Dissertation First Draft

190005680

2022-10-10

Contents

1 Introduction

Throughout this dissertation time blah blah

2 What is Functional Data?

To answer this question let's first think about what type of data a statistician is typically tasked with analysing and how this data would be given to the statistician. Let us take an example of a survey where patients are given a drug thought to reduce blood pressure. In this example, the statistician is given blood pressure readings for each patient on the first 4 days after the patient has taken the drug. These readings would typically be stored in a some form of data frame. This data frame could then be converted to a data matrix such that each column of the matrix represented a new patient and each row represented a new day on which readings were taken. An example of such a data matrix (with fabricated data) is given in (1) below.

$$\begin{pmatrix} 142 & 155 & 136 \\ 131 & 134 & 122 \\ 122 & 120 & 110 \\ 95 & 111 & 99 \end{pmatrix} \tag{1}$$

In this data matrix the first entry given in the top left represents patient 1's blood pressure reading (in mmHg) on day 1 after taking the drug, the entry directly below represents patients 1's blood pressure reading on day 2 and so on.

Now consider a scenario where the number of times a patient can have their blood pressure read within the 4 days of the survey was increased to twice a day. If a patients blood pressure was checked twice a day then each row of the data matrix would then represent a reading taken every half day which would make the dimensions of the new data matrix 3×8. A statistician would be in favour of this increase in information as having access to more regular blood pressure readings would allow the statistician to strengthen their inferences about the effects the drug has on a patients blood pressure. However, the statistician would be quick to point out that this increase would lead to their analysis taking longer as more computational power is required to make inferences from this larger data set. In today's modern era of technology it is becoming increasingly easier for machines to take measurements of various quantities at smaller and smaller increments of time. This leads to the production of massive data sets which can be difficult for even the most complex of computers to analyse. The field of functional data analysis seeks to address this issue.

Returning to the previous example, now consider a scenario where a persons blood pressure could be taken at any time the statistician requested such that any time within the survey period of 4 days was accessible. This would not be practically achievable as the statistician may ask to have the persons blood pressure read multiple times within the same minute for example. However, if the relationship between time and a patients blood pressure was able to be given by a function with which the statistician could input a point in time within the 4 days and the patients blood pressure would be returned then the statistician could do this. If such a function existed for every patient then the statistician may prefer to be able to conduct his analysis on these functions, not on individual blood pressure readings found using this function. The statistician could now consider their sample to no longer consist of blood pressure readings at various times for each patient. It would instead consist of 3 functions, one for each patient, which give the relationship each that patients blood pressure has with the time since the drug was taken. Assuming a function could be found for each patient, that would mean that if the observed values given in the data matrix shown in (1) were free from error they would simply be draws from this set of functions for each patient. This new sample, consisting of functions, as opposed to singular observations at various times for each patient, is a sample of functional data.

This change can be alternatively conceptualised as expanding the data matrix in (1) to be infinitely dimensional e.g. every point point in time (of which there are infinite) has an associated row in the data matrix.

This conceptualisation can sometimes be easier to understand as a function is a 1-1 mapping of points in the domain of the function (the input values) to values in the range of the function (the values output by the function). However, it clearly makes little practical sense as storing infinitely dimensional data matrices is impossible. Thinking of our samples this way does show some of the advantages that functional data analysis holds over other, more conventional, forms of data analysis.

3 Advantages of Functional Data

The consideration of the functional data as effectively being an infinitely dimensional data matrix shows a clear advantage of working with functional data over non-functional. This advantage is that it gives statisticians access to more data about a quantity of interest than most forms typical non-functional data analysis allow. A statistician, theoretically, has access to all information about how a quantity changes over time which allows for higher accuracy in predictions and in identifying trends in data.

Another advantage of using functional data is that it is a potential remedy to the so-called 'Curse of Dimensionality' (Riccio (2022)). This refers to the issue that as the number of dimensions of a dataset increases (be that the number of variables collected for each unit within a study or the number of times that variable is collected within the course of the study) the computational power needed to analyse the dataset grows exponentially. Functional data analysis allows a statistician to not work with large matrices of data but instead with functions for which there are standard mathematical approaches which are more computationally simplistic. High-dimensional data can also lead to over fitting of models to the data which reduces the predictive power of models. Functional data overcomes this issue by simplifying relationships between variables earlier on in the analysis which combats over fitting in the early stages of an analysis.

