Characterising the Shape of the COVID-19 Pandemic

Using Functional Data Analysis

Joseph Piekos

Department of Mathematics
Durham University
April 27, 2022

Abstract

The COVID-19 pandemic has been one of the deadliest in history, making visualisation and quantification of the trends in deaths crucial. I will characterise the different, smaller scale COVID-19 epidemics that occurred across England, using Functional Data Analysis. FDA provides smoothing methods that allow the data to be displayed as curves to identify their shape, with functional summary statistics used to quantify curve features. I will then attempt to explain and predict the trends in mortality using mobility and PCR test positivity data, focusing mainly on the total functional model. Some of the restrictions and differences in working with functional data instead of multivariate data are addressed throughout.

Contents

List of Figures

List of Tables

Plagiarism declaration

This piece of work is a result of my own work except where it forms an assessment based on group project work. In the case of a group project, the work has been prepared in collaboration with other members of the group. Material from the work of others not involved in the project has been acknowledged and quotations and paraphrases suitably indicated.

Chapter 1

Introduction

The COVID-19 pandemic has been one of the worst in human history, with more than 500 million cases and more than 6 million deaths worldwide attributable to the disease over a period of more than 2 years. Therefore, it is essential to understand how a modern pandemic spreads so that we can inform future government policy by identifying the key indicators of pandemic strength and growth. Functional Data Analysis (FDA) is a branch of statistics that allows us to do this, by analysing data sets made up of smooth curves. In particular, we can reduce a large amount of discrete data collected over time into a set of curves. We can then analyse the *shapes* of these curves, in order to dissect the trends and find statistical relationships, within and between variables.

I will apply FDA techniques primarily to deaths from COVID-19 (mortality) in England, in order to further characterise and confirm the trends we saw first hand or reported by the government. In the context of FDA, we say we are considering mortality as a *functional variable*, and I will also analyse its statistical relationship with other functional variables: Mobility and PCR test positivity (% of tests that come back positive), which are known possible predictors of COVID-19 deaths. The data will be studied at the resolution of 78 English counties, with the hope of also characterising the smaller scale epidemics that occurred around the country, during the first wave of the pandemic.

COVID-19 studies using FDA have been done using data from the US (Tang, T. Wang, and P. Zhang 2020) and Italy (Boschi et al. 2021). However, as far as I can tell, FDA has not been used to examine UK COVID-19 data, so I hope this report can contribute to this area and provide some different perspectives.

Throughout the following chapters, I will introduce the theory of FDA, and use the methods to analyse the data as I go. This report consists of 8 chapters, with Chapter 1 introducing the concept of functional data. Chapter 2 presents the details of the coronavirus data set I will be using. Chapter 3, Chapter 4, and Chapter 5 describes the steps required to do any functional data analysis, including basis expansions, smoothing, and curve registration.

Chapters 6 and 7 describe a subset of advanced FDA techniques: Functional versions of regression, covariance, and box plots. I will use these to answer questions such as: What variables are statistically related to COVID-19 mortality? Which areas of England were affected the worst by the Pandemic? Finally, Chapter 8 summarises the methods, results, and challenges of performing a functional data analysis, and then suggests possible further research.

All the data and R code used to produce the results, simulations, and plots in this report is available from my GitHub page: https://github.com/joepiekos/Project-III. The code is produced by myself, either from scratch or by heavily modifying existing code.

1.1 What is Functional Data?

A *functional* data set, as described by Kokoszka and Reimherr (2017, p. 1), is of the form:

$$\underbrace{y_i(t_{i,j})}_{\text{smooth function}} \quad , \quad \underbrace{t_{i,j} \in [T_1, T_2]}_{\text{continuum}}, \quad \underbrace{i = 1, 2, \ldots, N}_{\text{number of replicates}}, \quad \underbrace{j = 1, 2, \ldots,}_{\text{sampling points}} J_i \tag{1.1}$$

The fundamental idea in FDA is that each object we analyse is a smooth function, $y_i(t)$, which we assume exists for all values of t in a common interval $[T_1, T_2]$. This interval (or continuum) normally represents time, hence the letter t, however it can be anything from frequency to probability. To give this abstract idea of functional data some context, consider a situation where we are measuring the heights of boys as they grow up.

Each i represents a different person and $y_i(t)$ is the height *function* for that person. We call it a "function" because, in theory, we could know their height at every single moment in time. However, storing such an uncountable number of values is just not possible, and would be awkward to measure. Therefore, we observe instances of **functional data** as the heights y_i at specific times $t_{i,j}$, where $j = 1, \ldots, J_i$ represents the number of times we measure them. One thing to note is that these points $t_{i,j}$ do not have to be the same for each subject.

The assumption of *smoothness* of the functions means that we expect adjacent data points, $y_i(t_{i,j})$ and $y_i(t_{i,j+1})$, to not be drastically different from each other. This assumption is the key thing that separates functional data from standard multivariate data. Mathematically, a *smooth* function has at least one derivative. In the case of our height example, FDA allows us to consider the acceleration (2^{nd} derivative) etc. of the boys' growth.

In summary, by just introducing the structure of functional data, we have already discovered many of the benefits of using FDA (Ullah and Finch 2013):

- FDA methods are flexible in the sense that the **sampling intervals** for data do not have to be equally spaced for all cases.
- Values measured at different times for a certain subject will not be **independent**, this is not a problem for a functional data object. Classic multivariate statistics typically requires independence assumptions to reduce complexity.
- FDA takes into account the **smooth** behaviour of the functions that are assumed to underlie and to generate the observations. For example, assuming height increases smoothly over time isn't an unreasonable idea.
- FDA methods can often extract additional information contained in the function's **derivatives**.

1.2 Some examples of FDA

In this section, I will present two examples of functional data, in order to show the difference from a standard multivariate data set. Both data sets use time as the continuum for sampling. Functional data analysis is inherently a visual branch of statistics, so I will show the **smoothed** curves representing each data set. Smoothing methods are discussed in detail in Chapter 4, however for reference both data sets here are smoothed using a roughness penalty on a B-Spline basis.

1.2.1 Growth data

Data here is sourced from Tuddenham and Snyder's 1954 Berkeley Growth Study (Tuddenham 1954). Height data acts as an easy introductory example to FDA, and I will use it to show some key features and benefits of using functional data. However, the main focus of the report, and what the theory will be explained alongside, is the COVID-19 data introduced in Section 1.2.2. Height data possesses some key properties of any functional data: It varies smoothly over time, and can be measured, in theory, at any point in time. In particular, the observed heights represent instances of an underlying height **function**, which can be thought of as a naturally smooth data generating process. Height is also not sampled at equal intervals for the 18 years, as can be seen in the first column of Table 1.1 and in Figure 1.1 below. As already noted, this won't be a problem, as we are simply using this data to estimate the underlying function.

Table 1.1: Heights for 10 boys from 1 to 18 years old Includes the notation from Equation 1.1 for a functional data set.

Age (years)	Height of boy i									
	1	2	3	4	5	6	7	8	9	10
$t_{i,1} = 1$	81.3	76.2	76.8	74.1	74.2	76.8	72.4	73.8	75.4	78.8
$t_{i,2} = 1.25$	84.2	80.4	79.8	78.4	76.3	79.1	76.0	78.7	81.0	83.3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t_{i,29} = 17$	193.8	176.1	170.9	181.4	172.5	172.6	171.1	184.7	171.6	187.8
$t_{i,30} = 17.5$	194.3	177.4	171.2	181.6	172.5	173.2	171.8	185.0	172.3	188.2
$t_{i,31} = 18$	195.1	178.7	171.5	181.8	172.5	173.8	172.6	185.2	172.9	188.4

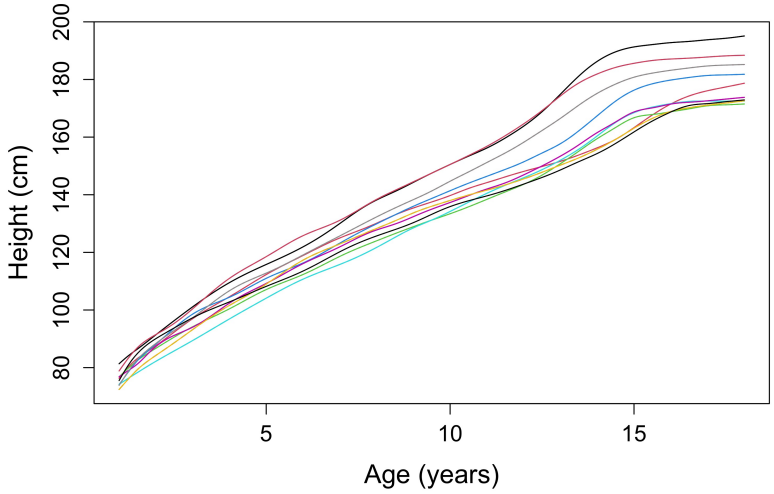

Figure 1.1: The height of 10 boys, each measured at 31 ages. Each smooth curve is one **functional** object, instead of each value of height being one data object. Curves are smoothed using a roughness penalty with $\lambda = 10^{-4}$ and an order 4 B-spline basis.

Figure 1.1 shows the height data smoothed into 10 curves. Producing this plot allows us to notice some features of the children's growth, which aren't as obvious with just the numerical data. In particular, we look for two types of variability, **amplitude** and **phase**. Amplitude variation is obvious in the curves, which simply represents certain boys being taller than others. Phase variation accounts for shifting of the curves with respect to the domain, time. There doesn't appear to be much obvious phase variation in the height curves, however we know of such thing as a "body clock", so we might expect there to be some in the derivative curve. Before any functional data analysis, we must almost always separate amplitude and phase variation to avoid confounding our results. This is done using **curve registration** methods, and is described in detail in Chapter 5.

1.2.2 COVID-19 mortality data

We can also get functional data from the COVID-19 pandemic, in the form of mortality numbers sourced from the UK coronavirus dashboard (UK Health Security Agency 2022a). I will also look at other data, which will be used as independent variables to explain variation in our dependent variable, mortality. This will include functional variables in mobility and positivity. Scalar variables can also be considered simultaneously in these models, however in this report I will focus solely on using functional variables.

In this section, I will present mortality as another example of functional data, and the rest of the COVID-19 data set will be discussed in Chapter 2. The mortality data has the same functional structure to the growth data in Table 1.1. There are daily deaths measurements (converted into death rates per 100,000) for the entire 180 days, giving us a dense set of functional data (see Remark 1). The mortality data are smoothed in almost the exact same way as the growth data, with the added complexity here being that the curves must be nonnegative (since we cannot have a negative number of deaths). This is a concept known as smoothing a *constrained* function, and is explained in Chapter 4.

Another thing to note is how the mortality curves in Figure 1.2 have some more interesting features: Multiple peaks and steeper gradients to name a few. This means smoothing the data requires more careful decisions to be made, in particular the "amount" of smoothing we do versus how much of the variation in the data we capture. This decision is a classic example of a **bias variance tradeoff**. In Chapter 4 we will quantify the amount of smoothing we want with smoothing parameters, λ (roughness penalty) and K (number of basis functions), to accurately show the trends in the data.

***Remark* 1** (Sparse vs dense data)**.** The "**sparsity**" of data in FDA has different meanings depending on the context. Typically, it refers to when data are only collected at a small number of time points over the whole interval, where non-sparse data is called **dense**. This is quite a common occurrence since, for example, patients can forget to show up for their already scarce yearly appointments. There is no formal definition for sparse data, however X. Zhang and J.-L. Wang (2016) provide a practical definition: If for all subjects i, the number of sampling points, J_i, is greater than the number of curves, N, then the data is referred to as "dense". A second definition for sparsity is discussed by Aneiros and Vieu (2014) and refers to the problem of deciding whether to use the whole functional observation, $y_i(t)$, in a regression problem or only a part of it. For the purposes of this report, we will go with the first definition in terms of having more sampling points than the number of curves. All the COVID-19 data that I am using are therefore dense, and I will not mention Sparse-FDA methods again. However if interested, Kokoszka and Reimherr (2017, Chapter 7) gives a good introduction to some sparse methods.

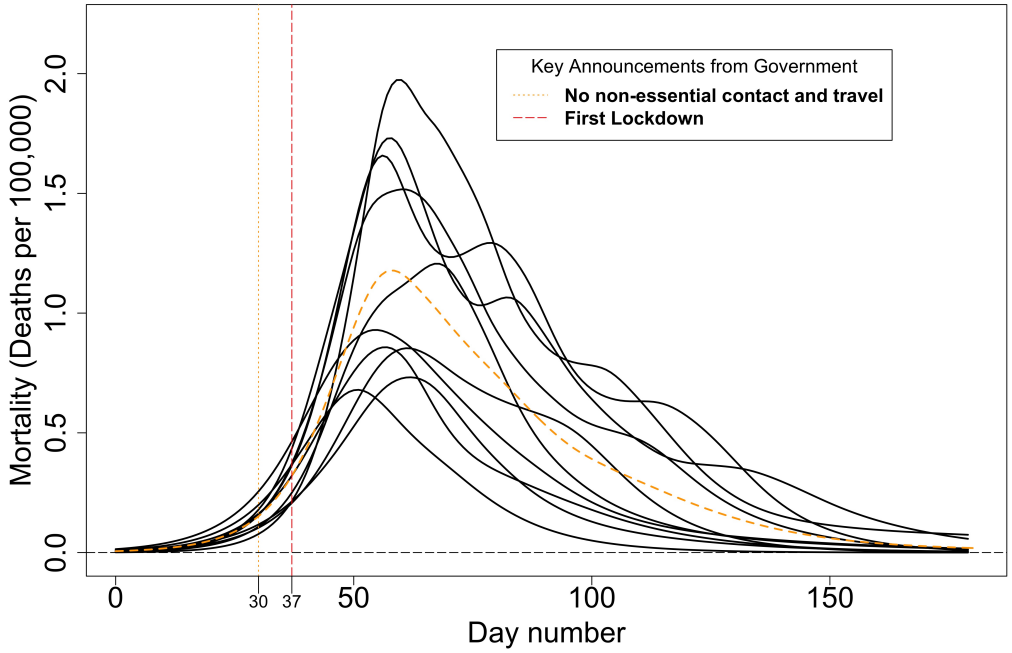

Figure 1.2: The mortality curves for 10 of the 78 English counties, over 180 days from 15/02/2020 to 13/08/2020. The two vertical dotted lines represent key moments in the pandemic, which give context to the rising mortality. The curves have been smoothed using a B-spline basis of order 4 with a roughness penalty and the added constraint of non-negativity. The orange dotted line represents the mean function of the 10 curves, which simply the mean of the function values pointwise.

Chapter 2

Data retrieval and processing

The previous chapter introduced the concept of functional data. Here, we will further examine the mortality data set from Section 1.2.2, introduce more data related to it, and then discuss some questions that we could answer about them.

A **functional variable** has the same definition as in Chapter 1: The discrete data are collected over a **continuum**, normally time, and are considered as the realisations of a **smooth function**. One smooth function is one *functional* data point of a functional variable. Excluding the height data previously discussed, I will consider three functional variables: Mortality, mobility and positivity. All three are from the COVID-19 pandemic, and from now on, unless we consider the "height" example (see Section 1.2.1), all data will be related to COVID.

A key consideration when choosing this functional data set is the **data resolution**, which leads to the concept of a function's dimensionality. Ramsay and Silverman (2005) explain that expert knowledge of functional analysis (note that this is a different branch of mathematics from functional *data* analysis) is required to fully understand dimensionality, which is slightly outside the scope of this report. Therefore, I will give *dimensionality* a more practical definition of the total amount of information required to define all the "features" of the function, i.e. local maxima and minima.

There is no correct answer when it comes to choosing the data resolution. It comes down to the context of the problem, alongside some general "rules of thumb". Ramsay and Silverman (2005) recommend 12-16 pieces of information for growth curves as in Figure 1.1. Boschi et al. (2021) researched COVID-19 in Italy at a resolution of 20 Italian "regions", however they stated that this resolution limited the size of the statistical models that can can be reliably fit on the data. From these two sources I concluded that having anything about 20 would be effective, and in the end data for 78 English counties was the most complete data set which fit this criterion.

There are technically 84 counties in England (Wikipedia 2022), however 6 of them had incomplete data and were excluded from my data analysis. This isn't too much of a problem, as our sample size of 78 is still large and includes a spread of counties across the country.

Seven of the counties are *metropolitan* counties, such as Greater London or Tyne and Wear, which are sort of "piecewise" counties made up of multiple cities and smaller districts. Therefore I decided to combine data so we could consider these important counties as a single object. Furthermore it was difficult to find data that was collected over a consistent timescale. Certain counties started collecting data on deaths and PCR test at different times, so I set these variables to zero for the time periods before the first value was collected.

2.1 Functional Variables

In the following sections, all COVID-19 data will be for 181 days from 15/02/2020 to 13/08/2020. Throughout the report, "day 0" represents 15/02/20 and "day 180" represents 13/08/2020.

2.1.1 Mortality

Mortality data is sourced from the UK coronavirus dashboard, published by the UK Health Security Agency (2022a). Data are initially given as cumulative deaths, where the deceased was identified as a case of COVID-19 by a positive test and died within 28 days of being identified as a case. To produce the mortality curves, I calculated daily death increments and converted each into a mortality number (or death rate) using the following formula:

$$\text{Death rate} = \left(\frac{\text{County deaths}}{\text{County population}}\right) \times 100,000 \tag{2.1}$$

The multiplicative factor of 100,000 means that we interpret the above *death rate* as the number of deaths per 100,000 people, allowing us to compare regions even though they have different populations. I chose to define the death rate in this way because it resulted in values of mortality ranging from 0 to 7, instead of all values being decimals between 0 and 1. This means it is easier for a reader to notice differences in mortality more obviously, and also scaling the values up away from 0 helps to prevent the final smoothed curve going below zero (we fully prevent this happening using constrained functions in Chapter 4, however this preprocessing means we lose less information when we do end up smoothing in this way.) The ONS guide on mortality statistics states that death rates should be calculated using mid-year population estimates (Cornish and Lohawala 2021, p. 7), hence I have used mid-2019 population estimates sourced from the ONS (2020).

2.1.2 Mobility

Mobility data is sourced from Google's Community Mobility Reports (Google LLC 2022), with daily measurements for each English county, as with the mortality data. A value of **mobility**, which is expressed as a percentage, reflects the number of visits and the length of stay at different locations compared to a baseline. For reference, the baseline is the median value for the corresponding day of the week, during five weeks from 3$^{\text{rd}}$ January to 6$^{\text{th}}$ February 2020. Figure 2.1 shows the mobility curve for County Durham, in particular for trips to grocery and pharmacy shops

2.1.3 Positivity

Positivity is the final functional variable I will be considering, and it represents the percentage of people on a given day who took a COVID-19 PCR test and returned a positive result. The data is sourced, like the mortality data, from the UK Health Security Agency (2022b). The value on a given day is actually calculated by the UKHSA using rolling sums, of the previous 7 days, of the number of people who tested positive and number of people who took a test. Then the former sum is divided by the latter, to find the positivity. Figure 2.2 shows two examples of positivity curves, that we will use the methods in the next few chapters to create. The curves look similar to the mortality curves in Figure 1.2, however the distinguishing factor is that the y-axis now has limits of 0 and 100.

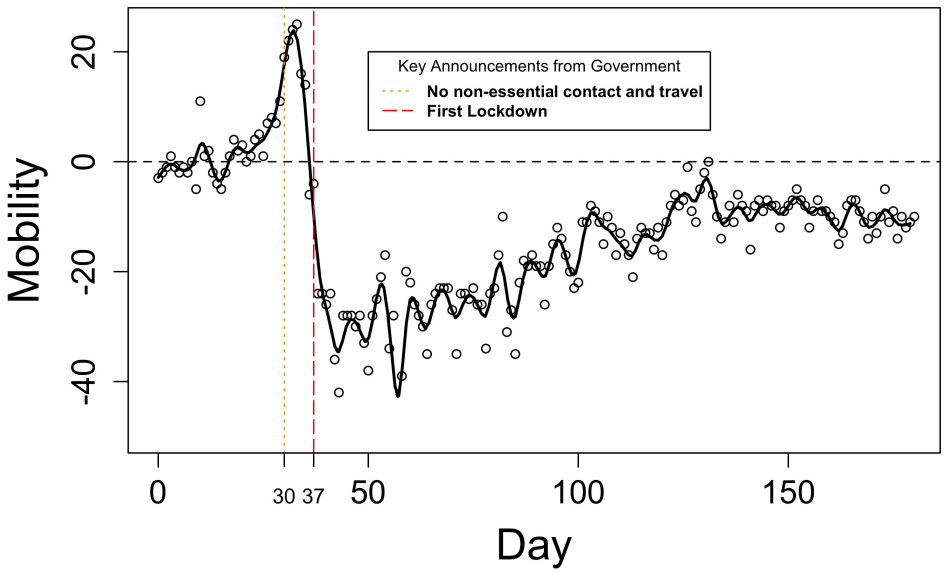

Figure 2.1: The mobility curve for County Durham (grocery and pharmacy visits), over 180 days from 15/02/2020 to 13/08/2020. The two vertical dotted lines represent key moments in the pandemic, such as the first lockdown on March 23rd, 2020, which give context to the sharp increases and decreases in trips to these places. The curve has been smoothed using a B-spline basis of order 4 with a roughness penalty and smoothing parameter of $\lambda = 1$.

