = FALSE]
      mean(cluster_dist)
    })
    return(mean(hi_values))
  }

  shi <- heterogeneity_index(grupos, dist_matrix)

  return(list(
    grupos = grupos,
    resultado = resultado,
    matriz_coef = dist_matrix,
    SHI = shi
  ))
}
```

```r
coef_matrix3 <- agrupamiento_funcional(num_bases = 10, num_grupos = 2,
    metodo = "kmeans")
summary(coef_matrix3$resultado)

conteo_grupos <- table(coef_matrix3$grupos)
print(conteo_grupos)

elbow_method <- function(matriz_coef, max_grupos = 10, metodo = "kmeans",
    semilla = 123) {
  dist_matrix <- as.matrix(dist(matriz_coef, method = "euclidean"))

  metric_values <- numeric(max_grupos)

  set.seed(semilla)

  for (k in 1:max_grupos) {
    if (metodo == "kmeans") {
      resultado <- kmeans(dist_matrix, centers = k, nstart = 10)
      metric_values[k] <- resultado$tot.withinss
    } else if (metodo == "hierarchical") {
      resultado <- hclust(as.dist(dist_matrix), method = "ward.D2")
      grupos <- cutree(resultado, k)
      metric_values[k] <- heterogeneity_index(grupos, dist_matrix)
    }
  }

  plot(1:max_grupos, metric_values, type = "b", pch = 19, frame = FALSE,
       xlab = "Number of Groups K", ylab = ifelse(metodo == "kmeans",
           "WCSS", "Heterogeneity Index"),
       main = paste("Elbow Method for", ifelse(metodo == "kmeans",
           "K-Means", "Hierarchical Clustering")))

  optimal_k <- which.min(diff(metric_values)) + 1
  cat("Optimal number of clusters (elbow point):", optimal_k, "\n")

  return(list(
    metric_values = metric_values,
    optimal_k = optimal_k
  ))
}

heterogeneity_index <- function(cluster_assignments, dist_matrix) {
  unique_groups <- unique(cluster_assignments)
  hi_values <- sapply(unique_groups, function(g) {
    indices <- which(cluster_assignments == g)
    cluster_dist <- dist_matrix[indices, indices, drop = FALSE]
    mean(cluster_dist)
  })
  return(mean(hi_values))
```

```
}

elbow_method(coef_matrix3$matriz_coef)
```

Bibliography

[1] M. Avery, *Literature Review for Local Polynomial Regression*, Unpublished, 2010.

[2] J. S. Marron, et.al, *Functional data analysis of amplitude and phase variation*, Statistical Science, 2015

[3] J.Mercer, et.al, *Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations* , Philosophical Transactions of the Royal Society of London, 1909

[4] F.Scheipl, et.al, *Generalized functional additive mixed models*, Electron. J. Statist, 2015.

[5] J. O. Ramsay, B. W. Silverman, et.al, *Functional Data Analysis*, Electron. Springer, 2005.

[6] J. Gertheiss, et.al, *Functional Data Analysis: An Introduction and Recent Developments*, Wiley. Annual Review of Statistics and its Application, 2024.

[7] P. Kokoszka, M. Reimherr *Introduction to Functional Data Analysis*, CRC Texts in Statistical Science, 2017.

[8] C. M. Crainiceanu *Functional Data Analysis with R*, CRC Monographs on Statistics and Applied Probability, 2024.

[9] F. Ferraty, P. Vieu *Nonparametric Functional Data Analysis*, Springer, 2006.

[10] B. Ufuk, S. Han Lin, *On function-on-function regression: partial least square-sapproach*, Environmental and Ecological Statistics, 2020.

[11] H. Ma, Z. Zhu, *NContinuously dynamic additive models for functional data*, Journal of Multivariate Analysis, 2016.

[12] M. Reimherr, B. Sriperumbudur, *Optimal Prediction for Additive Function-on-Function Regression*, Electronic Journal of Statistics, 2017.