Working with functions instead of single observations of data also allows a statistician to look at the behaviour of the derivatives of these functions. This can reveal some interesting variation within and between curves that difficult to assess when working with non-functional data. This advantage will not be discussed at length within this dissertation. The reader can look to (REFERENCE FOR FUNCTIONAL DERIVATIVES)

There is a clear issue with this discussion being that functional data is impossible to have access to in a practical setting. A statistician can only have access to a finite amount of readings of quantities and the relationship between a quantity and time is typically not known otherwise statistical techniques would not have to be used to assess these relationships. In practical applications of functional data analysis the functional data must be estimated from the information that a statistician has access to, being measurements of a certain quantity for several time points. This can reduce the accuracy of the information a statistician has access to, however, careful consideration of the potential assumptions made in the process of estimating the functional data from the observed data can reduce this inaccuracy. The Creating a Functional Variable section will detail how functional data can be estimated non-functional observations of data.

4 Creating a Functional Variable

4.1 How is Observed Data Related To Functional Data?

As discussed previously, if a statistician wants to work with functional data they must estimate these functions from their observed data (These being collections of observations of a quantity over time for several replications). Therefore, there must be a relationship between the functional data and the observed data.

The observations that a statistician typically has access to consist of observations of a quantity over time for several relaications and the functional data that can be derived from these observations are functions which a point in time can be input and the value of the quantity at that time for that replication is returned.A natural assumption would, therefore be that the observed value of a quantity at a specific time for a specific replication is the value that the functional data for that replication takes at the time that the observation was taken. This would imply that each observation is completely free of error as the functional data describe the relationship between the quantity of interest and time without any error. This is not a practical assumption as there is often random noise or limitations in the devices used to measure quantities which introduces errors in the observed values.

Therefore, it is assumed that there is error in the observations. Within this dissertation, it will be assumed that the error in each observation are independent and identically distributed normal random variables with constant variance. This relationship is given in Equation (2).

$$Y_i = f(t_i) + \epsilon_i \tag{2}$$

where Y_i is the value of the variable of interest at time t_i and $\epsilon_i \sim N(0, \sigma^2)$ (σ is a constant).

There are techniques which allow for the relaxation of some of the assumptions made about the error in each observation. ____________ details a method which allows the assumption of independence in errors between observations to relax and _________ details a method which allows a relaxation of normality in the errors.

In order to estimate a functional sample from observed data a process called smoothing must be used. To understand how to perform this process the concept of basis functions must be introduced.

4.2 Basis Functions

In order to give the reader an understanding of how to create a functional data object (one function which describes the relationship between a variable and time), both mathematically and in computationally in R, a new example will be used, which will be referred to throughout this dissertation as Example 1. In Example 1, it is assumed that the relationship between a variable and time is already known. This variable Y is related to time by the functional relationship given in Equation (3). This relationship is relatively simple and is expressible through typical mathematical notation, e.g. it is analytic. The reader should note that this will not typically be the case for relationships between variables and time. In fact, it is generally not possible to attain a simply expressible function for this relationship.

$$Y(t) = f_1(t) = \frac{1}{2}(t + t^2 + e^{-\frac{1}{5}t} + e^{-\frac{1}{30}t+8})) \tag{3}$$