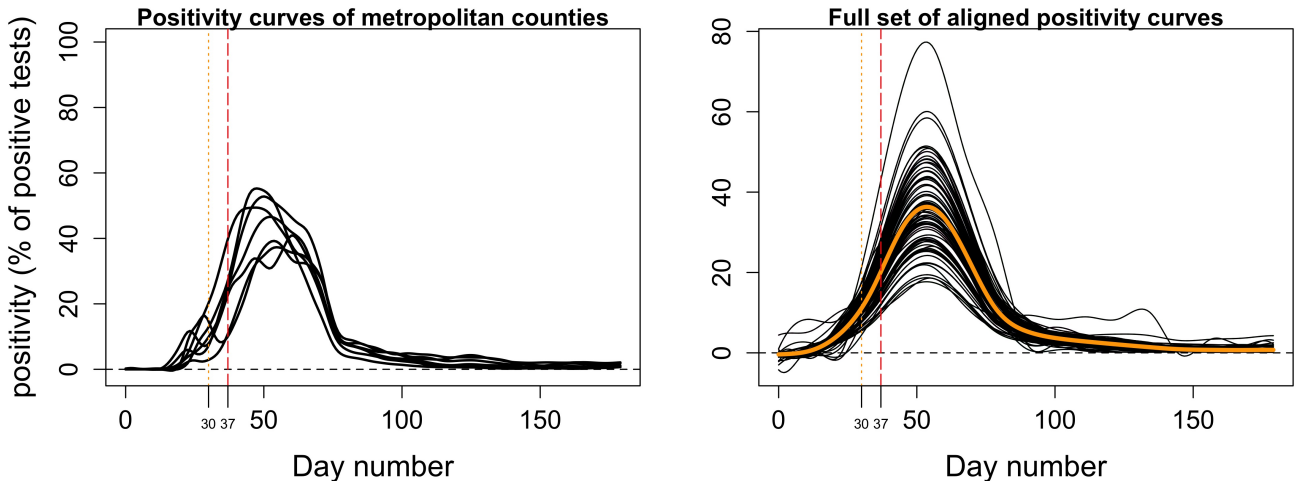

Figure 2.2: Two examples of positivity curves. Left panel: Smooth curves representing the seven metropolitan counties in England (Greater London, Greater Manchester, Tyne and Wear and so on). The curves are smoothed using a roughness penalty with a smoothing parameter of $\lambda = 5$ (selected using GCV). Right panel: The full set of 78 positivity curves, which have been smoothed in the same way as the in left-hand image. However, I have then also applied continuous registration align them, and remove variation along the time axis.

Chapter 3

First steps in FDA

Chapters 1 and 2 introduced the concept of functional data alongside the functional data set I will be working with. However, it also contained a lot of FDA jargon, with words like "smoothing" and "registration". Figure 3.1 includes some of these terms, in particular laying out the steps that must be taken with functional data before a full functional data analysis can be performed. The next few chapters aim to explain what these terms mean, the mathematics behind the methods, and show their implementation using R.

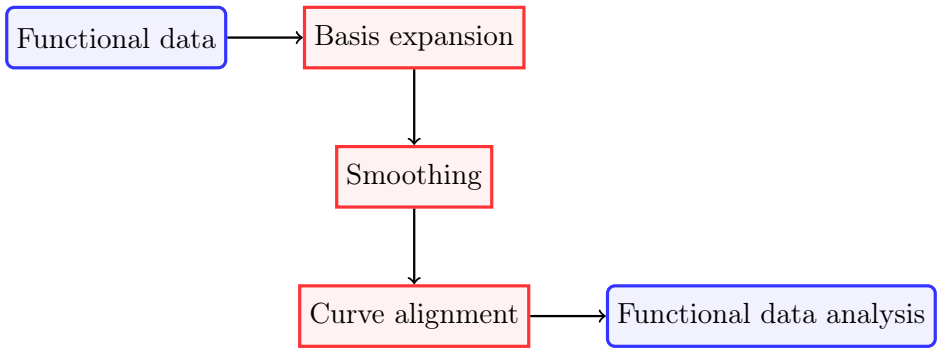

Figure 3.1: The general steps that must be taken before any functional data analysis.

Knowledge of undergraduate matrix algebra and real analysis will generally be assumed when discussing the theoretical side of FDA. However, I will ensure to explicitly include any external theorems, lemmas, etc. on these topics if needed.

3.1 Basis Expansions

In all FDA problems, we are assuming that there is an underlying function, $x_i(t)$, for which our discrete data, $\boldsymbol{y}_i$ are a few instances of at certain times, $t_{i,j}$. However, in order to do any *functional* data analysis, we need a mathematical representation of this function (which includes values for the whole interval, not just our finite time values). The most basic idea would be to just choose one of our favourite known functions (e.g. $\sin(t)$ or e^t) and maybe try to translate or scale it to fit all the data we have. It would take a very long time to find such an appropriate function, and in the case of Figure 2.2 would be practically impossible.

Instead, we use a set of special functions, called basis functions:

Definition 3.1.1 (Basis functions)**.** A set of functions, $\{\phi_1(t), \phi_2(t), \ldots, \phi_K(t)\}$, which are linearly independent of each other. These functions form a **basis** for a vector space, V, if any element of V can be written as a linear combination of the basis functions.

In FDA, different sets of basis functions are used depending on the scenario, as certain bases each have their own strengths. Once we have chosen a set of basis functions, we can choose a linear combination of them (known as a basis expansion, see Definition 3.1.2) that ultimately looks like our desired function. In this report, I will be focusing on the B-spline basis, which is used to construct spline functions.

***Remark* 2** (Spline function space)**.** Referring back to Definition 3.1.1 of basis functions, the only vector space (or function space) we will need a basis for is the **spline function space**, S. However, when considering FDA as a whole, we work with functions in the larger space of square integrable functions, L^2 (Kokoszka and Reimherr 2017, p. 37). This is part of the world of Hilbert spaces and functional analysis which, as stated before, is not necessary for this report and hence will not be delved into any further.

Definition 3.1.2 (Basis expansion)**.** Given a set of basis functions, $\{\phi_1(t), \phi_2(t), \ldots, \phi_K(t)\}$, the **basis expansion** of a function, $y(t)$, is a linear combination of the basis functions:

$$x(t) = \sum_{k=1}^{K} c_k \phi_k(t) = \boldsymbol{c}^T \boldsymbol{\phi} \tag{3.1}$$

Where $\boldsymbol{c}$ is a column vector of the coefficients, c_k, of each basis function.

One of the main benefits of using a basis expansion is that we have gone from having to store an infinite number of values for our underlying function, to just a finite number, K, of coefficients for the basis functions in our basis expansion.

Interpolation vs smoothing

When collecting data in real life, there is always a certain level of "error" associated with each observation. This may arise from errors in our measurement tools, problems with the method, or simply random errors where it is almost impossible to distinguish where it came from. We want all of our results to only take into account the "true" value of our function, $x(t_j$ (Ramsay and Silverman (2005, p. 40) calls this the **signal**), so a large portion of statistics involves minimising the effect of error. The first step is to explicitly define the relationship between the signal and any external error, and how it produces the observed data, y_j:

$$y_j = x(t_j) + \epsilon_j \tag{3.2}$$

If we ignore the error term, ϵ_j, then finding the mathematical representation of the underlying function simply comes down to **interpolation**. Interpolation involves finding a function (**interpolant**) that goes through all the discrete data points, to estimate the data points in between the already known ones. On the other hand, **smoothing** takes into account the error and must compensate for it, instead of just finding a function that goes through all our observations. For the remainder of this chapter, we will ignore the error, and then consider it fully in Chapter 4.

3.1.1 Choosing a set of basis functions

The choice of basis functions, should reflect the basic features of whatever data set is being used. For example, whether the data is periodic, only on a certain interval or whether we have the intention of looking at derivatives. The two bases which together cover most situations are the **Fourier** basis and the **B-spline** basis. The Fourier basis is the well known set of trigonometric functions, which allows us to write basis expansions of the form $x(t) = (c_1 + c_2 \sin(\omega t) + c_3 \cos(\omega t) + c_4 \sin(2\omega t) + \dots)$. As well as the number of basis functions, it requires only one parameter, the periodicity ω of the *cos* and *sin* functions. An advantage to the Fourier basis expansion is its ability to represent periodic functions accurately. Also, fast computation of Fourier coefficients is possible, using methods such as the Fast Fourier Transform. However, Fourier bases struggle to represent functions with high local variation and discontinuities in function value/derivatives.

The curves we will be estimating are not particular periodic, and we can't guarantee that they will have stables curvature and variation. A more flexible basis would be more helpful, and this is exactly what the B-spline basis is. The set of B-splines are piecewise polynomial functions, which can be used to estimate non-periodic functions with strong local features. Each B-spline has an order m (one higher than the highest degree in the polynomial, and a set of *knots* defining the pieces of these piecewise functions. B-spline expansions are also efficient due to B-splines having *compact support*. The next section deals with B-splines, where knots, compact support and more will be discussed in detail.

COVID-19 data could be considered both periodic and non-periodic, depending on the section of the pandemic you are looking at, i.e. if we considered all three "waves" then you could argue the pattern in deaths repeats. However, since my data encompasses just the first COVID-19 wave in the UK and includes some strong features (sharp changes, peaks, and valleys), I will use the B-spline basis. The next section provides a more detailed introduction into what splines are and how they are computed.

3.2 Spline functions and the B-spline basis

Splines are essentially piecewise polynomial functions, that provide a computationally efficient way of representing data which has strong local features. De Boor (2001) provides a comprehensive discussion on spline functions, first introducing the B-spline basis and then spline functions in general. I will introduce them in the opposite order, as I believe this is better for understanding spline properties, whereas De Boor's approach is too heavily algebraic for the scope of this report.

Ramsay and Silverman (2005) describe the structure of a spline function. Here I build on their definition, by converting it into a more explicit mathematical one:

Definition 3.2.1 (Spline function)**.** Let $[a, b] \in \mathbb{R}$ be an interval on the real line, such that it is divided into L disjoint subintervals:

$$[a, b] = [t_0, t_1] \cup [t_1, t_2] \cup \cdots \cup [t_{L-1}, t_L] \tag{3.3}$$

Where t_i are called the **breakpoints** (or knots) of the interval $[a, b]$. An order m **spline function**, $S : [a, b] \to \mathbb{R}$, is defined piecewise such that over each i^{th} subinterval it is a degree $m - 1$ polynomial, $P_i(t)$:

$$S(t) = \begin{cases} P_1(t), t \in [t_0, t_1] \\ \vdots \\ P_L(t), t \in [t_{L-1}, t_L] \end{cases} \tag{3.4}$$

If spline order, m, is greater than 1, then adjacent polynomials, $P_i(t)$ and $P_{i+1}(t)$ must, at each breakpoint, agree in value and derivative up to derivative order $(m-2)$.

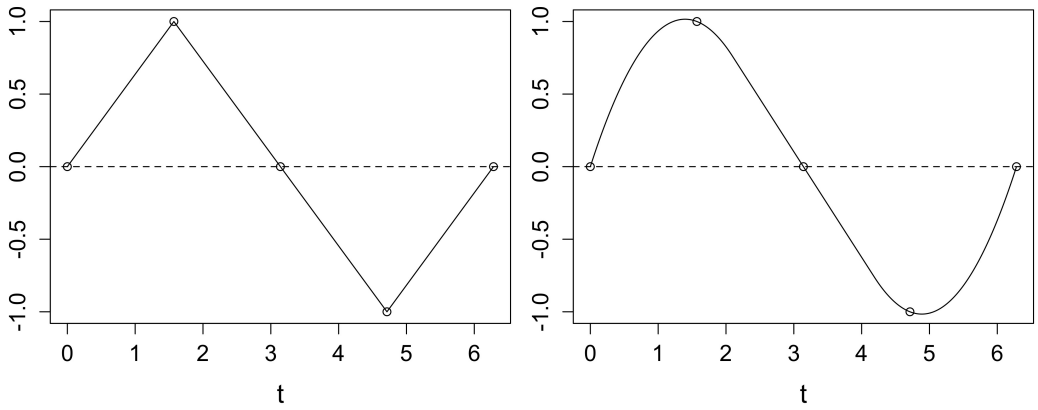

Figure 3.2: Two spline functions of different orders, fit to a sine function. The left panel shows an **order 2** (piecewise linear) spline function, so it must have continuity up to the $(m-2) = 2-2 = 0^{th}$ derivative i.e. continuous but non-differentiable, which is obvious from the plot. The right panel shows an **order 3** (piecewise quadratic) spline function, which will have a continuous first derivative but discontinuous derivatives otherwise.

Figure 3.2 shows an example of how the order of a spline function affects how it looks. Our eyes are only very good at noticing continuity up to the first derivative (without actually just plotting the derivative), hence why a 3$^{\text{rd}}$ order spline is the highest order shown. However, this isn't to say at all that order 4 and higher splines are useless. If we wanted to consider the acceleration of the heights back in Section 1.2.1, we would require an order 4 spline, which will have the continuous second derivative we can plot.

When creating a spline, Definition 3.2.1 tell us we must provide two parameters: A numerical **order**, m, and a set of breakpoints, $\boldsymbol{\tau}$. The term "breakpoint" is used interchangeably with the term "knot" in some texts (Hastie and Tibshirani 1990), however there is a subtle difference which I believe is worth noting:

- **Breakpoints** are the set of **unique** values that define the subintervals of a spline.
- **Knots** are the actual sequence of values that define the subintervals. There can be multiple knots at one breakpoint, but if the knots are distinct, then they have the same definition as breakpoints.

From now on, I will only use the term "knots", since having multiple knots at one point can be beneficial when describing a function which has abrupt changes. Also, having consistent terminology is useful in general.

To summarise so far, we have defined spline functions, which is the overall function that I will use to represent the functional data for each English county (i.e., we will have 78 spline functions for the 78 counties we are considering).

However, in order to represent such a function using only our discrete data, we must provide a set of basis functions for a basis expansion. There are multiple different bases that can be used to represent

spline functions, such as the truncated power series basis (see Hastie and Tibshirani (1990, p. 25)), however the overwhelming favourite is the B-spline basis (In fact, the "B" helpfully stands for basis). B-splines also have very wide programming language support, especially in R.

Curry and Schoenberg (1966, p. 73) provide the original, algebraic definition of B-splines. However, this isn't very informative for the purpose of explicit computation, so will not be stated here. Instead, I will provide a descriptive definition of B-splines and then detail how B-splines are computed using the recursive Cox-de Boor formula (De Boor 2001, p. 89):

Definition 3.2.2 (B-spline basis)**.** Given a set of knots, $\boldsymbol{\tau} = \{t_0, t_1, \ldots, t_i, \ldots, t_L, \}$, the i^{th} **B-spline** of order m, denoted $B_{i,m}(t)$ is:

- A spline function that has compact support (i.e. is positive) over no more than m subintervals.

The *compact support* property of B-splines ensures very efficient computation. As we will see in Chapter 4, when using a basis expansion, we must evaluate our basis functions at certain points. Therefore, if we know in advance that a lot of these values are zero, it will save us a lot of effort.

Properties of B-splines

1. Any spline function, $S_{m,\tau}(t)$ of order m with knot sequence $\boldsymbol{\tau}$, can be written as a linear combination of order m B-splines:
$$S_{m,\boldsymbol{\tau}}(t) = \sum_{i=0}^{L} c_i B_{i,m}(t) \tag{3.5}$$
Hence B-splines act as a **basis** for the spline function space.

2. Order 1 B-splines over a set of knots, $\boldsymbol{\tau}$, are piecewise constant functions:
$$B_{i,1}(t) := \begin{cases} 1, t \in [t_i, t_{i+1}) \\ 0, \text{otherwise} \end{cases} \tag{3.6}$$

3. Higher order ($m > 1$) B-splines are defined recursively using the Cox-de Boor formula:
$$B_{i,m}(t) = \frac{t - t_i}{t_{i+m-1} - t_i} B_{i,m-1}(t) + \frac{t_{i+m} - t}{t_{i+m} - t_{i+1}} B_{i+1,m-1}(t) \tag{3.7}$$
Where t **without** an index represents the argument value (time)of our function, and t **with** an index represents the knot value (e.g. t_i is the i^{th} knot value). Also i represents the first knot value, on the left of the B-spline, where it is zero (see Remark 3 below).

***Remark* 3** (Coincident knots)**.** The Cox-de Boor recursive formula works very straightforwardly when the knots are all distinct. However when creating a full B-spline basis, at the boundaries of our interval we place m knots on top of each other instead of the "normal" 1 (Ramsay and Silverman 2005, p. 51). Essentially, this is because for each extra knot added at a point, the order of the highest continuous derivative of our spline decreases by 1. The reason multiple knots are therefore added at the interval boundary values, t_0 and t_L, is because we do not have information about what our function is doing outside the interval. Therefore we do not want to make **any** assumptions about differentiability at the boundaries, and adding coincident knots allows us to do this.
The downside to this is that Cox-de Boor isn't implemented as easily for B-splines at the boundary/with coincident knots, but it is still possible. This won't be a problem, as R can very easily create a B-spline basis and we will not have to do it by hand.

The proofs of these properties are not particularly informative in the context of this report, however brief details are given below:

Details of proof of B-spline properties

1. Proving B-splines are a basis (property 1) for spline functions comes down to proving that B-splines are linearly independent and span the spline function space (De Boor 2001, p. 98). The spanning property is included in the definition of the spline function space, and linear independence is more technical and won't be discussed here.

2. The recursive Cox-de Boor formula (property 2 and 3) defines B-splines recursively in terms of the previous order B-spline. The i index simply represents the subinterval over which we are computing the B-spline for. Once again, De Boor (2001, p. 90) details how this formula is derived, but in essence it is based on the algebraic definition of B-splines.

Example (B-spline using Cox-de Boor recursion)

We begin by specifying the knot sequence for our B-splines. Here I will use $\boldsymbol{\tau} = (0, 1, 2, 3, 4, 5)$ and aim to compute order 3 B-splines ($m = 3$). I will explicitly show the calculation of $B_{1,3}(t)$, the 3^{rd} order B-spline which first goes to zero at $t_1 = 1$:

Using the notation from (3.6), we can label our knot values: $t_0 = 0, t_1 = 1, t_2 = 2, t_3 = 3, t_4 = 4$ and $t_5 = 5$. Using the same Cox-de Boor recursion equation, the desired B-spline is defined recursively:

$$\begin{aligned} B_{1,3}(t) &= \frac{t - t_1}{t_3 - t_1} B_{1,2}(t) + \frac{t_4 - t}{t_4 - t_2} B_{2,2}(t) \\ &= \frac{(t-1)}{2} B_{1,2}(t) + \frac{(4-t)}{2} B_{2,2}(t) \end{aligned}$$

Now we perform the recursion step and define the two required B-splines of order 2, $B_{1,2}$ and $B_{2,2}$ using the Cox-de boor formula again:

$$\begin{aligned} B_{1,2}(t) &= \frac{t - t_1}{t_2 - t_1} B_{1,1}(t) + \frac{t_3 - t}{t_3 - t_2} B_{2,1}(t) & B_{2,2}(t) &= \frac{t - t_2}{t_3 - t_2} B_{2,1}(t) + \frac{t_4 - t}{t_4 - t_3} B_{3,1}(t) \\ &= \frac{t-1}{1} B_{1,1}(t) + \frac{3-t}{1} B_{2,1}(t) & &= \frac{t-2}{1} B_{2,1}(t) + \frac{4-t}{1} B_{3,1}(t) \\ &= \begin{cases} 0, & \text{if } t < 1 \text{ or } t > 3 \\ (t-1), & \text{if } t \in [1, 2) \\ (3-t), & \text{if } t \in [2, 3) \end{cases} & &= \begin{cases} 0, & \text{if } t < 2 \text{ or } t > 4 \\ (t-2), & \text{if } t \in [2, 3) \\ (4-t), & \text{if } t \in [3, 4) \end{cases} \end{aligned}$$

Where the final lines, use the definition of an order 1 B-spline. Substituting these two expressions into

the one for $B_{1,3}(t)$ gives:

$$B_{1,3}(t) = \begin{cases} 0, \text{ if } t < 1 \text{ or } t > 4 \\ \frac{(t-1)}{2}(t-1), \text{ if } t \in [1,2) \\ \frac{(t-1)}{2}(3-t) + \frac{(4-t)}{2}(t-2), \text{ if } t \in [2,3) \\ \frac{(4-t)}{2}(4-t), \text{ if } t \in [3,4) \end{cases}$$

$$= \begin{cases} 0, \text{ if } t < 1 \text{ or } t > 4 \\ \frac{1}{2}t^2 - t + \frac{1}{2}, \text{ if } t \in [1,2) \\ -t^2 + 5t - \frac{11}{2}, \text{ if } t \in [2,3) \\ \frac{1}{2}t^2 - 4t + 8, \text{ if } t \in [3,4) \end{cases}$$

This final expression is piecewise quadratic, agreeing with the fact the B-spline is order 3. Also, due to it being order 3, each piece will agree in value and first derivative at each knot (see Figure 3.3's right panel for the plot of $B_{1,3}(t)$)

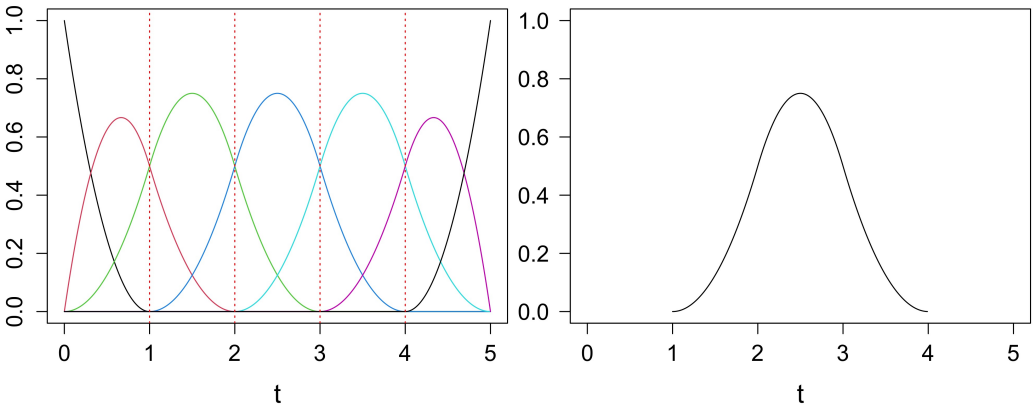

Figure 3.3: Left panel: A full set of B-spline basis functions, of order 3 (quadratic) and with knot sequence $\boldsymbol{\tau} = (0, 1, 2, 3, 4, 5)$. Right panel: One of these B-splines, in particular $B_{1,3}(t)$, which was computed by hand using the Cox-de Boor recursion formula instead of using the *fda* package in R.

Chapter 4

Smoothing functional data

The previous chapter dealt with methods for mathematically representing a whole function, $y_i(t)$, given discrete data $y_{i,j}$, using a set of basis functions $\{\phi_1(t), \phi_2(t), \ldots, \phi_K(t)\}$. However, using only a basis expansion without considering sampling error will lead to a very common problem, overfitting. This means we are fitting our function to the data too closely, and the resulting curve will be too "wiggly". However, in FDA we assume that our functions and, by extension, the data-generating process are smooth. Therefore, we want a method that removes local data variation whilst keeping the general trend in the data. This is where **smoothing** comes in, and any general smoothing method gives us control over how much (local) variation in the data we want to capture in our estimated function.

To make this more concrete, let us return to the example of the B-spline basis. When creating a set of B-splines, James Ramsay, Hooker, and Graves (2009, p. 35) state the number of basis functions should be defined by:

$$\text{Number of basis functions} = \text{Spline order} + \text{Number of interior knots} \tag{4.1}$$

Where "interior knots" are simply the knots (Recall knots are not the unique values, unlike breakpoints, so counting multiple knots at one point is important with this equation.) which are not at the boundaries and in particular with my notation, this excludes knots at t_0 and t_L. Using this relation would be what I will call a "full" basis expansion. Here is where the first smoothing technique presents itself: Using **fewer basis functions**. This method of smoothing is known as **least squares smoothing**, and is the same method used in standard multiple linear regression.

4.1 Least squares smoothing

As a brief recap of standard regression theory, least squares smoothing calculates the coefficients, of the basis expansion (3.1). In particular, for each curve, it calculates the coefficients that minimise the sum of squared errors:

$$\begin{aligned} \text{minimise} \qquad R(\boldsymbol{c}) &= \sum_{j=1}^{J_i}[y_j - \sum_{k=1}^{K} c_k\phi_k(t_j)]^2 \\ &= (\boldsymbol{y} - \boldsymbol{\Phi c})^T(\boldsymbol{y} - \boldsymbol{\Phi c}) \\ \Rightarrow \qquad \hat{\boldsymbol{c}} &= (\boldsymbol{\Phi\Phi}^T)^{-1}\boldsymbol{\Phi}^T\boldsymbol{y} \end{aligned}$$

Where $\mathbf{\Phi}$ is a $J_i \times K$ matrix, that contains each of the K basis functions evaluated at each of our sampling points t_i. Also, y is a column vector of length J_i, containing our functional data i.e. daily mortality, mobility, and positivity.

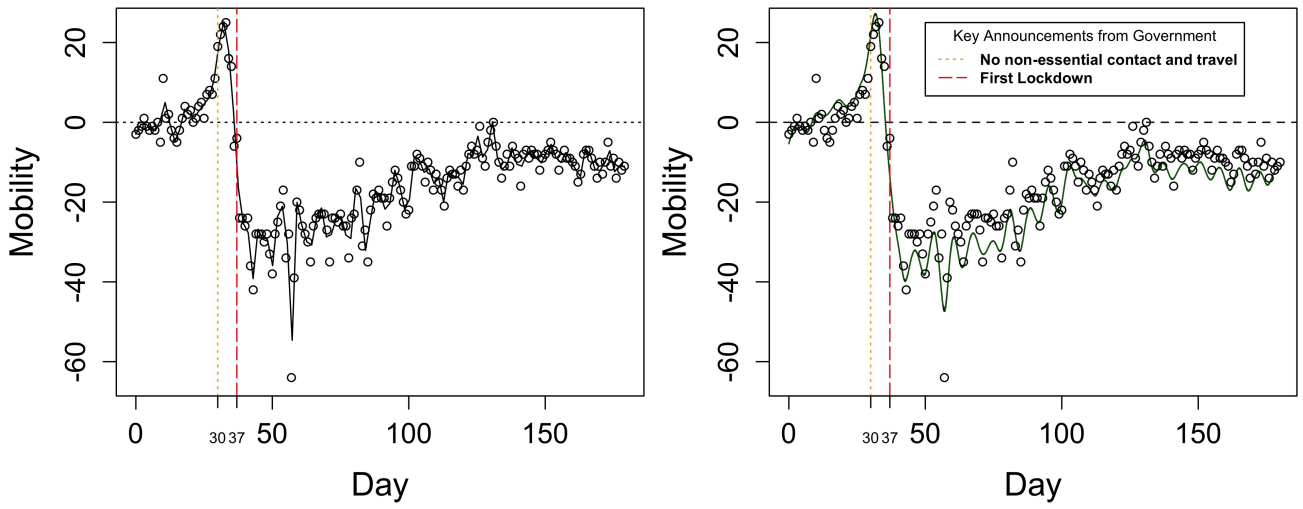

Figure 4.1: Mobility data for County Durham, that has been smoothed using two different methods: Using least squares (left) and with a roughness penalty (right). Using least squares only gives us a discrete choice for the level of smoothness, and hence gives us a rougher curve. Roughness penalties will be discussed later in this chapter, and amongst other benefits, give us a continuous choice for the level of smoothing.

Least squares isn't a bad method to use, it has a nice closed form expression and is easy to implement, however it does have some flaws. Firstly, it assumes every single residual (ϵ_j in equation 3.2) has constant variance. This can be countered by including weighting within our least squares criterion. Second, it only gives us discrete control over the level of smoothing. This is done by lowering the number, K, of basis functions to capture less variance and produce a smoother curve. When working with noisy and dense data, such as the COVID-19 data in this report, we want as much control as possible over smoothness. Therefore, in the next section, I will introduce **roughness penalty** smoothing, which improves on least squares' shortcomings and more.

Note: There are algorithms to choose the "best" value of K if least squares is used, which aim to minimise the mean squared error of our estimated function. They mainly involve standard stepwise methods, as used in regression analysis, but cross-validation can also be used. This K can be considered as a smoothing parameter, and I will focus on algorithms for choosing a different parameter related to roughness penalty smoothing in the next section, which use generalized cross-validation.

4.2 Roughness penalty smoothing

The main goal of smoothing data is to estimate a function that accurately fits the data whilst not displaying too much local fluctuation, since a lot of this "noise" is random and does not help in identifying the general trend. A roughness penalty is a way of quantifying this concept of how much a curve is fluctuating, which allows us to then balance the two aspects of our function: Roughness and goodness of fit. We begin by defining the total **roughness** of a function.

Definition 4.2.1 (Roughness)**.** Given a function $x(t)$ that is twice differentiable over an interval $[a, b]$, the total **roughness** of the function is given by:

$$R_2(x) = \int_a^b [x''(t)]^2 dt$$

The subscript in $R_2(x)$, indicates we are looking at the second derivative of the function x we are estimating. In theory, if we were fitting the derivative of a function, we would use higher derivatives in the roughness calculation. For example, if we wanted to fit the acceleration curve of height, we would integrate the fourth derivative of the original height function.

Although Definition 4.2.1 is the widely accepted definition of roughness, there are other possible options. For example, we could use the maximum value of $|x''(t)|$, however, this is less of an overall measure of roughness, so in my opinion, it is less useful for comparing whole functions. Also, Green and B. W. Silverman (1993), provides an intuitive reason why the integral definition is used: Prior to the development of software for curve drawing, a thin, flexible piece of wood called a "spline", would be bent to match the shape of the curve. The energy stored in the bent spline (strain energy) would be higher for a curve which fluctuates heavily, and the actual expression for strain energy has a leading term proportional to $\int [x''(t)]^2 dt$.

The standard residual sum of squares criterion only considers goodness-of-fit and, if there were no restrictions (such as smoothness) on our function, then minimising RSS would simply result in a curve which goes through every single data point (interpolation). This curve would have incredibly high roughness, which is a feature that we do not want in Functional Data Analysis. Now we have defined a function's roughness, we can expand the standard least squares smoothing criterion to include a **roughness penalty**:

Definition 4.2.2 (Roughness penalised RSS)**.** Given a set of observations $y_1, \ldots, y_{J_i}$, a basis expansion of our function as in Definition 3.1.2 and a real parameter $\lambda > 0$, the **Roughness penalised residual sum of squares** is defined as:

$$\begin{aligned} \mathrm{RP}(\boldsymbol{c}) &= \sum_{j=1}^{J_i} [y_j - \sum_{k=1}^{K} c_k \phi_k(t_j)]^2 + \lambda \int_a^b [x''(t)]^2 dt \\ &= (\boldsymbol{y} - \boldsymbol{\Phi c})^T (\boldsymbol{y} - \boldsymbol{\Phi c}) + \lambda R_2(x) \end{aligned}$$

The first term of $RP(\boldsymbol{c})$ is exactly the same RSS term as used in least squares, written in matrix notation. The parameter $\lambda \in \mathbb{R}_{\geq 0}$, allows us to choose the level of smoothing, i.e. to balance having our function stay close to the given data but also still be a smooth function. Having $\lambda \to 0$, indicates we are penalising roughness less, resulting in a highly variable function that fits the data very well. On the other hand, $\lambda \to \infty$, shows our desire to have a very smooth function since the RSS term (which represents goodness-of-fit) is now insignificant compared to the roughness penalty term. Therefore, when minimising $RP(c)$, this integral term will determine the optimal function. This will naturally be the function with the lowest roughness and hence is the smoothest. By definition, the integral term includes the "true" underlying function $x(t)$, however, as has been the case throughout this report, we do not know this true function, and therefore must use a basis expansion. Obviously, this once again poses the question. If we substitute in the basis expansion, we can write the second term of the criterion in matrix notation as well:

Lemma 1 (Roughness penalty with a basis expansion)**.** *Given a basis expansion, $x(t) = \boldsymbol{c}^T\boldsymbol{\phi}$, the roughness penalty, $R_2(x)$ can be written in matrix notation as:*

$$R_2(x) = \boldsymbol{c}^T K \boldsymbol{c}$$
$$\textit{where} \quad K = \int (\boldsymbol{\phi}'')(\boldsymbol{\phi}'')^T dt$$

K, the **penalty matrix** is a K $\times$ K matrix containing integrals of products of each B-spline basis function, since when integrating a vector/matrix we simply integrate componentwise.

Proof. Let $R_2(x)$ be a roughness penalty as defined before, and $x(t) = \boldsymbol{c}^T\boldsymbol{\phi}$ a basis expansion, where $\boldsymbol{c}$ and $\boldsymbol{\phi}$ are both length K vectors. It is possible to simplify the integral expression for the roughness penalty using some simplex matrix algebra:

$$\begin{aligned}
& & R_2(x) &= \int [x''(t)]^2 dt \\
\text{where} \quad \boldsymbol{\phi}'' = \frac{d^2\boldsymbol{\phi}(t)}{dt}, & & &= \int [(\boldsymbol{c}^T\boldsymbol{\phi})'']^2 dt \\
& & &= \int [\boldsymbol{c}^T(\boldsymbol{\phi}'')\boldsymbol{c}^T(\boldsymbol{\phi}'')] dt \\
\text{using} \quad \boldsymbol{c}^T\boldsymbol{\phi} = (\boldsymbol{c}^T\boldsymbol{\phi})^T = \boldsymbol{\phi}^T\boldsymbol{c}, & & &= \int [\boldsymbol{c}^T(\boldsymbol{\phi}'')(\boldsymbol{\phi}'')^T\boldsymbol{c}] dt \\
& & &= \boldsymbol{c}^T \left(\int (\boldsymbol{\phi}'')(\boldsymbol{\phi}'')^T dt \right) \boldsymbol{c} \\
& & &= \boldsymbol{c}^T K \boldsymbol{c}
\end{aligned}$$

□

The justification for line four of this proof, is based on the fact that $\boldsymbol{c}^T\boldsymbol{\phi}$ is the product of two length K vectors. In particular, it is an *inner* product resulting in a single value. Therefore, the transpose essentially does nothing, and we use the identity as we do above. The alternative to an *inner* product is an *outer* product, an example of which is the integrand of the penalty matrix, K.

Using Lemma 1, we can write the penalised RSS fully in matrix notation:

$$\text{RP}(\boldsymbol{c}) = (\boldsymbol{y} - \boldsymbol{\Phi c})^T(\boldsymbol{y} - \boldsymbol{\Phi c}) + \lambda \boldsymbol{c}^T K \boldsymbol{c} \tag{4.2}$$

Now, we have defined our smoothing criterion, $RP(\boldsymbol{c})$, we can actually compute our smoothed function, $\hat{x}(t)$, as an estimate for the underlying data generating function, $x(t)$. In order to do this we use Definition 3.1.2 of a basis expansion, and calculate an estimate, $\hat{\boldsymbol{c}}$, for the coefficient vector, $\boldsymbol{c}$. This requires some well known results from matrix calculus/algebra. Proofs are omitted for the first two as they aren't very instructive in terms of this project, however for the third property I have included one.

Lemma 2. *a) Given a symmetric matrix, A, and column vector, $\boldsymbol{c}$:*

$$\frac{\partial \boldsymbol{c}^{\boldsymbol{T}} A \boldsymbol{c}}{\partial \boldsymbol{c}} = 2A\boldsymbol{c}$$

b) Given column vectors, $\boldsymbol{a}$ and $\boldsymbol{c}$:

$$\frac{\partial \boldsymbol{a}^T \boldsymbol{c}}{\partial \boldsymbol{c}} = \boldsymbol{a}$$

*c) The penalty matrix, K, is **symmetric***

Proof. Recall the definition of the penalty matrix, $K = \int (\boldsymbol{\phi}'')(\boldsymbol{\phi}'')^T dt$. Where $\boldsymbol{\phi}^T = (B_{1,m}, \ldots, B_{\mathrm{K},m})$ is the vector of order m B-spline basis functions. We can now write the matrix K explicitly as:

$$K = \int \begin{pmatrix} B''_{1,m}(t)B''_{1,m}(t) & \ldots & B''_{1,m}(t)B''_{K,m}(t) \\ \vdots & \ddots & \vdots \\ B''_{K,m}(t)B''_{1,m}(t) & \ldots & B''_{K,m}(t)B''_{K,m}(t) \end{pmatrix} dt$$

$$= \begin{pmatrix} \int B''_{1,m}(t)B''_{1,m}(t)dt & \ldots & \int B''_{1,m}(t)B''_{K,m}(t)dt \\ \vdots & \ddots & \vdots \\ \int B''_{K,m}(t)B''_{1,m}(t)dt & \ldots & \int B''_{K,m}(t)B''_{K,m}(t)dt \end{pmatrix}$$

Since function multiplication is commutative, we can see that this particular matrix K is symmetric. We used the fact that when integrating a matrix, we simply integrate it componentwise. Therefore, to prove in more generality that a penalty matrix is symmetric, we can just consider the integrand, $(\boldsymbol{\phi}'')(\boldsymbol{\phi}'')^T$ and test for symmetry. In particular, for any matrix X, $(X^TX)^T = X^T(X^T)^T = X^TX$, that is, X^TX is symmetric. Therefore, we can conclude that $(\boldsymbol{\phi}'')(\boldsymbol{\phi}'')^T$ is symmetric, and hence K is symmetric. □

Given this setup, we can now compute what our smoothed function is. In particular, what the coefficients are in its basis expansion.

Theorem 4.2.3. *Let $\boldsymbol{y_i} = (y_1, \ldots, y_{J_i})$, be a set of discrete data for a replicate, i, observed at sampling points $t_{i,j}$, for $j = 1, \ldots, J_i$. Let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_K)$ be a set of* K *basis functions, and $\boldsymbol{\Phi}$ be a $J_i \times K$ matrix containing the K basis functions each evaluated at the J_i different sampling points. Then the **coefficients of the basis expansion**, $x(t) = \sum_{k=1}^{K} c_k \phi_k(t)$, that minimise the Roughness penalised RSS, including the penalty matrix, K, are given by:*

$$\hat{\boldsymbol{c}} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda K)^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}$$

Proof. Expand the Roughness penalised RSS, considered as a function of the coefficients $\boldsymbol{c}$:

$$\begin{aligned} \mathrm{RP}(\boldsymbol{c}) &= (\boldsymbol{y} - \boldsymbol{\Phi c})^T(\boldsymbol{y} - \boldsymbol{\Phi c}) + \lambda \boldsymbol{c}^T K \boldsymbol{c} \\ &= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{\Phi c} - \boldsymbol{c}^T\boldsymbol{\Phi}^T\boldsymbol{y} + \boldsymbol{c}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi c} + \lambda \boldsymbol{c}^T K \boldsymbol{c} \\ &= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T\boldsymbol{\Phi c} + \boldsymbol{c}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi c} + \lambda \boldsymbol{c}^T K \boldsymbol{c} \quad (\star) \end{aligned}$$

Using the relation, $\boldsymbol{y}^T\boldsymbol{\Phi}\boldsymbol{c} = \boldsymbol{c}^T\boldsymbol{\Phi}^T\boldsymbol{y}$ since they are each 1×1 vectors, and transposes of each other. Then, differentiating, $\text{RP}(\boldsymbol{c})$ with respect to $\boldsymbol{c}$:

$$\frac{\partial \text{RP}(\boldsymbol{c})}{\partial \boldsymbol{c}} = -2(\boldsymbol{y}^T\boldsymbol{\Phi})^T + 2\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{c} + 2\lambda K\boldsymbol{c}$$

Where we have used results from Lemma 2 to differentiate the expression above. In particular, for the second term of expression $(\star)$, we used result b) with $a = \boldsymbol{y}^T\boldsymbol{\Phi}$. Furthermore, for the third and fourth term, we used results a) and c) from Lemma 2, where we set $A = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$ and $A = K$ respectively.