Throughout this example, the smoothing process will be conducted as if the analytical form of this function is not known. To demonstrate how the smoothing process works, data is simulated from the function which mimics the data a statistician may receive in a typical analysis. The function is then be estimated from this data using smoothing. It is assumed that the assumptions about the relationship between the functional data and observed data established in Equation (2) are true for this data set when this data is generated. The variance of the error terms, (ϵ_i), is made to be 30 as this would simulate an observation error similar to that what would seen in a practical setting. The time points of the simulated data are evenly spaced in the region of $t \in [0, 40]$. The R Code used to simulate this data is given below.

```r
set.seed(190005680)

function_ex_1 <- function(Time){
  (1/2)*(Time+Time^2+exp(-1/5*(Time))+exp(-(1/30)*Time+8))
}

sigma_2_errors <- 30

Time_Grid <- seq(0,40,length.out=40)

Observations <- function_ex_1(Time_Grid)
```

```r
Number_Of_Observations <- length(Observations)

Observation_Errors <- rnorm(Number_Of_Observations,0,sigma_2_errors)

ex_1_data <- data.frame(
  Time = Time_Grid,
  Y=Observations+Observation_Errors
)
```

The simulated data is shown graphically in Figure 1 below.

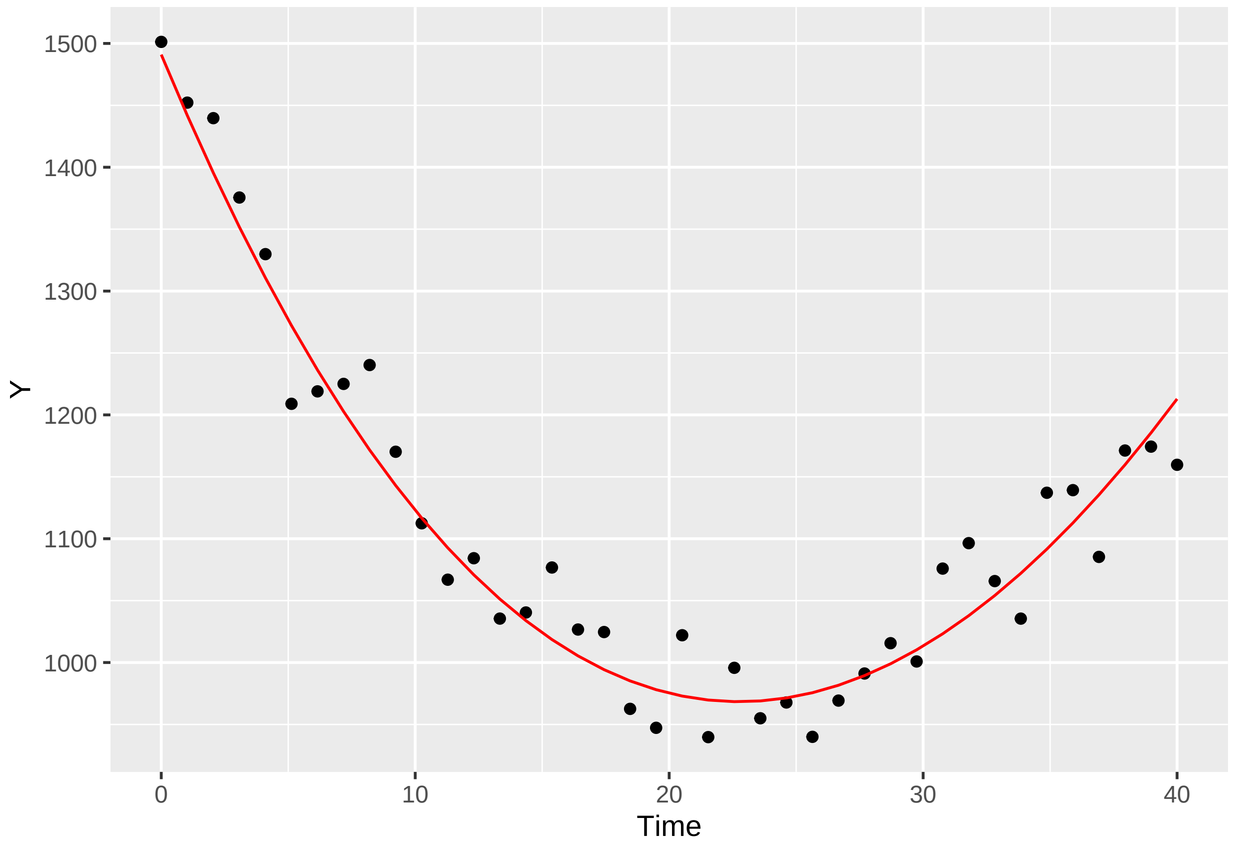


Figure 1: A Plot of Y over Time For Our First Example Dataset. The red line gives the actual relationship Y(t)

In order to estimate the original function, $f_1(t)$, from this new dataset some mathematical building blocks are needed which we can combine in some way to construct this function. When working with vectors, basis vectors, sets of vectors that can be linearly combined to construct any vectors within a defined vector space, can be thought of as the building blocks for all vectors. Similarly, basis functions can be thought of as the building blocks of all functions.