The minimum value of $\text{RP}(\boldsymbol{c})$ is found by setting this derivative to zero and rearranging:

$$\begin{aligned}
&-2(\boldsymbol{y}^T\boldsymbol{\Phi})^T + 2\boldsymbol{\Phi}^T\boldsymbol{\Phi}\hat{\boldsymbol{c}} + 2\lambda K\hat{\boldsymbol{c}} = 0 \\
&\Leftrightarrow -\boldsymbol{\Phi}^T\boldsymbol{y} + (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda K)\hat{\boldsymbol{c}} = 0 \\
&\Leftrightarrow (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda K)\hat{\boldsymbol{c}} = \boldsymbol{\Phi}^T\boldsymbol{y} \\
&\Leftrightarrow \hat{\boldsymbol{c}} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda K)^{-1}\boldsymbol{\Phi}^T\boldsymbol{y}
\end{aligned}$$

□

***Remark* 4** (Calculating the penalty matrix, K)**.** Recall from the proof of Lemma 2, that the penalty matrix, K, is a matrix of integrals where each integrand is the product of two B-spline basis functions. Computing these integrals can be done using a numerical method, known as the Newton-Cotes formulas (Che et al. 2011, p. 448). When the integrand is a polynomial, as are B-spline functions, the formulas can give an exact answer. If other bases are used, then the answer is an approximation. No further explicit details of this formula will be given, and the integrals (and hence the coefficient vector $\hat{\boldsymbol{c}}$) can be computed easily using software.

4.3 Selecting the smoothing parameter using GCV

So far, we have chosen a basis (B-splines) and observed some discrete functional data. However, the final piece of the smoothing puzzle is the smoothing parameter λ. The term "parameter" somewhat implies that there should be a "best" value of λ that we need to try to estimate. However, in reality, as discussed by Green and B. W. Silverman (1993), choosing the smoothing parameter can be viewed as a free choice, which can simply be **guided** by *automatic* parameter selection methods. The term **hyperparameter** is commonly used in this case. Subjectively choosing different values of λ allows us to compare the features of a curve on different scales. For example, consider the smoothed mobility curve in Figure 2.2. For the chosen value of $\lambda = 1$ in that case, we can see local variation in mobility (possibly attributable to the fact that people tend to go grocery shopping on the weekend, when not working). However, if we were only interested in the overall trend in mobility over the 6 months, we would increase λ to smooth the data more, regardless of what any algorithm tells us to choose for λ.

On the other hand, if we don't have this preconception of how much we want to smooth, or if we are working with many data sets, we can turn to criteria to help us decide. The most well known is the method of Cross-Validation. A variation of this, known as Generalized Cross-Validation, developed by Craven and Wahba (1978), will be the chosen method I use for selecting λ.

Cross-Validation

Cross-Validation is a criterion which picks the smooth function that can most accurately predict "new" observations. If we are only presented with one data set, then the immediate question is: What data do we use to create the smoothed function, and what do we use to test this function? Leave-One-Out Cross-Validation answers this question by fitting the smoothed function to all the data, apart from one data point. This function is then used to predict the left-out value, for which a residual can be calculated. This is repeated for every value in the data set, and the (squared) sum of these Cross-Validation residuals forms the criterion we aim to minimise.

Consider one step of the Cross-Validation method, where we "leave out" the observation y_k. From the remaining data, we estimate a smooth function, denoted $\hat{x}^{[k]}(t)$. We quantify this function's prediction accuracy using a **Cross-Validation score**:

Definition 4.3.1 (Cross-Validation score)**.** Given a vector of discrete data for the i^{th} replicate, $\boldsymbol{y}_i = (y_1, \ldots, y_j, \ldots, y_n)$, where $n = J_i$. Denote by $\hat{x}^{[j]}(t)$, the function that minimises the roughness penalised RSS for all data points excluding y_j. Then the **Cross-Validation score** (CV score) of this data set is defined as:

$$CV(\lambda) \coloneqq \frac{1}{J_i} \sum_{j=1}^{J_i} (y_j - \hat{x}^{[j]}(t_j))^2$$

Note that $CV(\lambda)$ is a function of λ because each "leave-one-out" estimated function, $\hat{x}^{[j]}(t)$, is computed using Theorem 4.2.3 which takes λ as an input. Using the expression for $CV(\lambda)$, we can calculate CV scores for multiple values of λ, and pick the value of λ which minimises $CV(\lambda)$. Using the above expression requires us to smooth a new function every time we calculate the CV score, which is computationally expensive. Therefore, maximising the efficiency of calculating $CV(\lambda)$ is essential. This is what the following proposition, and then Generalized Cross-Validation, will address.

Before we develop Cross-Validation further, it is important to note that "picking" the value of λ minimising $CV(\lambda)$ is not trivial. We cannot guarantee that there is a unique minimum, and numerical minimisation algorithms must be used. Grid-search is the recommended approach in simple cases (Green and B. W. Silverman 1993, p. 30), which is essentially automated trial and error in which the set of λ values to be tested is predefined. Further developments have been made in minimising Cross-Validation scores when there are competing models involved. For example, Moore and Lee (1994) suggest using the RACE algorithm which involves the Bayesian approach of putting a distribution on $CV(\lambda)$, then eliminating models which we are confident are worse than any other model as we gradually include more data points. Although the whole Cross-Validation can be quite time-consuming, it has the huge benefit of not requiring extra data to be collected. This makes Cross-Validation essential in situations where data is limited and hard to collect, such as for the COVID-19 data we are using in this report. This is because the availability of COVID-19 data is essentially dictated by health services, and we cannot count the number of deaths ourselves.

Now, I present a theorem that allows us to calculate Cross-Validation scores without the need to recompute a "leave-one-out" smoothed function every time. To prove the theorem, a brief lemma is required. The proof of this lemma is omitted, but Gu (2013, lemma:3.2) shows the specific details. The eventual proof of Theorem 4.3.3 is also based on a short proof by Gu (2013, p. 68), however I present a full proof including details that he skipped over.

Lemma 3. *The function $\hat{x}^{[j]}(t)$ which minimises the roughness penalised RSS using all values in $\boldsymbol{y}$ apart from y_j, also minimises the "full" penalised RSS where the omitted value y_j is set to $\hat{x}^{[j]}(t_j)$, the value of the "leave-one-out" function evaluated at the argument value t_j*

As a result of this lemma, we can define the fitted values and hence the hat matrix for $\hat{x}^{[j]}(t_j)$, in the same way as we would for a smoothed function that used all data points. This will be an important fact in the proof of the upcoming theorem. Also, we will make use of a well known object known as the hat matrix. For completeness, I provide a definition of it here.

Definition 4.3.2 (Hat matrix)**.** Given a coefficient vector $\hat{\boldsymbol{c}}$ that minimises $\text{RP}(\boldsymbol{c})$ as in Theorem 4.2.3, define the **hat matrix**, H_λ as the matrix that maps the vector of observations $\boldsymbol{y}$ to the fitted values $\hat{\boldsymbol{x}}$. In particular:

$$\begin{aligned} H_\lambda =& \boldsymbol{\Phi}(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda K)^{-1}\boldsymbol{\Phi}^T \\ \text{where } \hat{\boldsymbol{x}} =& \boldsymbol{\Phi}\hat{\boldsymbol{c}} \\ =& H_\lambda \boldsymbol{y} \end{aligned}$$

Theorem 4.3.3. *Let H_λ be a hat matrix. Then we can express the Cross-Validation score, $CV(\lambda)$, for the i^{th} replicate in our data set as:*

$$CV(\lambda) = \frac{1}{J_i}\sum_{j=1}^{J_i}\left(\frac{y_j - \hat{x}(t_j)}{1 - H_{j,j}}\right)^2$$

*Where J_i is the number of sampling points for the i^{th} replicate, $\hat{x}(t_j)$ represents the estimated smooth function (minimising the roughness penalised RSS) calculated from **all** the data, and $H_{i,i}$ represents the i^{th} diagonal element of the hat matrix.*

Proof. We begin by making use of Lemma 3, which allows us to write an expression for the fitted values of our "leave-one-out" function, $\hat{x}^{[j]}(t)$. In particular, for fixed j, we can write $\hat{\boldsymbol{x}}^{[j]} = H_\lambda \widetilde{\boldsymbol{y}}$, where $\hat{\boldsymbol{x}}^{[j]}$ is the vector of fitted values of the estimated function, i.e. with elements $\hat{x}_k^{[j]} = \hat{x}^{[j]}(t_k)$. Also, $\widetilde{\boldsymbol{y}}$ is defined as follows:

$$\widetilde{y}_k = \begin{cases} y_k, k \neq j \\ \hat{x}^{[j]}(t_j), k = j \end{cases}$$

where y_k are simply elements of the original vector of observations $\boldsymbol{y}$. All we have done is put a value back into our data vector where we originally removed one, in particular the function value $\hat{x}^{[j]}(t_j)$, which was the required condition for Lemma 3. Using $\hat{\boldsymbol{x}}^{[j]} = H_\lambda \widetilde{\boldsymbol{y}}$, we can write the j^{th} element of $\hat{\boldsymbol{x}}^{[j]}$ explicitly using the definition of matrix multiplication: $\hat{x}_j^{[j]} = \sum_{k=1}^{J_i} H_{j,k}\widetilde{y}_k$ $(\star)$. Now consider the expression $y_j - \hat{x}^{[j]}(t_j)$, which was the measure of prediction accuracy from the original Cross-Validation score expression.

$$
\begin{aligned}
y_j - \hat{x}^{[j]}(t_j) = y_j - \hat{x}_j^{[j]} &\overset{(\star)}{=} y_j - \sum_{k=1}^{J_i} H_{j,k}\widetilde{y}_k \\
&= y_j - \sum_{k \neq j} H_{j,k} y_k - H_{j,j}\hat{x}^{[j]}(t_j) \\
&= y_j - \sum_{k=1}^{J_i} H_{j,k} y_k + H_{j,j} y_j - H_{j,j}\hat{x}^{[j]}(t_j) \\
&= y_j - \sum_{k=1}^{J_i} H_{j,k} y_k + H_{j,j}(y_j - \hat{x}^{[j]}(t_j)) \\
&= y_j - \hat{x}(t_j) + H_{j,j}(y_j - \hat{x}^{[j]}(t_j))
\end{aligned}
$$

Now, we have the expression:

$$
\begin{aligned}
& y_j - \hat{x}^{[j]}(t_j) = y_j - \hat{x}(t_j) + H_{j,j}(y_j - \hat{x}^{[j]}(t_j)) \\
& \text{rearranging} \implies y_j - \hat{x}^{[j]}(t_j) = \frac{y_j - \hat{x}(t_j)}{1 - H_{j,j}}
\end{aligned}
$$

Substituting this expression into the definition of CV score completes the proof:

$$
CV(\lambda) = \frac{1}{J_i} \sum_{j=1}^{J_i} \left(\frac{y_i - \hat{x}(t_j)}{1 - H_{j,j}} \right)^2
$$

□

This result is extremely helpful, as it saves us a lot of time when calculating Cross-Validation scores. In particular, note the new expression for $CV(\lambda)$ contains the term $\hat{x}(t_j)$ rather than $\hat{x}^{[j]}(t_j)$). Since we are summing over j, we initially had to smooth J_i (the number of sampling points) functions. Now we only have to compute one, i.e. $\hat{x}(t)$, which is smoothed based on the whole data vector $\boldsymbol{y}$ with no "leaving out".

There is however a new obstacle, the diagonal values of the hat matrix, $H_{j,j}$. These are a much less time-consuming set of values to compute compared to before, and can be done using an algorithm developed by Hutchinson and Hoog (1985, pp. 101–104).

Generalized cross-validation

The expression for the *Generalized* Cross-Validation score is similar to the final expression we found for $CV(\lambda)$ in Theorem 4.3.3. In particular, we denote the Generalized CV score by $GCV(\lambda)$. GCV was developed for two main reasons, one computational and one statistical. The computational reason was to avoid having to calculate the diagonal elements of the hat matrix, like we need to for $CV(\lambda)$. We will see that GCV requires only the trace of the hat matrix, which is even easier to find since the trace can be expressed in terms of eigenvalues. However, Green and B. W. Silverman (1993) notes that

this motivation is practically redundant now, since the diagonal finding algorithm from Hutchinson and Hoog (1985) is very effective combined with increasing levels of computational power.

The second motivation for developing GCV relates to the fact that standard Cross-Validation can end up under-smoothing the data (Ramsay and Silverman 2005). GCV has been found to be more reliable in this sense and hence is preferred in this statistical sense. Since CV and GCV are relatively similar, I choose to use GCV for smoothing my data due to its statistical advantage. Now I provide the definition of the GCV score, $GCV(\lambda)$, before applying it to the mobility data as an example of selecting a good smoothing parameter.

Definition 4.3.4 (Generalized Cross-Validation score)**.** Given a vector of discrete data for the i^{th} replicate, $\boldsymbol{y}_i = (y_1, \ldots, y_j, \ldots, y_n)$, where $n = J_i$. Denote by $\hat{x}(t)$, the smoothed function which minimises the roughness penalised RSS, and where H_λ is the corresponding hat matrix. Then the Generalized Cross-Validation score (GCV score) is defined as:

$$\begin{aligned} GCV(\lambda) &:= J_i^{-1} \frac{\sum_{j=1}^{J_i} (y_j - \hat{x}(t_j))^2}{(1 - J_i^{-1}\mathrm{tr}(H_\lambda))^2} \\ &= \left(\frac{J_i}{J_i - \mathrm{tr}(H_\lambda)}\right)\left(\frac{\sum_{j=1}^{J_i} (y_j - \hat{x}(t_j))^2}{J_i - \mathrm{tr}(H_\lambda)}\right) \end{aligned}$$

Where the second expression for $GCV(\lambda)$ is simply found by taking a factor of J_i^{-1} out from the denominator. Note that this definition is almost identical to the one for $CV(\lambda)$, but with the diagonals of the hat matrix swapped out for the trace term.

Example 1. Here, we use the mobility data, introduced in Section 2.1.2. As a brief reminder, the data is for 180 days between February and August 2020, and each data point is a percentage, reflecting the number/duration of visits to grocery shops and pharmacies compared to a baseline before the pandemic began.

Before using Generalized Cross-Validation to find the best value of the smoothing parameter, λ, some manual testing of values can be done, so we can restrict the interval of values we want to test. In the case of mobility data, $\lambda \in [0.1, 1000]$ seems to produce a range of curves that capture the full range of smoothness (at least visually). If we are to produce a plot of the various GCV scores, then an x-axis scale of 0.1 to 1000 would be quite awkward. Therefore, we turn to the tried and tested method of a log scale. In particular, I will calculate $GCV(\log_{10}\lambda), \log_{10}\lambda \in [-1, 3]$, where the smoothing parameter is used in smoothing the mobility data based on a roughness penalty with order 4 B-splines. Although obvious in this case, note that taking the logarithm is well defined in all GCV cases, since $\lambda > 0$ by definition.

The curvature of the GCV plot in Figure 4.2 is relatively low around this minimum, so any value of λ around this point would be an effective smoothing parameter with a low overall GCV score. When I initially smoothed this data, to produce Figure 2.2, I chose $\lambda = 1$, which was close to the optimal value of $\log(\lambda) = -0.3\,(\lambda \approx 0.5)$, but in my opinion produced a slightly nicer curve visually for identifying key curve features.

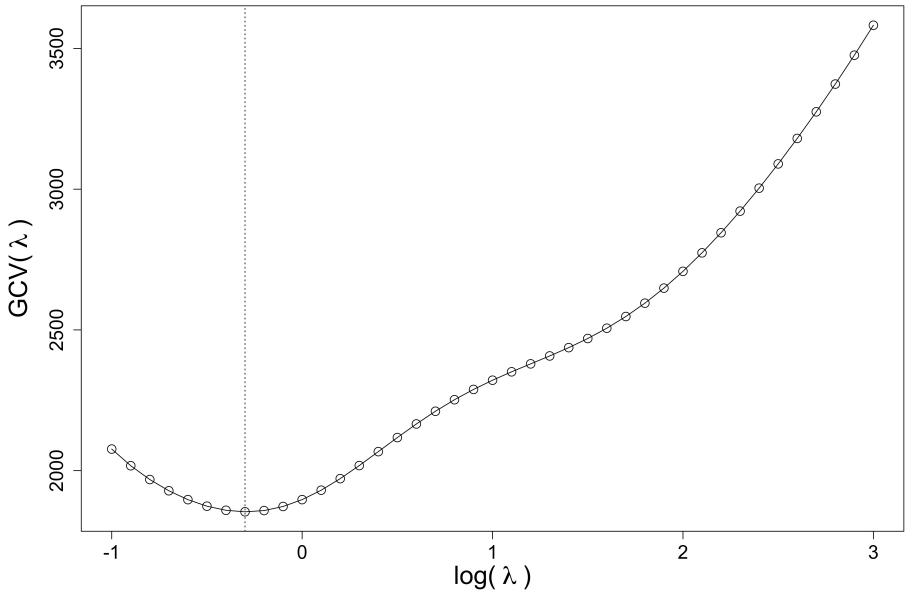

Figure 4.2: Plot of GCV values for the mobility data. The predetermined interval of λ values, $\lambda \in [0.1, 1000]$, the GCV score is calculated for 41 values of λ. For each value of the smoothing parameter, the GCV score is summed over the mobility curves for each of the 78 replicates. The vertical dotted line indicates the rough minimum of these GCV scores, which is found at $\log(\lambda) = -0.3$.

4.4 Adding further smoothing constraints

The smoothing methods discussed so far only assume smoothness of the resulting function. Especially in the case of our mortality data, where there are a handful of zero values at the start, for the function to be smooth it will have to go negative before increasing (Technically we could add extra knots at certain points to allow discontinuity in the first derivative, as discussed in Remark 3, however here we want to find a genuinely nonnegative, fully smooth function). Therefore, in reality we might require further assumptions such as positivity and monotonicity, amongst other things. Smoothing, with these kinds of restrictions, is known as *Constrained Smoothing*. Normally, these constraints are there simply as reality checks, i.e. we can't have a negative number of deaths. However, in some cases, such as if we wanted to estimate a probability density function, the $[0, 1]$ bounds and integrating to 1 conditions are essential to further calculations.

Fortunately, the only constraint we will require is that our function is positive (in particular, for the mortality curves). This method of smoothing is based on the fact that the exponential function is always positive, and that its inverse is the real-valued log function. In particular, for a **positive**, smoothed function $x(t)$ we can define it in terms of an exponential and log:

$$x(t) = e^{W(t)} \tag{4.3}$$

$$= e^{(\boldsymbol{\phi}^T \boldsymbol{c})} \tag{4.4}$$

$W(t)$ is the logarithm of the underlying function $x(t)$. We take the logarithm of our data and then smooth that using the normal roughness penalty, since the log function is not constrained to any positivity.

Therefore, we can expand $W(t)$ in terms of basis functions, as above in (4.4) Then, in order to retrieve the positive smooth function we simply evaluate the exponential function at the values of this smoothed log function, as in (4.3). Since the exponential function is positive everywhere, the resulting smoothed function will also be positive.

The other change we must make is to the roughness penalty criterion. We define the roughness of our smoothed function in terms of this logarithm function, $W(t)$ (Ramsay and Silverman 2005, p. 113), however the RSS term is still defined in terms of the final positive smoothed function. In particular, we adjust the criterion $\mathrm{RP}(\boldsymbol{c})$ as follows:

$$\mathrm{RP}(\boldsymbol{c}\,; W) \coloneqq (\boldsymbol{y} - e^{W(\boldsymbol{t})})^T(\boldsymbol{y} - e^{W(\boldsymbol{t})}) + \lambda \int [W''(t)]^2 dt \tag{4.5}$$

Equation (4.5) can be considered a function of $\boldsymbol{c}$, since $W(\boldsymbol{t})$ is written in terms of a basis expansion with the corresponding coefficient vector, $\boldsymbol{c}$.

When smoothing an unconstrained function, we were able to analytically minimise the fitting criterion to find an expression, $\hat{\boldsymbol{c}}$, for the estimated coefficients. However, in the case of a positive function, this is not possible, and we must use numerical methods. Fortunately, such iterative methods converge quickly.

An example of this positive smoothing method is shown back in Figure 1.2.

Chapter 5

Curve Registration

5.1 Phase and Amplitude Variation

Using the methods discussed so far, we have almost completed all the steps required to create a functional data set, as laid out in Figure 3.1. At this point, we will have a set of smooth curves: 78 in the case of the COVID-19 data. However, before we can perform any functional inference, we must *register* (or *align*) our curves. This is the process of separating phase and amplitude variation, such that when we want to analyse one of these types of variability, no confounding occurs.

Definition 5.1.1 (Amplitude variation)**.** Given two curves $x_1(t)$ and $x_2(t)$, we say that they show **amplitude variation** if the curve features at a particular argument value vary in height.

Definition 5.1.2 (Phase variation)**.** Given two curves $x_1(t)$ and $x_2(t)$, we say that they show **phase variation** if, for a given curve feature of the same size (such as a maximum), the curves vary solely in argument value.

In particular, most registration methods remove phase variation between curves, since phase variation can often be explained by prior knowledge of the situation and intuition, e.g. A growth spurt could in fact just be a period of strong growth followed by weak growth caused by external factors, as opposed to an internal biological clock. Still, this can be seen as an easy way out of analysing phase variation, and it can still be properly analysed if desired (such as by looking at something called the *time warping function*). This is why these methods are called curve *alignment* instead of curve scaling, because we transform the function argument, often nonlinearly, instead of the function value.

Transforming the argument of a function is motivated by concepts known as *system* time and *clock* time (Marron et al. 2015, p. 3). Clock time is simply the rigid measure of time displayed on clocks that we can all agree on (unless you're Einstein!). However, system time is the "body clock" of a dynamical system, whether that be the differing growth rates between children or the timing of the waves of a pandemic in different countries. I think J. O. Ramsay and Li (1998) summarise this concept eloquently, as the situation when the rigid metric of physical time is not relevant to the internal dynamics of a real-life process.

Analysis of both phase and amplitude variation can answer important questions about functional data. For example, for our COVID-19 data, we can ask questions such as: Can we identify individual counties that experienced more severe mortality rates? Was there any lag between peaks in mortality for certain areas? The first question is one of amplitude variation, and the second is about phase. The

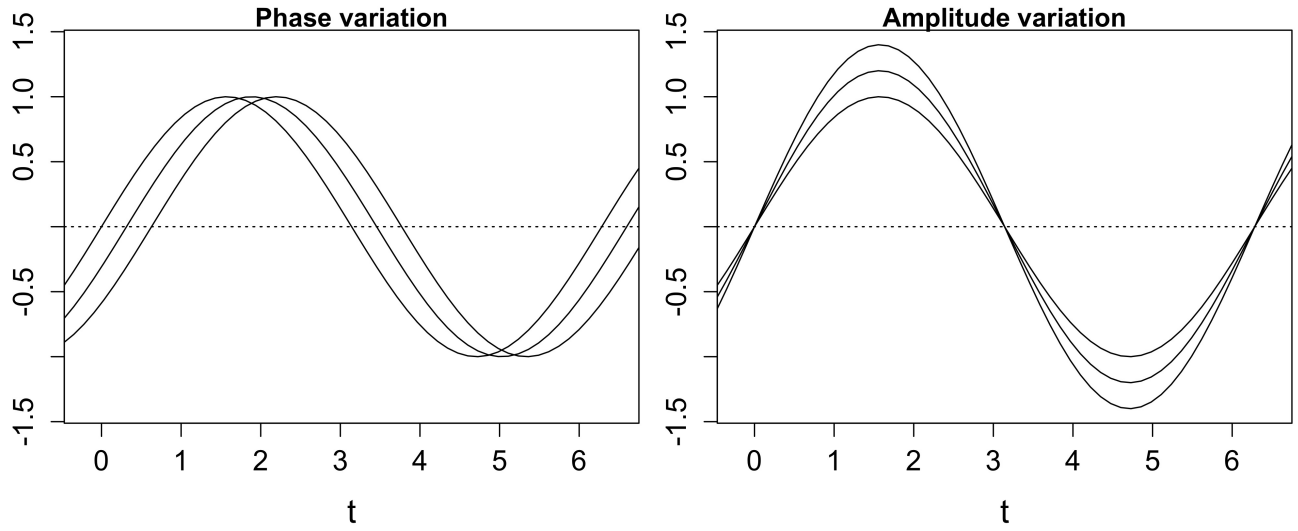

Figure 5.1: A simulated example of phase vs amplitude variation. Left panel: An example of phase variation with three graphs, $\sin(t), \sin(t - 0.1\pi), \sin(t - 0.2\pi)$, that are shifted only in terms of their argument. Right Panel: An example of amplitude variation with three graphs $\sin(t), 1.2\sin(t), 1.4\sin(t)$, which are different by a multiplicative factor which changes the "height" of the curve.

answers to these questions could, for example, be used to inform future government health policy across the country and reduce possible disparities that exist geographically.

The best way to understand the difference in these two types of variation is visually. Figure 5.1 shows the basic difference between phase and amplitude variation, where we can essentially see that phase represents lateral differences and amplitude represents vertical differences. This is all assuming we are working with curves, however FDA can be expanded to work with surfaces, i.e. medical imaging. In this higher dimensional case, the methods of curve registration are not easily expanded, and Modersitzki (2003) discusses the approaches to image registration.

There are a number of challenges when it comes to curve registration, in particular defining what we want an "aligned" set of curves to mean. The simple answer is that we might just want visible features to be aligned, but this is less important than statistical interpretability (Kneip and Ramsay 2008, p. 1155). If we are aligning curves simply for them to "look nice", then there hasn't been much point in developing all of these FDA methods in the first place. Therefore, registration methods must give curves that produce useful models and summary statistics. A good registration method is known as a *consistent* method, which we will define but not talk much more about, since proving a method is consistent isn't particularly enlightening in terms of this report. Fortunately, all registration methods we work with will be consistent.

Definition 5.1.3 (Consistent registration procedure)**.** Given a set of unregistered functions, $\mathcal{X}$, all of which have similar features. A registration procedure, $\mathcal{R}$, is **consistent** if the following two conditions hold:

- The set of registered functions outputted by $\mathcal{R}$ is a convex subspace of $\mathcal{X}$
- Performing the registration procedure on an already registered function has no effect.

The definition of $\mathcal{X}$, as the set of functions with "similar" features is obviously not rigorous and a fully rigorous one is provided by Kneip and Ramsay (2008, pp. 1156–1158). A good example, to illustrate what the above definition means, is to consider a set of positive functions with one peak each. The set of registered functions must also be positive with one peak, but in particular a weighted average of any two of them must also be a positive function with one peak (hence **convex** subspace). The second condition is more self-explanatory and ensures that our registration is optimal.

The convexity condition in Definition 5.1.3 is one of the key factors that allows our curves to have *statistical interpretability*, as I mentioned above. An important example is the mean curve of a set of functions: A consistent set of aligned curves will have a mean curve that has the same general features as the initial curves, hence it will be a much better reflection of the average trend. As well as this, if we do not remove phase variation, functional regression fits are negatively affected (Marron et al. 2015, p. 4), which is important to note since we will be using functional regression analysis later on.

Least squares registration

The most basic registration method is known as *shift* registration, and simply involves translating a curve horizontally by a constant so that a given curve feature occurs at the same *clock* time for all the curves. A criterion can be devised, similar to the roughness penalised RSS for smoothing curves, that registers a function by minimising the squared differences between the aligned curve and mean curve of the sample. For this reason, this method is typically referred to as *least squares* registration. It can be shown that this method breaks down, i.e. stops being a *consistent* registration method. An example of this happening can be seen in Figure 5.2. For this reason, I will not go into any more details/apply this method any further in this report. Instead, I will focus on *landmark* registration and *continuous* registration, which are much more robust.

5.2 Landmark registration

Recall, the general goal of curve registration is to separate amplitude and phase variation in a set of curves, such that the two can be analysed separately. For the case of landmark registration, we do this by transforming the argument of these functions such that a chosen, discrete number of landmarks of the functions all occur at the same time. In particular, we want to find a non-linear function, $h_i(t)$, called a time warping function such that a registered function, $x_i^\star(t)$ is given by:

$$x_i^\star(t) = x_i(h_i(t)) \tag{5.1}$$

What we define as a *landmark* of a curve is entirely up to the user. This is one of the benefits, but also a huge drawback, of landmark registration as a method. Typically, landmarks are features of a curve such as extrema or zero values, however, with a very variable set of curves it can be too difficult and too time-consuming to identify all the landmarks to align the curves around. When landmarks can be identified, landmark registration is an effective method of aligning curves, but when they cannot be fully identified, we must turn to other methods. Gasser and Alois Kneip (1995) developed a method to more efficiently identify landmarks, using something known as the *structural intensity* of a function, however this doesn't prove particularly useful now with the invention of continuous registration procedures (where the whole curve is used in the alignment). In fact, Ramsay and Silverman (2005, p. 142) suggests that, from experience, a combination of the two methods is most effective. In particular, you would begin with landmark registration using obvious landmarks, then proceed to continuously register the output to deal with the remaining, less obvious phase variation.

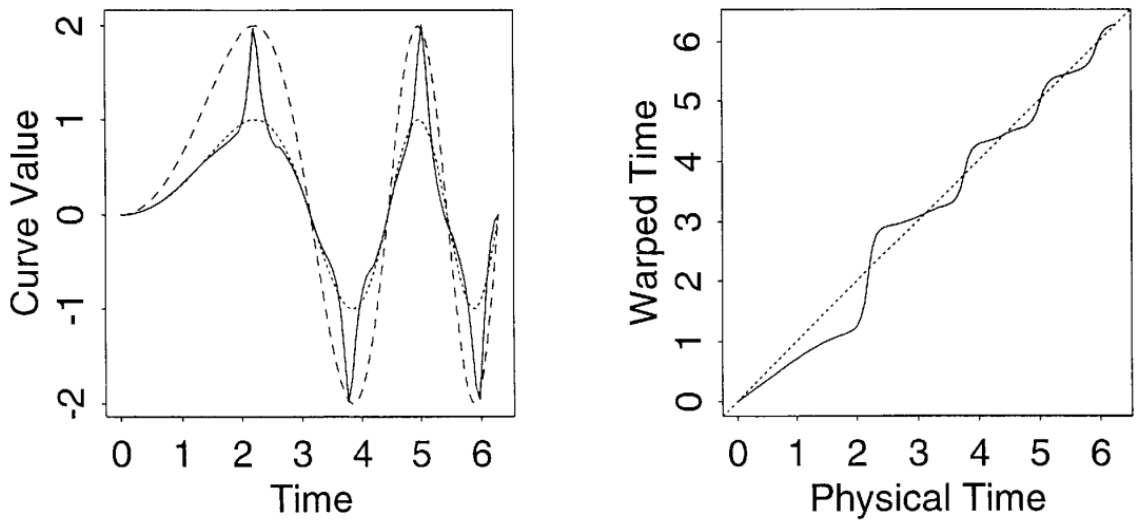

Figure 5.2: Example of trying to align a curve which only shows amplitude variation, using least squares shift registration, from J. O. Ramsay and Li (1998). Left panel: The dashed line is the one to be registered, with the dotted line being the target curve. There is only amplitude variation present in this dashed curve, and the solid line represents what it looks like after being "aligned". Clearly, this aligned curve has been distorted in ways we don't want from a registration procedure. Right panel: The corresponding time warping function.

Having introduced the concept and practical aspects of landmark registration, I now turn to the explicit details of how we find the time-warping function $h_i(t)$. Firstly, I use a definition of the time-warping function provided by Ramsay and Silverman (2005), but with extra details added to make the concept as clear as possible.

Definition 5.2.1 (Time-warping function)**.** Let $x_i(t)$ be a smooth function over an interval $[a, b] \in \mathbb{R}$ with a set of values $\boldsymbol{t}_i^\star = (t_{i1}, \ldots, t_{il}, \ldots, t_{iL})$ corresponding to the argument values of L landmarks of $x_i(t)$. Denote by t_{0l} the argument value of the mean function's corresponding l^{th} landmark. The **time-warping function** of $x_i(t)$ is a continuous function, $h_i(t)$ satisfying the following properties:

1. $h_i(a) = a$
2. $h_i(b) = b$
3. $h_i(t_{0l}) = t_{il}, \forall\, l \in \{1, 2, \ldots, L\}$
4. $h_i(t)$ is monotonically increasing.

The justification for $h_i(t)$ being monotonic is simply based on the fact that time flows in one direction. If any part of the time-warping function was decreasing, then we could find non-transformed time values, $t_1 < t_2$, such that their transformed counterparts $t_1^\star$ and $t_2^\star$ are the opposite way round, i.e. $t_1^\star > t_2^\star$. This would mean that transformed time would go back on itself at some points, which in itself is confusing, but also contradicts the fact that time flows in one direction. Based on this argument, we could have $h_i(t)$ being monotonically decreasing. However, this would mean the time axis would get flipped when

aligning curves, which isn't intuitive or helpful towards our goal. To cut a long story short, assuming monotonicity is essential and saves us a lot of trouble.

In order to explicitly find such a function, we can use linear interpolation of the points in Definition 5.2.1. As always, using linear interpolation is a quick and easy method of curve fitting, but it still comes with its flaws. In particular, a linearly interpolated $h_i(t)$ is essentially a discrete approximation of a fully continuous, $h_i(t)$, which takes into account every point of the original function. Therefore, we are ignoring what happens in between landmark points, and hence cannot be satisfied with this method.

The next step up from this, is to use smoothing methods, based on the constrained smoothing discussed in Section 4.4. By definition, h_i is a monotonically increasing function and, by slightly altering the method for smoothing a positive function, we can smooth a monotonic function.

To make this more concrete, recall the expression (4.3) for a positive smoothed function: $x(t) = e^{W(t)}$ where $W(t)$ is the logarithm of the underlying data generating function. A monotonically increasing function is equivalent to a function with an everywhere positive first derivative, therefore we can rethink the monotonic smoothing problem as a positive smoothing problem. In particular, we can rewrite (4.3), using $h(t)$ to match the notation of the time-warping function:

$$\frac{d}{dt}h(t) = e^{W(t)} \tag{5.2}$$

Then we can integrate the expression (5.2) with respect to t to produce an expression for our monotonic time-warping function, $h(t)$:

$$h(t) = C_0 + C_1 \int_0^t e^{W(s)} ds \tag{5.3}$$

Where C_0 and C_1 are constants. In this case, we smooth the logarithm function using the same criterion (4.5), and then retrieve our smooth time-warping function using (5.3). Since (5.2) can also be considered as a first order differential equation, we determine the value of these constants in the standard way using initial conditions. In particular, we have boundary conditions for the values of $h(t)$ on the interval we are considering, as in Definition 5.2.1.

Any given time-warping function essentially tells us whether an argument value must be brought forwards or backwards in time in order to be registered. Each function will be a curved line which either bends above or below the diagonal, and a simple visual inspection can tell us a lot about the phase variation of the curves.

Figure 5.3 shows an application of landmark registration to 4 of the 78 mortality curves, where I have only used four of the curves, so the resulting alignment is easier to notice. In particular, I registered the first derivative of the mortality curves based on four landmarks: The first two crossings of zero, and the first prominent maximum and minimum. The registered curves are shown on the right of Figure 5.3, where the alignment of these four landmarks is clear. The rest of the function, from around day 75 onwards, has definitely improved in its alignment but is not perfect. Identifying landmarks in the second half of the interval is much more difficult, and we can see that the four curves are clearly not well aligned after the minimum around day 70. Therefore, the use of a continuous registration method would be helpful here. The next section will provide details of how this is done.

Furthermore, the four warping functions are shown in Figure 5.4, computed using expression (5.3) with a B-spline basis of order 4 for the $W(t)$ logarithm function. All four are reasonably close to the diagonal, where the diagonal line represents the identity warping function. This implies that only a small amount of alignment was required overall. However, we can see that two of the functions were

required to be shifted back in time, since their warping functions bend below the diagonal. The other two were transformed forward in time since their warping functions bend above the diagonal.

The reason why I smoothed the first derivative is motivated by the fact that manual identification of zero values is easier than identifying the timing of maxima and minima, since extrema of the original curve correspond to zeros of the first derivative. From the aligned set of curves, we can tell that the fastest increase in mortality was around 43 days into the UK pandemic (at least in the four counties: Bath and North East Somerset, Bedford, Blackburn with Darwen and Blackpool).

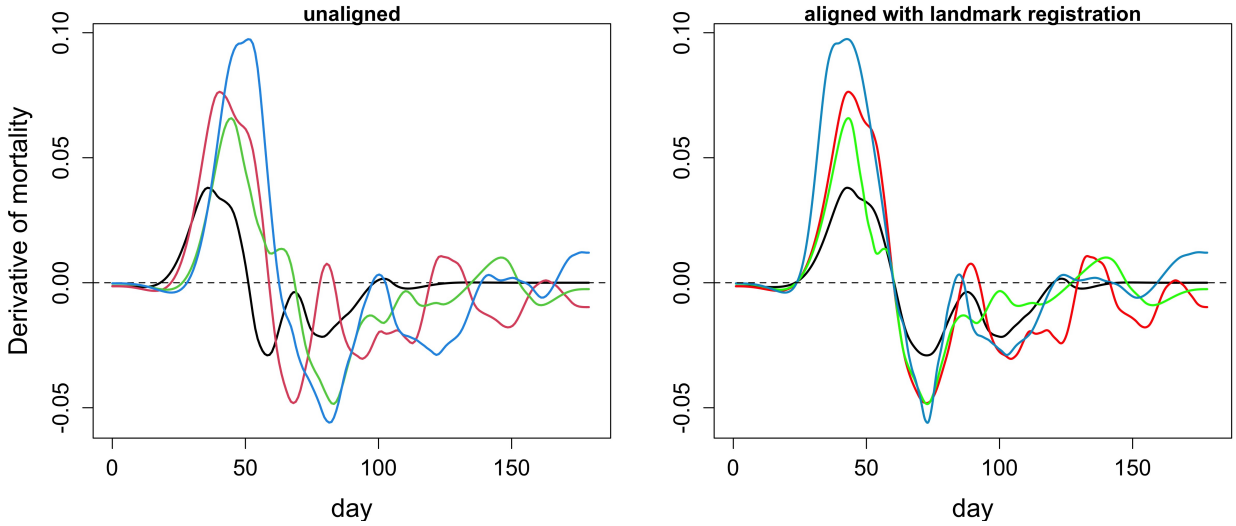

Figure 5.3: Four mortality curve aligned via landmark registration. The curves are coloured, where the matching coloured curve on the right is the registered version of the original on the left. Left panel: Using the mortality curves we have smoothed already (roughness penalty with $\lambda = 800$ selected by GCV), the smoothed first derivative of the first four counties' mortality has been plotted. Right panel: The result of applying landmark registration with four chosen landmarks: The first two crossings of zero, as well as the first prominent maximum and minimum. Then the time-warping functions, see Figure 5.4, are used to transform the time arguments of the unaligned functions, to produce the aligned set of curves.

5.3 Continuous registration

Continuous registration is a method that takes into account the whole curve that is to be registered, instead of just discrete landmarks. It is similar to least squares registration in the fact that we will end up minimising the value of a criterion. However, in this case, the criterion is based on principal component analysis. The main benefit of continuous registration will, in fact, be that it is automatic and does not require the tedious landmark identification from the previous method.

It was somewhat implicitly assumed in the previous section that we align our functions to a *target* function. We must make this explicit in this case by denoting the target function by $x_0(t)$. In particular, the mean function of the unaligned curves will be used as the target function in all cases. The choice of the target function is mainly down to the user and the specifics of the problem, and Marron et al. (2015, p. 7) provides some examples of when a function other than the mean might be used. A slightly more advanced approach, involving the mean function, would be to form a *Procrustes* iteration. The initial

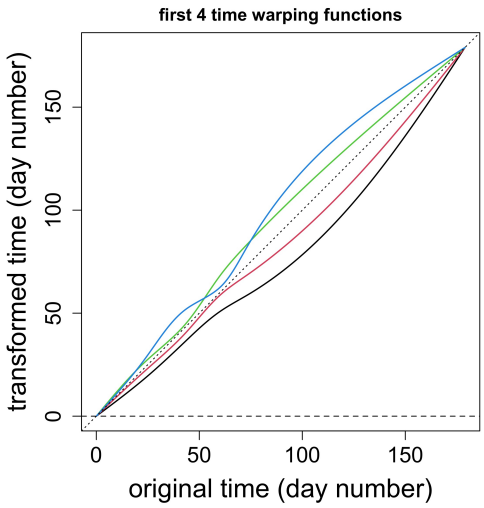

Figure 5.4: Four time warping functions, corresponding to the four, landmark registered, mortality derivative curves in Figure 5.3 with colours matching the warping function to its corresponding registered curve. A dotted line of gradient 1 has been added to represent the identity warping function. Two of the functions are below the dotted line, representing time being slowed down to match the target function. For the two functions above the diagonal, time has been accelerated to match the target function.

step would use the mean of the unaligned curves as the target function, then a new mean function would be calculated from the aligned functions. The iterations proceed by setting the next target function to be the new mean function until there are no visible improvements.

To derive the continuous registration criterion, we begin by considering the target function, $x_0(t)$, and a registered function, $x_i(h_i(t))$ (using the notation from (5.1)). If these two functions were to agree everywhere, we would consider the function $x_i(h_i(t))$ to be perfectly *aligned* to the target function. We could produce a scatter plot of their values against each other, and this would like a perfect straight line, with gradient 1. However, now consider the case where the functions are not registered, and producing a similar scatter plot would result in a more random spread of points. Another way of phrasing this is that the first set of points are one dimensional, and the second are two-dimensional. The general goal of continuous registration is to get two functions as "close" to this one-dimensional case as possible, and a simple, simulated example of this is shown in Figure 5.5. We achieve this by using the well known method of principal component analysis.