Basis functions are sets of functions which can be linearly combined to produce any function within the function space they belong to. Basis functions are very powerful tools for function estimation and are often used to estimate functions from sets of points in space. This dissertation will work with two sets of basis functions, the Fourier Basis and the B-Spline Basis which are both sets of functions, the elements of which can be linearly combined to construct functions with a domain in $[t_0, t_1]$ where t_0 and t_1 are arbitrary.

The function f_1 can be expressed as a linear combination of a set of basis functions for the functional space it is contained in. This basis function expansion of f_1 is shown in Equation (4).

$$Y(t) = f_1(t) \approx \sum_{k=1}^{K} c_k \phi_k(t) = \boldsymbol{c'}\boldsymbol{\phi}(t) \quad t \in [t_0, t_1] \tag{4}$$

where c_k is the coefficient of the k_{th} basis function $\phi_k(t)$ and $\{\phi_k(t) : k \in \mathbb{Z}^+\}$ is a set of basis functions whose span is a set of functions with domain in $[t_0, t_1]$. This expansion can be used for any function that a statistician may attempt to estimate.

$\boldsymbol{\phi}(t)$ is a vector of the basis functions used to construct $f_1(t)$ evaluated at time t and $\boldsymbol{c}$ is a vector of coefficients for each of these basis functions.

The reader may notice the approximation sign in Equation (4). For this to be an equals sign K would have to be equal to the number of basis functions which define the function space that $f_1(x)$ belongs to. This is not always achievable as sometimes there is an infinite number of these basis functions in a functional basis. A computer simply could not store all of the coefficients for these basis functions. Therefore, the statistician must select how many basis functions they wish to use based on how well they wish to approximate the function.

It was mentioned that this dissertation will use the Fourier Basis and the B-Spline basis. These will be defined in the The Fourier Basis and The B-Spline Basis sections respectively.

4.2.1 The Fourier Basis

The Fourier basis is an orthonormal set of basis functions which can approximate any periodic function. These basis functions are defined by their period T. The period of a function is defined in Equation (5). The set defined by T can be linearly combined to give any T-periodic function.

$$\phi(t+T) = \phi(t) \tag{5}$$

The form of the functions within the Fourier basis system defined by period T is given below:

$$\begin{aligned} \phi_1(t) &= 1 \\ \phi_2(t) &= sin(\omega t) \\ \phi_3(t) &= cos(\omega t) \\ &.... \\ \phi_{2n}(t) &= sin(n\omega t) \\ \phi_{2n+1}(t) &= cos(n\omega t) \end{aligned}$$

where $n \in \mathbb{Z}$ and $t \in \mathbb{R}$.

The ω in these functions is defined in Equation (6).

$$\omega = 2\pi/T \tag{6}$$

To create a Fourier basis system in R the create.fourier.basis command in the fda R package is used. An example is given below:

```
create.fourier.basis(c(45,65),65,5)
```

This command will create the a set which contains the first 65 fourier basis functions with period 5 as defined above. The fda package does not allow for these functions to be defined over all $t \in \mathbb{R}$ as this is not what is typically needed in the context of estimating functions from observed data. Therefore, this function makes the user input a domain that they wish these functions to be defined over. In this case the first argument passed to this function gives this range. For the functions created by this command $t \in [45, 65]$.

The second argument passed to this function gives the desired number of basis functions. Fourier basis functions must always come in sin cos pairs e.g if $\phi_2(t) = sin(\omega t)$ is included in a set then $\phi_3(t) = cos(\omega t)$ must also be included. The function create.fourier.basis accommodates for this, so that if an even number of basis function (K) is requested (there should always be an odd number) the function will also include that basis functions cosine pair (making the number of output basis functions $K + 1$).