Principal component analysis allows us to determine the optimal number of dimensions required to represent a set of data, as well as quantify how much variation in the data is captured in each dimension. Explicitly, this is done by calculating the eigenvectors and eigenvalues of the covariance matrix of the data. Therefore, in order to apply PCA to our functional data, we need a functional equivalent of this covariance matrix. Ramsay and Silverman (2005, p. 140) suggest using a version of X^TX, where X is the $n \times 2$ matrix containing the values of the target and the registered function, evaluated at a mesh of n points. In particular, X^TX would be a 2×2 matrix with each element containing a long sum. The functional version simply swaps out these sums for integrals, reflecting the fact that we are working with functions. The functional version of X^TX is denoted by $C(h)$ and is defined as follows:

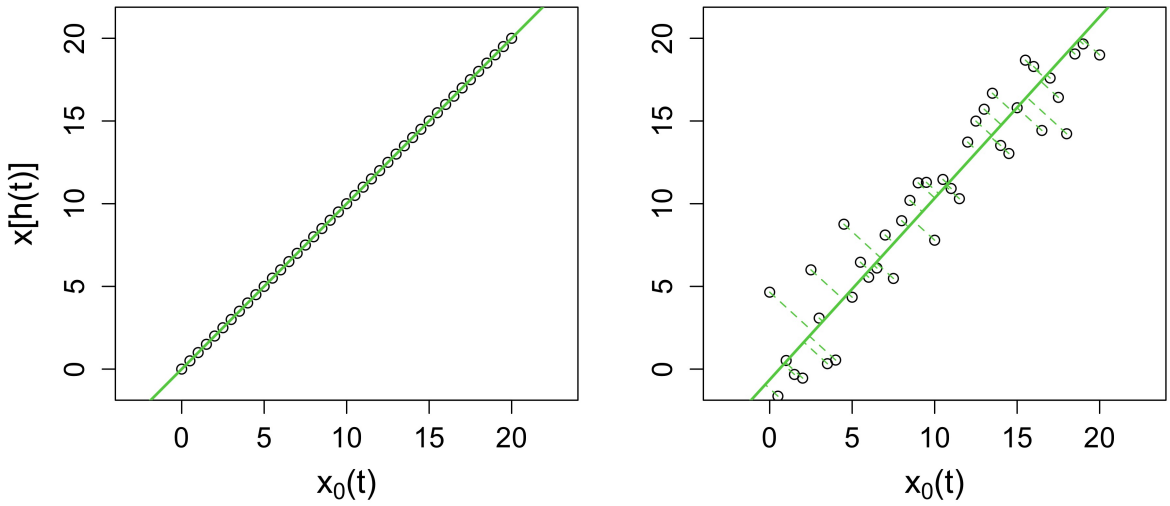

Figure 5.5: A simulated example visualising the use of principal component analysis in continuous registration, where the green lines the first principal components of each sets of values. $x_0(t)$ represents the target function and $x[h(t)]$ is the registered function using a time-warping function h. Left panel: A perfect registration, where the values of the target and the registered function perfectly agree. As a result, only one principal component completely describes the data, and the second eigenvalue will be zero. Right panel: The values of $x_0(t)$ are the same as in the left image, however the values of the "registered" function have been simulated by adding a random normally distributed error (mean = 0, standard deviation = 2) onto the values from the left image. The dashed lines are simply the perpendicular projections onto the first principal component, showing that a second principal component would be required in this case.

$$C(h) = \begin{pmatrix} \int (x_0(t))^2 dt & \int x_0(t)x[h(t)]dt \\ \int x_0(t)x[h(t)]dt & \int (x[h(t)])^2 dt \end{pmatrix} \tag{5.4}$$

The justification for using $X^T X$ is not given by Ramsay and Silverman (2005), however, I describe the reason we use it here. It is based on a key step of PCA known as *rescaling*, which standardises the mean of the data to zero and the standard deviation to one. The following proposition proves that we can use $X^T X$ as a representation of the covariance matrix:

Proposition 5.3.1. *Let X be an $n \times m$ matrix representing n values of m random variables, $X_1, \ldots, X_m$. If all the random variables have mean zero, then:*

$$\Sigma = \frac{X^T X}{n-1} \tag{5.5}$$

Where Σ denotes the $m \times m$ covariance matrix of the random variables.

Proof. Recall that for the m vectors, $X_1, \ldots, X_m$, containing n data points each, we can estimate the elements of the covariance matrix of the data using the following formula:

$$\Sigma = \frac{1}{n-1}\sum_{i=1}^{m}(X_i - \bar{X})^T(X_i - \bar{X})$$

Where $\bar{X}$ represents the mean vector, and we divide by n-1 to achieve an unbiased estimate. However, by assumption this mean vector is the zero vector, therefore our formula simplifies to:

$$\begin{aligned}\Sigma &= \frac{1}{n-1}\sum_{i=1}^{m} X_i^T X_i \\ &= \frac{X^T X}{n-1}\end{aligned}$$

□

The scalar factor of $\frac{1}{n-1}$ does not present a problem, since multiplying a matrix by a scalar simply scales all its eigenvalues by that same factor. The reason this is not a problem for our continuous registration will become clear shortly.

Now that we have a functional equivalent of a covariance matrix, we can perform PCA. In the case of a registered function that is perfectly aligned to the target function, the second (when the eigenvalues are ordered from largest to smallest) eigenvalue of $C(h)$, denoted by λ_2, would be zero. The zero eigenvalue reflects the fact that we require only one principal component to represent the values of the two functions (see the left panel of Figure 5.5). In any other case, this second eigenvalue will be nonzero. The goal of continuous registration is to find a time-warping function, $h(t)$, that produces a registered function that results in a minimum value for the second eigenvalue, λ_2, of $C(h)$. That is quite a long description of continuous registration, and we can describe it more succinctly with a mathematical criterion:

$$\text{minimise} \quad EIG(h) = \lambda_2[C(h)] \quad \text{w.r.t. } h(t) \tag{5.6}$$

Where $\lambda_2[C(h)]$ is the second eigenvalue of $C(h)$. Since we are finding the h that minimises $EIG(h)$, the actual value of the eigenvalue does not matter as long as it is the minimum. Therefore, the scalar factor $\frac{1}{n-1}$ will not change the optimal time warping function given by the criterion (5.6).

$EIG(h)$ currently does not impose any restrictions on the time-warping function. However, we would like it to be smooth and monotonic as it was for landmark registration. Therefore, we assume that $h(t)$ can be expressed in the same form as in expression (5.3) to satisfy the monotonicity requirement. Then, to have h smooth, we smooth the logarithmic function $W(t)$ in terms of a basis expansion as we did for landmark registration. As a result, we expand $EIG(h)$ to include a roughness penalty with a smoothing parameter λ:

$$EIGR(h) = \lambda_2[C(h)] + \lambda \int (W''(t))^2 dt \tag{5.7}$$

Where $EIGR(h)$ represents a roughness penalised version of the $EIG(h)$ criterion. In (5.7) distinguish between the eigenvalues of $C(h)$, λ_1 and λ_2, and the smoothing parameter, λ, using the notation where eigenvalues possess a subscript index and the smoothing parameter does not. Also note, $W''(t)$ is what we defined as the roughness of $W(t)$ in Definition 4.2.1, and $\lambda \in [0, \infty) \subset \mathbb{R}$.

The continuous registration process is an iterative, numerical technique, so can take a lot longer to do than landmark registration. However, both methods give curves that are much better aligned compared

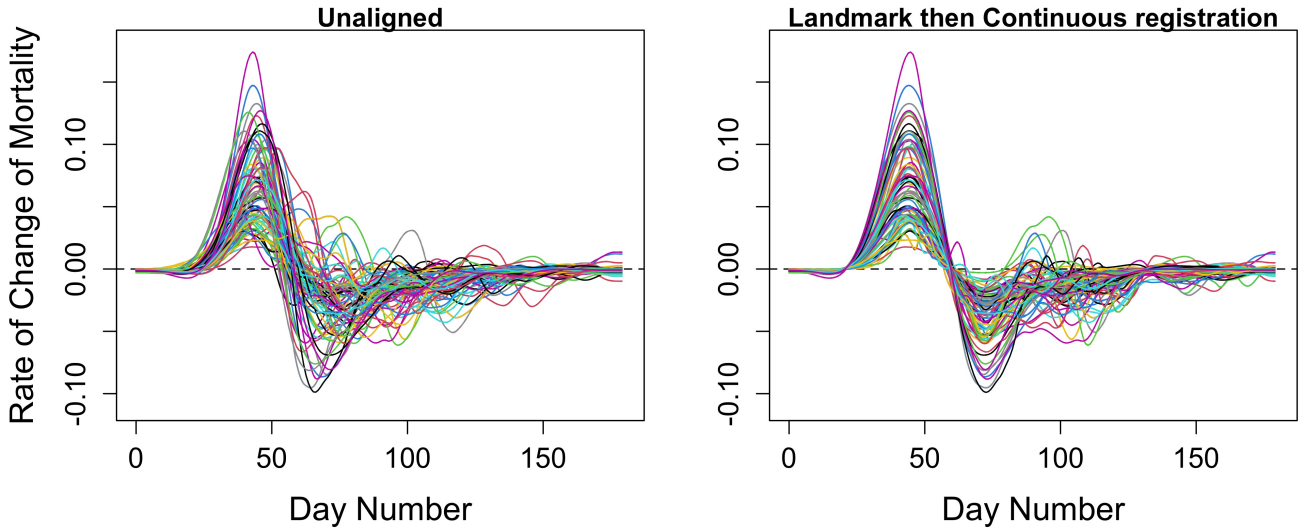

Figure 5.6: A comparison of the effects of landmark combined with continuous registration on all 78 mortality first derivative curves. Left panel: The 78 curves are smoothed using a roughness penalty, with a smoothing parameter $\lambda = 800$ (selected using GCV), the curves are then plotted with no registration. Right panel: The 78 curves are then aligned using landmark registration, with the same four landmarks as before: Two zero crossings and the largest maximum and largest minimum. Continuous registration is then applied as well, producing the final plot.

to how they started. Therefore, continuous registration is normally the method chosen due to its automatic nature. From my experience of applying these two methods, manually selecting 4 landmarks for each of the 78 curves took much longer than the continuous registration process. Regardless, since time is not a limiting factor in my case, I perform both methods in succession.

Continuing the example of the mortality derivative curves, the results of the combined registration method on all 78 curves can be seen in Figure 5.6. Amplitude variation at a given time can be much more easily compared between curves now, and phase variation has been minimised. From around day 80 onwards, there is still an element of non-alignment, however it was much more difficult to pick out landmarks around these points. Despite this, ignoring a few anomalous curves, the overall alignment is definitely sufficient for further analysis. In order to improve this registration even further, we could attempt to select 2 or 3 landmarks on the right-hand sides of each curve. However, the benefits would not be that significant for the extra time it would take, and I am happy with the registration already achieved.

Recall that one of the benefits of performing curve registration is the more accurate computation of mean functions. We conclude this chapter by producing two mean functions, shown in Figure 5.7, for the mortality data we have been working with in this chapter. Looking at the left curve in Figure 5.7, we can see that, on average, mortality (number of deaths per 100,000 people) peaked around day 58 in the UK (exactly three weeks after the first lockdown began). If we allow roughly two weeks of lag between COVID-19 symptom onset and death (Harrison, Docherty, and Semple 2020, Table 1.), this result implies that the lockdown came into effect at a reasonably correct time. However, this 2 week lag is only a median value, and also we have not considered time lag in death registration and the time from catching COVID-19 to symptom onset. As a result, the timing of the lockdown might have actually

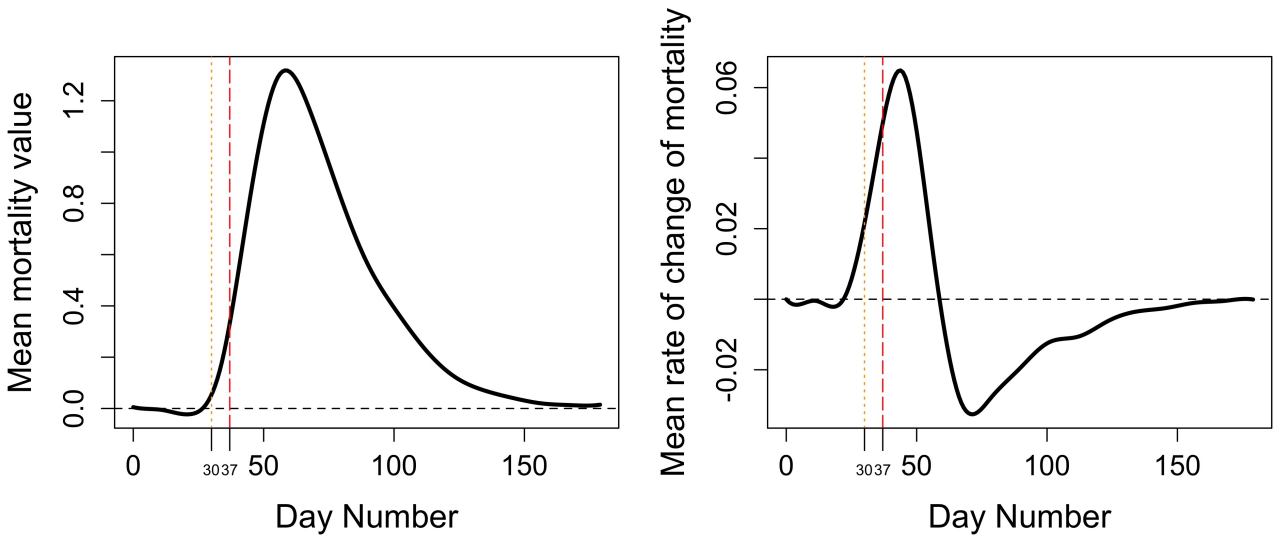

Figure 5.7: The mean functions of the aligned mortality curves (left) and the mean of the aligned first derivative curves of mortality (right) over all 78 counties. Functions were smoothed using a roughness penalty with smoothing parameter $\lambda = 800$. A combination of landmark and continuous registration was used to align the curves. From the aligned set of curves, the pointwise mean functions were then computed.

been even slightly late. In either case, further analysis would be beneficial, in the form of other mean functions or perhaps regression analysis.

First, we consider the former of these options with the mean *derivative* curve. We can see a maximum in the rate of change in mortality on day 44 and a minimum on day 72. The peak represents the point at which deaths were rising the fastest, which in this case was 1 week after the lockdown began. The lockdown started before this peak, however the rate of increase in mortality was still comparatively high to normal. What we can see is that reasonably quickly, by day 72, the derivative value drops to a negative valued minimum. This means deaths were decreasing, and continued to decrease, giving us the impression that the first lockdown had an overall positive effect on lowering deaths.

Chapter 6

Further exploratory FDA methods

We have now completed all the steps laid out in Figure 3.1, to go from discrete observed data to a functional data set that we can analyse using FDA. In the process of smoothing and aligning the curves, we were able to perform some descriptive data analysis as we went along, however with our processed data set it is possible to perform further FDA methods. In the next chapter, I will perform functional regression analysis, where we look for relationships between multiple functional data sets. However, in this chapter, I will compute two more exploratory statistics: Covariance functions and functional band depth. The second of which will be used to create functional equivalents of a box plot.

6.1 Covariance functions

Functional covariance has a similar interpretation to standard covariance as the strength and direction of the linear relationship between two values. However, in FDA, we must distinguish what we are analysing the covariance of. We can compare two points within a single function or two separate functions, each at one point. Both cases produce a form of covariance *function* that can be evaluated at two argument values. The first option produces what is known as a *covariance function*, and the other is a *cross-covariance function*. For the sake of space, I will focus on the *covariance function*.

Definition 6.1.1 (Covariance function)**.** Given a random function $x \in L^2$, over the interval $[a, b] \in \mathbb{R}$ the covariance function $cov : [a, b] \times [a, b] \to \mathbb{R}$ is defined as:

$$\text{cov}_x(t_1, t_2) = \mathbb{E}[(x(t_1) - \mu(t_1))(x(t_2) - \mu(t_2))] \tag{6.1}$$

Where $\mu(t)$ is the mean function of x, and L^2 is the space of square integrable functions.

This definition is based on one provided by Horváth and Kokoszka (2012, p. 26), and involves the Hilbert space L^2. It looks similar to the well known definition of covariance for two random variables, however we are now considering how values of one function vary with each other at different argument values. Since we are working on the application of these methods to a data set, instead of the heavy theoretical side, it would be more useful to have a sample estimated version of (6.1) which we can plug our data into.

The functional version of sample covariance, $\hat{\text{cov}}_x(t_1, t_2)$, is derived in the same way as in the multivariate case:

$$\hat{\text{cov}}_x(t_1, t_2) = \frac{1}{N}\sum_{i=1}^{N}(x_i(t_1) - \bar{x}(t_1))(x_i(t_2) - \bar{x}(t_2)) \tag{6.2}$$

Where $\bar{x}(t)$ is the mean function of the N curve replicates in the data set. As normal, the hat over $\hat{\text{cov}}_x(t_1, t_2)$ represents that the function is a sample estimate. In the case of *functional* covariance, there isn't much to gain from standardising it to a correlation function. Since we are comparing points within a single function, the scale will always be the same and the value of covariance will be interpretable. This reasoning does not work with *cross-covariance* between functions, but in that case we can standardise the function and just use *cross-correlation* (valued between 0 and 1) instead.

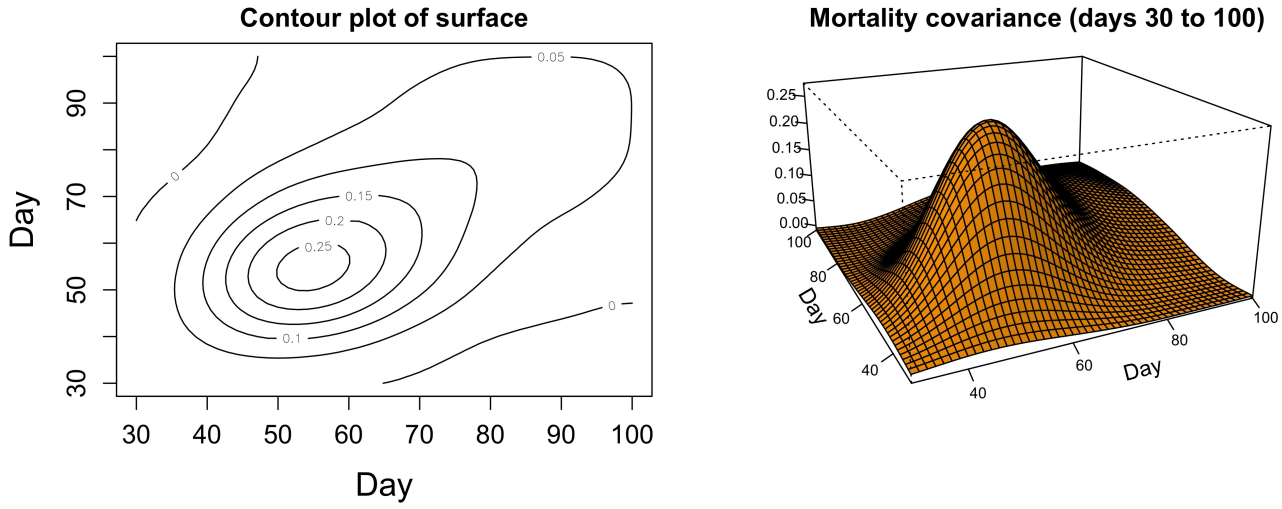

Figure 6.1: The covariance function of the mortality data, between days 30 and 100. Left panel: A contour plot representing the values of the covariance surface, which shows a maximum covariance around (55,55)

I compute the covariance function of COVID-19 mortality, shown in Figure 6.1. I choose to only show the function between days 30 and 100, because outside of this interval, mortality is small/zero, so covariance isn't as interesting to investigate. Every covariance surface will have a positive "ridge" along the diagonal of equal argument values. This is because $\text{cov}(t, t)$ is equal to variance when the argument values are the same, which is a nonnegative value. In order to interpret a bivariate covariance function, we can look along this ridge and see how the covariance drops in the perpendicular directions to it. For example, consider the peak covariance at roughly the point (50,50), where the covariance drops quite steeply on either side and drops to zero by the point (30,70). From this, we can see that positive relationships between mortality values do not seem to exist past a difference in the time value of 40 days (around 6 weeks). To summarise, given an increasing mortality value, mortality up to 6 weeks after that will tend to also be increasing. The same argument applies to decreasing mortality.