The final parameter passed to this function gives the period of the basis system (T). In this case the period of the functions T is 5.

Some examples of periodic data are air temperature over the course of a few days, as the temperature may follow similar trends each day (so period $T = 1\ day$) or the angle a lever is in relation to being upright over time (The fourier basis is often used to estimate functions from data related to angles).

The Fourier Basis can be used to construct functions which are periodic. The B-Spline basis system is used to estimate functions from data which is not periodic in nature.

4.2.2 The B-Spline Basis

A B-Spline basis system is a sets of piece-wise polynomial functions, defined by the range that they are defined over, which will be referred to as $[a, b]$ where $a, b \in \mathbb{R}$, their knots and break points and their order (which will be referred to as n). A set of B-Spline functions which define a B-Spline basis can be linearly combined to give any spline function of the same degree (which is $n - 1$). A spline function is a piece-wise polynomial function which is defined over a constrained domain $[a, b]$ that takes values within the real numbers. The most important aspect of spline functions for estimating functions from observed data is that they are functions defined within $[a, b]$, that are differentiable $n - 1$ times. For example, a spline of order 2 , e.g. $n = 2$, is only differentiable once and, therefore, spline functions of order 2 will be functions that are made up of piecewise straight line segments. Therefore, a set of order 2 B-Spline functions which form a basis, defined on $[a, b]$, can be linearly combined to give any function, defined on $[a, b]$, which is made up of piecewise straight line segments. For a detailed explanation of how B-Splines are defined see the Appendix Section of this dissertation. These sets of B-Spline functions which form bases, similarly to the sets of Fourier functions which form bases, can be used to estimate functions from our observed data using the basis function decomposition given in Equation (4).

It is not necessary to know how B-Spline functions are defined to make use of them. The only knowledge necessary are their key features. It was mentioned above that the sets of B-Spline functions which make up a basis system are defined by their knots and break points. The break points of a B-Spline basis are the points where each of the piecewise polynomial sections of the spline (created by linear combination of the B-Splines) meet. The knots are related to the break points such that there is at least one knot at each break point. Each knot placed at a break point signifies how many derivative terms should appear to have a smooth join at a break point. If there is one knot placed at a break point then the first $n - 2$ derivatives should appear smooth across the break point. This is a specific example of the general relation that if there are r knots at a breakpoint then the first $n - r - 1$ derivatives of the spline function must appear smooth across the break point.

It is also important to understand the relation between the number of knots and the number of functions in a B-Spline basis. This relation is given in Equation (7).

$$number\ of\ basis\ functions\ =\ order\ +\ number\ of\ interior\ knots \tag{7}$$

The interior knots of a B-Spline function refers to the knots that do not lie on the boundaries of the domain in which the basis functions are defined e.g. are not at a or b.

This understanding of knots and break points is very useful if a statistician plans on using the derivatives of functions they are estimating from their non-functional sample. The statistician can define the set of B-Spline functions which form the basis with a high enough order and an appropriate placement of knots such that the derivatives that the statistician wishes to use of the function estimated are reasonably smooth. It is also useful to understand how the number of basis functions is related to the number of knots as the number of basis functions inadvertently defines the dimensionality of our new samples.