Another thing to note is that this particular covariance function is zero or highly positive everywhere. This implies that, for the whole time period, there is either no relationship between mortality on different days or a strong positive association. Combining this with our 40-day limit for positive covariance that

we noticed previously, we can conclude the following: Over the 180 days representing the first wave of the pandemic, when deaths began to rise, they would tend to consistently rise for 6 weeks before reaching a peak (and continue to fall when they began to fall). Therefore, just from this covariance function, we can tell that the first wave of the COVID-19 pandemic lasted 12 weeks.

6.2 Functional band depth and box plots

Our journey in FDA has been partly defined by creating functional equivalents to multivariate statistical methods. The next step is to consider ordering of data, and to develop a way of *ordering* a set of functions. Having done this, we will be able to create a *functional box plot*, analogous to a univariate box plot.

To order curves, we will use the concept of statistical *depth*, which measures how "central" an observation is with respect to the overall spread of the data set. For functional data, this is known as a function's *band depth*, a measure developed by López-Pintado and Romo (2009). We first must define a concept known as the **band** determined by a set of functions.

Definition 6.2.1 (Function band)**.** Given a set of real-valued functions, $x_1(t), \ldots, x_n(t)$ over an interval $\mathcal{I}$, the **band**, $B(x_1, \ldots, x_n)$, delimited by these curves is a subset of $\mathbb{R}^2$ defined as:

$$B(x_1, \ldots, x_n) = \{(t, y) : t \in \mathcal{I}, \min_{k=1,\ldots,n} x_k(t) \leq y \leq \max_{k=1,\ldots,n} x_k(t)\} \tag{6.3}$$

A band is essentially a shaded area on a plane that contains all the curves $x_1, \ldots, x_n$, with the boundaries of this area being the maximum/minimum values of all the functions considered pointwise.

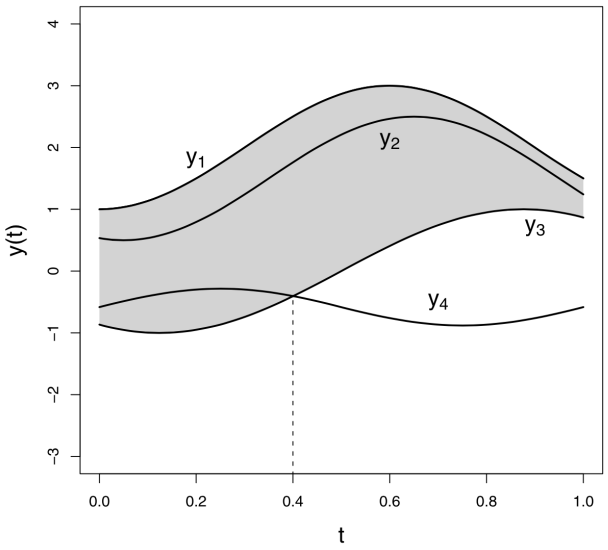

Figure 6.2: An example of functional bands.(Sun and Genton 2011, p. 319). The grey shaded area is the band, $B(y_1, y_3)$, bounded by $y_1(t)$ and $y_{1,3}(t)$. $y_2(t)$ is contained entirely in the band, whereas $y_4(t)$ is only partially contained in the band.

An example of a band defined by two functions is shown in Figure 6.2. Now we know what a band is, we can define the concept of the *band depth* of a function:

Definition 6.2.2 (k-Band depth)**.** Given a set of n functions, $x_1, \ldots, x_n$, define the k-Band depth of one of these functions x as:

$$BD_n^{(k)}(x) = \frac{\sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq n} \left[\mathbb{1}\big(\mathrm{G}(x) \subseteq B(x_{i_1}, \ldots, x_{i_k})\big) \right]}{\binom{n}{j}} \tag{6.4}$$

Where the sum is over all possible choices of k band defining functions from the n total functions. $\mathrm{G}(x)$ is the *graph* of x, i.e. the 2-D set of its argument and output values, and $\mathbb{1}$ is the indicator function for whether x is fully contained in each of the bands we sum over (which are each determined by each choice of k functions). Diving this sum by the binomial choice factor means that $BD_n^{(k)}(x)$ represents the proportion of all possible bands that contain the function x. The overall *band depth* of a function then simply sums these proportions over k, to cover all possible bands defined by differing numbers of functions.

Definition 6.2.3 (Band depth)**.** Summing (5.11) over the numbers of delimiting curves k, from 2 up to a chosen value K, gives us the band depth of a function x:

$$BD_{n,K}(x) = \sum_{k=2}^{K} BD_n^{(k)}(x) \tag{6.5}$$

Where we must sum from $k = 2$ because a band requires a minimum of two functions to be defined.

The simple interpretation of a band depth value is that a higher band depth implies the function is contained in more bands, and is hence more "central" with respect to the other functions. López-Pintado and Romo (2009) also define a modified band depth that takes into account how long a function is inside a band, instead of needing to be fully contained. This lowers the chance of two functions having the same depth value, and its definition isn't too different from the standard band depth. Finally, we define the function with the largest band depth as the *median* function.

Having calculated band depths for all of our functions, we can now order the functions from the largest band depth to the smallest. To create a functional box plot, we define the middle 50% region (analogous to the interquartile range in a univariate box plot) as the band defined by the top 50% of curves with the highest band depth (Sun and Genton 2011, p. 320). The outlier limits of the box plot are defined by extending the central 50% region's boundaries to include 1.5 times more band depth values. To conclude this chapter, I create a functional box plot for our 78 mortality curves, shown in Figure 6.3.

The pink region represents a band that contains functions of the top 50% of band depths in the 78 functions. We can see that it is at its widest around day 50, where mortality peaks, showing that the most variation in mortality is found around this peak. Also, the black solid line is the median function, for the county of Suffolk, with a maximum band depth of 0.488 (3dp). The dotted red lines are five curves identified as possible outliers. Four of the five only go outside of the limits right at the right-hand side of the x-axis, so I am not too worried about these functions, since smoothed functions always exhibit inflated variation at their tails as a result of the smoothing methods. The only curve I might choose to exclude in further analysis is the curve for the county of Hartlepool, which exceeds the outlier limit around day 75 in Figure 6.3.

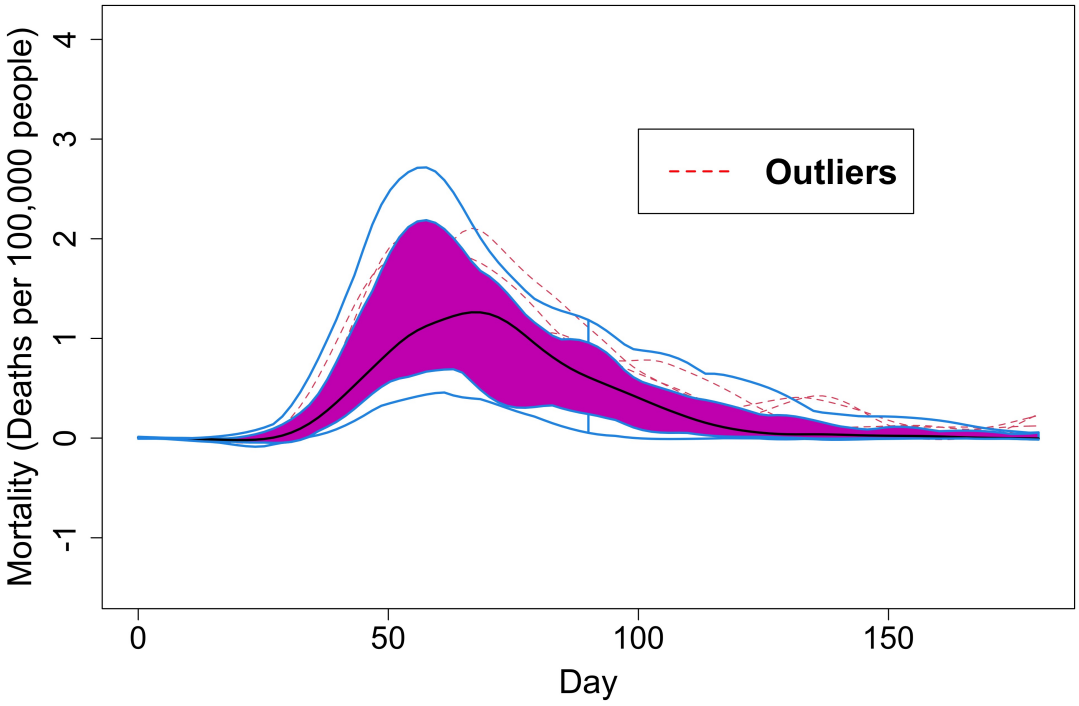

Figure 6.3: A functional box plot for the 78 COVID-19 mortality curves. The pink region represents the central 50% region containing all curves with the top 50% highest band depth. The outer blue lines represent the outlier limits, and the red dotted lines are functions that exceed these limits at some point. Therefore, they are considered as possible outliers.

Chapter 7

Linear models using functional variables

7.1 Overview of functional linear models

Up to this point, we have only considered FDA methods that analyse the variability of curves using only the curve itself. However, just as in standard multivariate statistics, the natural next step is to look at how other variables can be used to explain this variation. This is done using linear models, where the response and predictor variables are scalars. A *functional linear model* is an extension of this, where at least one of the response/predictors is a functional variable. In particular, there are three types of functional linear model: Function-on-Scalar, Scalar-on-Function and Function-on-Function. In these three terms, the word on the left represents the type of response variable, and the word on the right represents the predictor variables. The literature is extensive for each of the three types of linear model, so in this chapter I will focus solely on the case of Function-on-Function regression.

In the case of Function-on-Function regression, both the response and predictor variables are curves. These curves are defined at every point in a given interval, therefore we must decide how much information from the predictor curves is used to predict a given value of the response curve. The two most common solutions are in fact the two extremes: One point of the curve or the whole curve. The first is known as a *concurrent* functional model, and the second is a *total* functional model. In the same way that these models are expansions of multiple linear regression, there are also functional versions of regression summary statistics, such as R^2. Examples of this will be computed for the total functional model using the COVID-19 data.

The methods in this chapter are based on the assumption that we have only a few functional predictor variables. Qi and Luo (2018) discusses the situation where we have more predictor functions than replicates in our sample, such as in the case of fMRI scanning, where there are thousands of 3-D image functions, each for a different section of the brain. In this report, we are working with only two functional predictors: Mobility and Positivity, so we won't run into any dimension related problems since we have 78 curves in each case. I now introduce the concurrent functional model as a simpler example of Function-on-Function regression.

7.2 Concurrent functional regression

The concurrent functional linear model can be written as follows:

$$y_i(t) = \alpha(t) + \sum_{j=1}^{p} x_{ij}(t)\beta_j(t) + \epsilon(t) \tag{7.1}$$

Where $y_i(t)$ represents the response curve for the i^{th} replicate, $x_{ij}(t)$ is the j^{th} predictor function, $\beta_j(t)$ is a coefficient *function* and $\alpha(t)$ and $\epsilon(t)$ are intercept and error functions (assumed t. In order to predict the value of the response function at a time t, we only use the value of the predictor functions at that same time t.

An immediate simplification that we can make is setting the intercept function $\alpha(t) = 0$ (Harrison, Docherty, and Semple 2020). The justification for this is that we can assume the other three terms, $y_i(t), x_{ij}(t)$ and $\epsilon(t)$, in (7.1) all have a mean of zero. The zero mean of the error term is based on the typical standardised normal distribution assumption in regression. For the response and predictor functions, we can calculate their mean functions and subtract this from the functions themselves to standardise their mean, i.e. $\mathbb{E}[x(t) - \bar{x}(t)] = 0$. As a result, we know that the intercept function will have zero mean, and hence we can reasonably assume that it is just the zero function. The simplified form of model (7.1), written in matrix notation, is therefore:

$$\boldsymbol{y}(t) = \boldsymbol{X}(t)\boldsymbol{\beta}(t) + \boldsymbol{\epsilon}(t) \tag{7.2}$$

Where $\boldsymbol{X}(t)$ is an $n \times p$ matrix containing all p predictor curves for each of the n replicates. In the case of standard multiple regression, we choose the regression coefficient vector that minimises the least squares criterion. In the concurrent model case, we do something similar, however we must first smooth what is now a regression coefficient *function*. We use the roughness penalty smoothing of Section 4.2 by expanding $\beta_j(t)$ in terms of basis functions. Then our "functional least squares" criterion will instead calculate the coefficients of this basis expansion (which can be then used to compute the regression coefficient function). The roughness penalised concurrent criterion is stated below, however I omit the minimising solution here, and it can be found in Ramsay and Silverman (2005, p. 257):

Definition 7.2.1 (Roughness penalised concurrent model fitting criterion)**.** Given a concurrent functional linear model as in (7.2), the roughness penalised fitting criterion is given by:

$$RPCM(\boldsymbol{\beta}) = \int [\boldsymbol{y}(t) - \boldsymbol{X}(t)\boldsymbol{\beta}(t)]^T [\boldsymbol{y}(t) - \boldsymbol{X}(t)\boldsymbol{\beta}(t)] dt + \sum_1^p \lambda_j \int [\beta_j''(t)]^2 dt \tag{7.3}$$

Where the first integral is the functional equivalent of the residual sum of squares, and the second sum represents roughness penalties on each of the p regression functions.

The flexibility of having a separate roughness penalty for each regression function is important to note, and it allows us to control how much variation in each covariate we want to take into account in our model.

Although a concurrent linear model is computationally easier to fit, in my opinion a total functional linear model would be more appropriate for my data set. It is well known that deaths on a given day of a pandemic are affected by events days/weeks before the day of death. In particular, considering the mobility variable, someone may catch the disease because of a trip to the shops weeks before they end up dying. Therefore, if we were to only take the person's mobility on the day they die into account, it would be rather useless (they're probably not going grocery shopping on that day!). Therefore, a model that takes into account the predictor variables at all times would be ideal, and this is exactly what the *total* model does.

7.3 Total functional regression

The total functional model can be written as follows:

$$y_i(t) = \sum_{j=1}^{p} \int_I x_{ij}(s)\beta(s,t)ds + \epsilon(t) \tag{7.4}$$

Where we have set the intercept function to zero for the same reason as for the concurrent model. This model is essentially the same as the concurrent model, two main differences: The predictor variables are each integrated over an interval I (which for our data set would be the 180 days in the first half of 2020), and the coefficient function is now bivariate. When predicting the value of the response function at a time t, we can interpret the bivariate function $\beta(s,t)$ as the different weights given to the predictor variable at each time $s \in I$. Also, note that this coefficient function is now a *surface*, compared to a curve in the concurrent case. This makes it slightly harder to interpret, but it is definitely still possible.

The fitting criterion for the model (7.4) is similar to that used in the concurrent case (7.3). However, instead of imposing roughness penalties on the coefficient *surface* $\beta(s,t)$, we choose to smooth this surface simply by using fewer basis functions. This alternative to a roughness penalty was discussed back in Chapter 4, and we use it here because the equivalent calculations with the roughness penalty are too involved for this report. In particular, since we now must smooth a bivariate coefficient function, two sets of basis functions are required to describe $\beta(s,t)$ (Horváth and Kokoszka 2012, p. 130):

$$\beta(s,t) = \sum_{k=1}^{K}\sum_{l=1}^{L} c_{kl}\phi_k(s)\theta_l(t) = \boldsymbol{\phi}(s)\boldsymbol{C}\boldsymbol{\theta}(t) \tag{7.5}$$

Where $\boldsymbol{\phi}(s)$ and $\boldsymbol{\theta}(t)$ are vectors that contain the two sets of basis functions, and $\boldsymbol{C}$ is a $K \times L$ matrix containing the coefficients of the double basis expansion. In the case of this double basis expansion, lowering the number of basis functions has a different effect depending on if we reduce K (for $\boldsymbol{\phi}(s)$) or L (for $\boldsymbol{\theta}(t)$) (Ramsay and Silverman 2005, p. 282).

Truncating the $\boldsymbol{\phi}(s)$ basis ensures that the model is not over fit. A high number of basis functions, K, in this case, would mean that we are giving weight to very fine levels of detail from our predictor function, in order to predict the response function. However, we must remember that the predictor function itself has been smoothed, and hence we must lower the number of functions to reflect this. On the other hand, truncating the $\boldsymbol{\theta}(s)$ basis ensures the predicted curves are smooth. The process of choosing a certain type of basis function does not change, and I will stick to B-splines for both bases when fitting a model later in this chapter.

Using the basis expansion, we can write our *total* model fitting criterion (in the case of one predictor curve for cleaner notation):

$$TM(\beta) = \int_{I_2} \sum_{i=1}^{N} \left(y_i(t) - \int_I x_i(s)\beta(s,t)ds \right)^2 dt \tag{7.6}$$

$$= \int_{I_2} \sum_{i=1}^{N} \left(y_i(t) - \int_I x_i(s)\boldsymbol{\phi}(s)\boldsymbol{C}\boldsymbol{\theta}(t)ds \right)^2 dt \tag{7.7}$$

The expression in the large brackets is again the functional equivalent of the residual sum of squares. The two integrals ensure that all information about the predictor and response curves are taken into

account, where I and I_2 are the intervals over which they are defined, respectively. Minimisation of $TM(\beta)$ is done in R by solving a set of normal equations (Ramsay and Silverman 2005, p. 292), however explicit details are again not given since they are outside the scope of this report. This will give us the matrix $\boldsymbol{C}$ of the basis coefficients for $\beta(s,t)$, which we use to fit the model (7.4).

7.4 Model fitting and inference

I will now apply these methods to the COVID-19 data set, and fit two *total* functional linear models. One of them will use **mobility** to grocery shops and pharmacies as a functional predictor, whilst the other will use COVID-19 PCR test **positivity** rate. Both models will attempt to predict COVID-19 mortality (deaths per 100,00 people) for each of the 78 English counties in the data set.

The first step is to create a basis expansion for the coefficient surface, $\beta(s,t)$. Focusing on the case of the mobility model, after some trial and error, I chose to use 15 B-spline basis functions for both $\boldsymbol{\phi}(s)$ and $\boldsymbol{\theta}(t)$. As can be seen on the right of Figure 7.1 and in Figure 7.2, this ends up with a smooth coefficient surface and predictions which are accurate but not overly optimistic of the model's ability to predict mortality.

Interpreting the coefficient surfaces alone is hard to do, but we can attempt to take some interesting information from it. The way we interpret them is by picking a time on the mortality (response) axis and looking at whether the surface is above or below zero for points along the positivity (predictor) axis. I have tried to make features of the surfaces as clear as possible, but choosing an optimal angle to view the surfaces from was difficult.

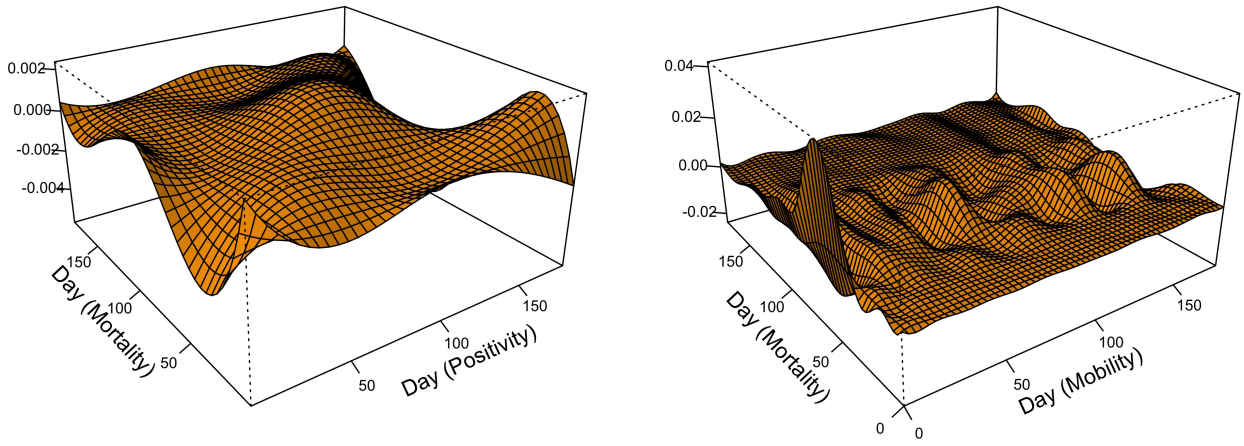

Figure 7.1: Two coefficient surfaces $\beta(s,t)$ for two separate total functional models, both predicting COVID-19 mortality. The t axis is labelled as "Day (Mortality) and the s axis represents the predictor of mortality/mobility. The left surface is from the model with mobility as a predictor, and the one on the right uses test positivity rate as a predictor. Both surfaces are interpreted by fixing a value on the mortality axis, then looking along the positivity axis to see whether the value of the surface is above or below zero. The sign of the surface value reflects the association between the two variables at that point. Positive regions represent a positive association (increase in predictor implies increase in response) between the two variables, and vice versa for regions below zero.

Firstly consider the surface on the left of Figure 7.1, for the positivity model. If you look along the diagonal, from $(0, 0)$ to $(150, 150)$, you can see a ridge which drops below zero on both sides of the diagonal. This implies that increased positivity at a point in time is a positive predictor of increased mortality for similar times around that time. Also, early on around day 10 we can see a positive weight given to positivity, when predicting mortality around day 70. This is when mortality peaked, so it makes sense that earlier positive tests would predict later mortality.

Second, consider the surface for the mobility model (right of Figure 7.1). An obvious feature to note here is the positive peaks early on, on the mobility axis. In particular, the peaks are stretch along day 50 to 90 on the mortality axis, implying early trips (before the lockdown around day 40) to grocery shops and pharmacies was a predictor of increased mortality after the lockdown began.

Another aspect of the models we can analyse is the "fitted functions" (like the fitted values of a standard linear model), i.e. the predictions that the model gives for the mortality curves in our dataset. Using these, we can assess the goodness-of-fit of our model. Firstly, we can visually examine the predicted curves to see how close they are to the actual mortality curves from the data. Two examples of this are shown in Figure 7.2 for the counties of Kent and County Durham. In both cases the predictions are quite accurate and definitely, at least, resemble the shape of the actual curve. This level of accuracy is reasonably consistent for most of the 78 counties in the data. However, I did notice the model seemed to produce underestimates for curves that had comparatively higher levels of mortality. This is most likely due to the fact that the data mostly contains curves with mortality peaking around 1 to 1.5, so naturally predictions above this would be brought down slightly.

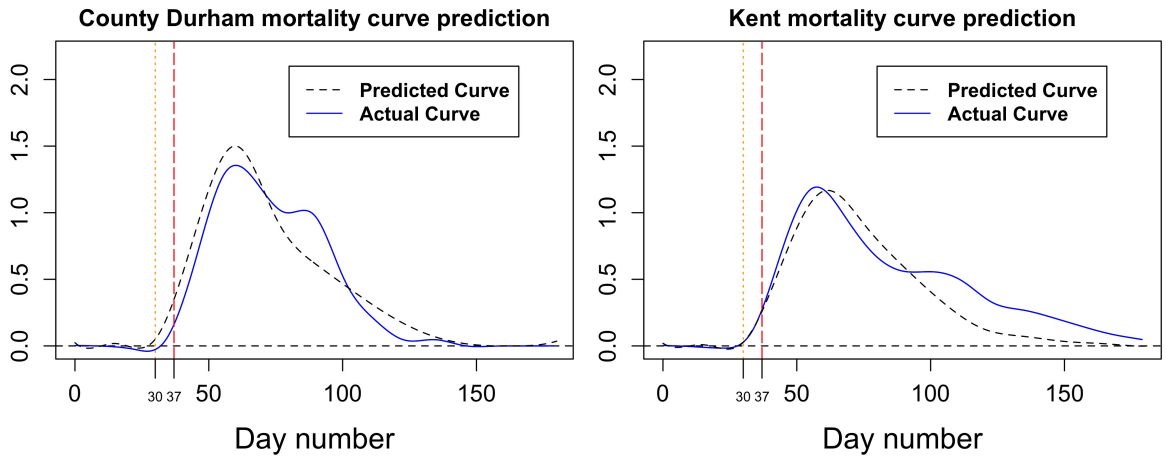

Figure 7.2: Using the fitted *total* functional model for mortality, the dashed curves represent the model's prediction of the mortality curve given the mobility curve for two counties: County Durham and Kent. The blue curve represents what the actual curve based on the mortality data.

A second, quantifiable goodness-of-fit measure is a functional equivalent of the R^2 value. In the case of functional data analysis R^2 is no longer a single value but, a function. We expand the standard definition to a function, using the standard R^2 expression pointwise for all values of the function's argument, t (Ramsay and Silverman 2005, p. 285):

Definition 7.4.1 (Functional R^2)**.** Given a total functional linear model, as in (7.4), we can produce N predicted response curves, $\hat{y}_i(t)$ from each of the N predictor curves. Using this, we define the

Functional R^2 function of the model as follows.

$$R^2(t) = \frac{\sum_{i=1}^{N}[y_i(t) - \bar{y}(t)]^2 - \sum_{i=1}^{N}[\hat{y}_i(t) - y_i(t)]^2}{\sum_{i=1}^{N}[y_i(t) - \bar{y}(t)]^2} \tag{7.8}$$

Where $y_i(t)$ is a smoothed function from the data, and $\bar{y}(t)$ is the mean function of those N functions.

The R^2 functions for both models are shown in Figure 7.3. In order to compute them, we evaluate the expression (7.8) at a fine mesh of t values, and then interpolate the resulting values into a function. As in standard linear models, each value of these R^2 functions represents the proportion of variance in the data we have captured in our model. The mobility model has an average R^2 around 0.5, and the positivity model has an average around 0.6. Both of these values reflect a reasonably good model, showing that both variables are effective in explaining variance in mortality. In both cases, however, the function seems to go wild at the two extremes of the argument value, even going negative around zero. A negative value of R^2 represents the original mean function being a better fit to the true data than the predicted curve. We can deduce this from formula (7.8), where the only way the whole expression can be negative is the second numerator is larger than the first. The first term is the sum of squared distances of the true function to the mean function, whereas the second term is the same but of the true function to the model fitted function. Therefore, a negative numerator means the distances to the mean function are smaller overall, and hence is what we would call a "better model".

In the case of our mortality models, $R^2(t)$ is negative up to around day 30. This makes sense since, at these points, we are essentially trying to fit a model to estimate a zero function (mortality is zero up to around day 30). There obviously isn't much point to this, and we might as well just use the mean of the functions.

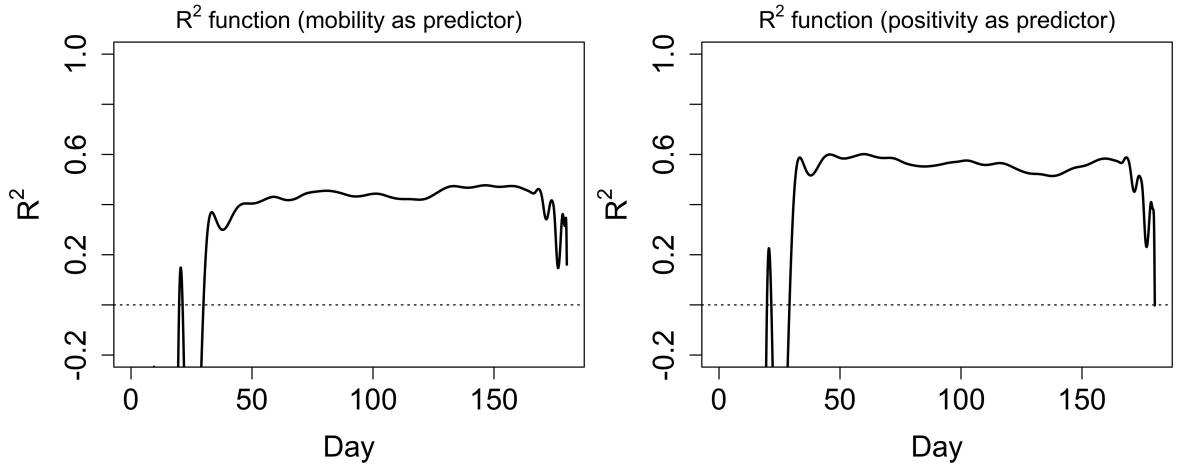

Figure 7.3: The R^2 function for the two total functional models, the left one with mobility as a functional predictor variable and the right-hand one with test positivity. For both functions, R^2 averages around 0.5 to 0.6, showing that the models capture around 50-60% of variation in mortality. Negative values of the function represent the mean function being a better fit to the data than the model, at that point.

Chapter 8

Conclusion

8.1 Conclusion

Over the course of this report, I have shown how COVID-19 data can be considered as an example of functional data. Using FDA methods, I was able to characterise the different smaller scale epidemics that occurred in each English county, over the first wave of the COVID-19 epidemic. Using roughness penalty smoothing with a B-spline basis, I produced 78 curves for 3 different functional variables of mortality, mobility, and positivity. Visually, these curves reconfirm the trends in deaths, travel, and cases that the UK saw reported by the government during the first wave of the pandemic. However, quantitatively, we were able to use these functions to perform descriptive and inferential data analyses.

Mean functions combined with curve registration gave an indication of the average COVID-19 trends in the country, whilst covariance functions quantified this trend by comparing the strength and direction of the relationship within a variable at any two times. For example, in the case of COVID-19 mortality, an aligned mean curve showed that daily COVID-19 deaths peaked 3 weeks after the lockdown began, at roughly 1.3 deaths per 100,000 people in each county. A mean function for the derivative of mortality showed that deaths were, on average, increasing faster about a week after the first lockdown began on March 23rd, 2020. Allowing for lags in death registration and the time from symptom onset to death, I concluded that the lockdown perhaps came a week or so too late.

A covariance function for mortality showed the relationship between deaths at any two points in time. The positive ridge of the covariance function (Figure 6.1) showed that variance in deaths on a given day, between counties, was at its peak around the same time the number of deaths peaked. Around this peak, deaths were still very variable county to county, shown by the high positive covariances. It must be noted that although the covariance values were only decimals below 0.5, mortality is defined on a scale resulting in values around 1. Therefore, a covariance of 0.25 is in fact "high" in this case. Overall, this showed that some counties did experience worse epidemics compared to others.

Another descriptive FDA method that I used was functional ordering, in terms of band depth values and functional box plots. Each of the curves has a band depth assigned to it, which is calculated relative to the other functions of the same variable. It essentially tells us how similar a curve is to all the other curves, and hence giving us a measure of "centrality" for a curve. A curve with a very low band depth might be considered an outlier, whereas the curve with the highest band depth is identified as the *median* function. In terms of COVID-19 mortality, band depth calculations concluded that the county of Suffolk experienced the most "typical" epidemic in the first half of 2020, with mortality peaking at about 1.2 and decreasing to near zero roughly 2 months later. The box plot acts analogously to box plots in the

univariate sense, with a central "50%" region with outer limits representing limits where a function is considered an outlier if it goes beyond them. The box plot for mortality identified one possible outlier as the unitary authority of Hartlepool. However, it only violated the limits for outliers over a short time span overall, therefore I decided that it did not have too negative of an effect on our results. In future analysis, I would consider removing this curve in advance.

Introducing a new type of variable in functional variables, in addition to standard scalar variables, allowed us to fit more complex linear models in Chapter 7. In particular, we can use combinations of scalar and functional variables for the predictors/response in a model. I focused on fitting fully functional models, in the form of *concurrent* and *total* models where both the predictors and the response are functions. The *total* model was the one that I used to make inferences, since it takes into account information from the predictors over the entire time period to predict a single value of the predictor. On the other hand, the concurrent model only uses pointwise values between the curves to make predictions. In particular, I decided that mobility and test positivity would be natural options to use as predictors of mortality. Fitting a total functional model produces a plot known as the coefficient surface, which we can use to quantify the direction of influence of the predictor variable on the response. In the case of positivity, a clear feature was that positive tests at the beginning of the pandemic were predictors of future increased deaths. The other model used mobility as a functional predictor variable, where mobility quantifies the number of trips and length of stay at grocery shops and pharmacies with respect to a pre-pandemic baseline. The coefficient surface in this case implied that early in the pandemic, trips to grocery shops and pharmacies were a predictor of increased mortality later on. Perhaps as a result of panic buying, shown in the mobility curves as highly increased function values before and as the lockdown began.

The coefficient surfaces are difficult to interpret on their own, therefore I considered the fitted functions of each model and calculated R^2 functions. The fitted curves are the set of mortality curves that the model would predict for each county given the input data, and in most cases produced predictions which were similar to the actual observed data. This implies that the models are quite effective for predicting mortality, and therefore could be used with data from other countries and future pandemics. To quantify the effectiveness of each model in capturing mortality variation, I computed an R^2 *function* for each model. This is a functional analogue to the R^2 *value* in standard statistics. Ignoring erroneous values at the tails of the function, the mobility R^2 averaged around 0.5, and the positivity R^2 around 0.6. These are reasonable values, and hence we can be confident that these models are a good initial step in capturing mortality variation.

Alongside this heavy application based look at FDA, I also looked at theoretical considerations which must be made when creating a functional data set. This included the choice of basis functions to represent our data, where I decided that the B-spline basis was the best choice due to its flexibility and efficient computation. In addition to this, approaches to curve fitting were considered, with a comparison of least squares smoothing and roughness penalty smoothing. Due to the smooth nature of our functions, we required a highly flexible and precise level of smoothing the data, which least squares cannot provide us with. As a result, I decided that adding a roughness penalty was the method to use, where a "best" value of its continuous smoothing parameter, λ is chosen using generalized cross validation. For each of the three functional variables, we applied different "amounts" of smoothing: $\lambda = \{1, 5, 800\}$ for mobility, positivity, and mortality respectively. A final essential step in functional data processing was the curve registration, which separated phase and amplitude variation in the data by aligning the curves. Two methods were proposed to do this, in landmark and continuous registration. It was concluded that a

combination of the two methods produces the best results, where we first aligned curves based on key features and then aligned the resulting whole curve using a global criterion. Having used these methods, any functional data set would then be ready for analysis.

8.2 Possible further research

FDA is a modern and developing area of research, with its theory based in well established subjects like functional analysis and stochastic processes. Although FDA is focused on analysing data, it has the scope to branch into areas such as probability theory by considering the distributions of functions on function spaces.

Furthermore, I have only discussed a few of the many FDA methods available. Further advances have been made in functional regression recently, with methods such as robust functional regression (Hullait et al. 2021) which deals with outliers better, or flexible functional regression (Ivanescu et al. 2015) which easily fits models with sparse data, multiple functional predictors, and more. These methods would be even more effective for dealing with real life data, and hence our functional data analysis of COVID-19 data would benefit from these methods.

A key aspect of all the FDA in this report, and in most literature, is that the functions have a single argument t. However, with the likes of medical imaging, a multidimensional argument would be required. I could possibly expand the COVID-19 FDA to include multidimensional arguments by considering the spatial aspect of the data, to further pinpoint regional differences in the effects of the pandemic.

Another possible different route to take with FDA is to consider the functional version of PCA (fPCA) in more detail. Alongside functional regression, fPCA is the other statistical method with the most current literature behind it and is an effective way of analysing the different "modes" of variation within a function. I decided that the descriptive methods alongside functional regression were the most efficient and effective way of analysing the data from a functional perspective. However, I am confident that fPCA would reveal further interesting features of the data if I were to use it in the future.

Finally, I only considered functional predictors in the regression section of this report. However, it is not too difficult to include scalar covariates, with a slight tweak to the theory. The models created in this report captured a high, but not perfect level of variation in mortality, and including various scalar variables such as regional access to healthcare, percentage of population over 65, etc. would improve models and allow for more detailed inference.

Bibliography

Tang, Chen, Tiandong Wang, and Panpan Zhang (2020). "Functional data analysis: An application to COVID-19 data in the United States". In: *arXiv preprint arXiv:2009.08363*.

Boschi, Tobia et al. (Aug. 2021). "Functional data analysis characterizes the shapes of the first COVID-19 epidemic wave in Italy". In: *Scientific Reports* 11, p. 17054. DOI: 10.1038/s41598-021-95866-y.

Kokoszka, Piotr and Matthew Reimherr (2017). *Introduction to functional data analysis*. Chapman and Hall/CRC.

Ullah, Shahid and Caroline F Finch (2013). "Applications of functional data analysis: A systematic review". In: *BMC medical research methodology* 13.1, pp. 1–12.

Tuddenham, Read D (1954). "Physical growth of California boys and girls from birth to eighteen years". In: *University of California publications in child development* 1, pp. 183–364.

UK Health Security Agency (2022a). *Cumulative COVID-19 deaths*. URL: https://coronavirus.data.gov.uk/metrics/doc/cumDeaths28DaysByDeathDate (visited on 03/02/2022).

Zhang, Xiaoke and Jane-Ling Wang (2016). "From sparse to dense functional data and beyond". In: *The Annals of Statistics* 44.5, pp. 2281–2321.

Aneiros, Germán and Philippe Vieu (2014). "Variable selection in infinite-dimensional problems". In: *Statistics & Probability Letters* 94, pp. 12–20.

Ramsay and Silverman (2005). *Functional Data Analysis*. Springer.

Wikipedia (2022). *Metropolitan and non-metropolitan counties of England — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Metropolitan%20and%20non-metropolitan%20counties%20of%20England&oldid=1083946204. [Online; accessed 19-March-2022].

Cornish, Danielle and Nadia Lohawala (2021). *User guide to mortality statistics*. English. ONS. 47 pp.

ONS (June 2020). *Estimates of the population for the UK, England and Wales, Scotland and Northern Ireland*. shorturl.at/blARU.

Google LLC (Mar. 2022). *Google COVID-19 Community Mobility Reports*. https://www.google.com/covid19/mobility/. (Visited on 03/07/2022).

UK Health Security Agency (2022b). *PCR test positivity percentage by specimen date*. URL: https://coronavirus.data.gov.uk/metrics/doc/uniqueCasePositivityBySpecimenDateRollingSum (visited on 03/02/2022).

De Boor, Carl (2001). "A practical guide to splines 2001". In: *Appl. Math. Sci.*

Hastie, T. J. and R. J. Tibshirani (1990). *Generalized additive models*. Chapman & Hall.

Curry, H. B. and I. J. Schoenberg (1966). "On Pólya frequency functions IV: The fundamental spline functions and their limits". In: *Journal d'Analyse Mathématique* 17.1, pp. 71–107. DOI: 10.1007/bf02788653.

Ramsay, James, Giles Hooker, and Spencer Graves (2009). *Functional Data Analysis with R and MATLAB*. Springer New York.

Green, Peter J and Bernard W Silverman (1993). *Nonparametric regression and generalized linear models: a roughness penalty approach*. Crc Press.

Che, Xiang Jiu et al. (2011). "The product of two B-spline functions". In: *Advanced Materials Research*. Vol. 186. Trans Tech Publ, pp. 445–448.

Craven, Peter and Grace Wahba (1978). "Smoothing noisy data with spline functions". In: *Numerische mathematik* 31.4, pp. 377–403.

Moore, Andrew W and Mary S Lee (1994). "Efficient algorithms for minimizing cross validation error". In: *Machine Learning Proceedings 1994*. Elsevier, pp. 190–198.

Gu, Chong (2013). *Smoothing spline ANOVA models*. Vol. 297. Springer.

Hutchinson, Michael F and Frank R de Hoog (1985). "Smoothing noisy data with spline functions". In: *Numerische Mathematik* 47.1, pp. 99–106.

Marron, James Stephen et al. (2015). "Functional data analysis of amplitude and phase variation". In: *Statistical Science*, pp. 468–484.

Ramsay, James O and Xiaochun Li (1998). "Curve registration". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 60.2, pp. 351–363.

Modersitzki, Jan (2003). *Numerical methods for image registration*. OUP Oxford.

Kneip, A and J Ramsay (2008). "Combining registration and fitting for functional models". In: *Journal of the American Statistical Association* 103.483, pp. 1155–1165.

Gasser, Theo and Alois Kneip (1995). "Searching for structure in curve samples". In: *Journal of the american statistical association* 90.432, pp. 1179–1188.

Harrison, EM, A Docherty, and C Semple (2020). *COVID-19: time from symptom onset until death in UK hospitalised patients*. https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/928729/S0803_CO-CIN_-_Time_from_symptom_onset_until_death.pdf.

Horváth, Lajos and Piotr Kokoszka (2012). *Inference for functional data with applications*. Vol. 200. Springer Science & Business Media.

López-Pintado, Sara and Juan Romo (2009). "On the concept of depth for functional data". In: *Journal of the American statistical Association* 104.486, pp. 718–734.

Sun, Ying and Marc G Genton (2011). "Functional boxplots". In: *Journal of Computational and Graphical Statistics* 20.2, pp. 316–334.

Qi, Xin and Ruiyan Luo (2018). "Function-on-function regression with thousands of predictive curves". In: *Journal of Multivariate Analysis* 163, pp. 51–66.

Hullait, Harjit et al. (2021). "Robust function-on-function regression". In: *Technometrics* 63.3, pp. 396–409.

Ivanescu, Andrada E et al. (2015). "Penalized function-on-function regression". In: *Computational Statistics* 30.2, pp. 539–568.