4.2.3 The Dimensionality of Samples of Functional Data

One of the main advantages of functional data analysis is the potential for the reduction of dimensionality within the data used for analysis. Returning to the original blood pressure example, the original dimensionality of the data, given in the data matrix in (1), was that there were 4 observations of blood pressure for 3 different replication, those being each patient, overall giving 12 observations. One would assume that when the sample moves from being the observed non-functional data to the sample being 4 functions that the dimensionality of the sample would be the four functions, one function for each patient. However, this is not the best way of thinking of the dimensionality of the sample. It is better to think of each function within the sample as being defined by the coefficients of the basis functions used to estimate the functions. Typically each function within the sample will be estimated using the same basis system as the data will be similar across all replications. Consider a situation in which a B-Spline basis composed of 2 functions is used to estimate the functions which give the blood pressure of each patient at a given time. The dimensions of the sample becomes 2 coefficients for each function with a function for each of the 3 patients which gives 3 pairs of coefficients (6 coefficients overall) which define our sample. This is how working with functional data reduces the dimensionality of samples and this reduction of dimensionality allows computers to conduct analysis faster than when working with non-functional data. This main advantage of working with functional data is important to remember when deciding how many basis functions we wish to use to estimate our functions. Using more basis functions than can lead to an increase in the dimensionality of our data but using too few could lead to a bad estimation of our function. How do we choose how many basis functions to use?

4.2.4 How Many Basis Functions?

It may seem intuitive to use as many basis functions as is computationally possible to estimate functions from non-functional data. Using more basis functions would allow for our basis function expansion, given in Equation (4), to approximate the shape of the actual function better. While this is generally the case it is not practical or necessary. Often adding more basis functions does not give the statistician any more information about the actual nature of the processes which produce the data. It can also lead to the problem of over fitting as in a practical application the function we do not know the true complexity of the function we are to estimate. The process of smoothing which will be introduced in the Finding Coefficients section will address this issue of over fitting and it is also part of the reason that adding more basis functions often does not produce a greater fit to the actual functions we are trying to estimate. Often it is a case of trial and error to find the number of basis functions which give the desired complexity for the data you are working with. (READ MORE INTO THIS)

4.2.5 Making a Set of B-Spline Basis Functions in R

To render B-spline basis functions in R for use in functions within the `fda` package the `create.bspline.basis` function from the `fda` package must be used. An example call to the function is given below

```
create.bspline.basis(c(0,4),order=2,breaks=c(0,1,3,4))
```

The first argument passed to this function gives the range that the B-Spline basis should be defined over, the second argument defines the order of splines and the final argument gives the values that break points should be placed at. Rendering basis functions using break points places one knot at each break point.

The following code gives another example of rendering basis functions. This time the set of basis functions is specified by giving the number of basis functions in the set and the order of the basis functions.

```
create.bspline.basis(c(0,10),16,7)
```

This code will render 16 B-spline functions of order 7 over the range of $t \in [1, 10]$. From the relation given in Equation (7), it can be seen that there should be $16 - 7 = 9$ interior knots within the basis system and so the functions to be defined by 9 knots that are equally spaced that cover the interval of $t \in [1, 10]$.

Returning to Example 1, the data from this example is non-periodic, e.g. it does not have any repeating patterns, so B-splines can be to estimate the function.

A question that arises in relation to this example is how to choose the order of B-splines that are needed. The answer to this questions depends on what is needed to be done with the estimated function. Ifthe derivatives of this function are needed then the general rule is to *fix the order of the B-spline basis to be at least two higher than the highest order derivative that is needed.* If the derivatives of the function are not going to be analysed, the general rule is that a linear combination of B-splines of order 4 will provide a smooth enough estimate of the function. Therefore, B-Splines of order 4 will be used for this example.

In Example 1, there are 40 data points. Placing a break point at each observation means that there will be 42 basis functions in the basis used to estimate the function. This set of basis functions is rendered in R using the code given below and are stored in the variable `basis_ex_1`.

The set of basis functions that will be used to estimate the function, $f_1(t)$, in this example has now been defined. these basis functions will be linearly combined to estimate the function as shown in Equation (4). However, this cannot be yet be done as the coefficients of these basis functions in the linear combination still need to be estimated. Two methods of finding these coefficients are introduced in the Finding Coefficients section.

4.3 Finding Coefficients

Throughout the previous sections the process of smoothing has been repeatedly mentioned. Smoothing a data set refers to the process of approximating a function from that data set that gives the general trend of the variable of interest over a certain parameter, taking account of errors in measurement. In this case, the parameter a variable is changing over is typically time, however other parameters are sometimes used, e.g. angles. Smoothing is similar to interpolation however interpolation assumes that there is no error or noise in the observations and so the function estimated passes through every point in the data whereas smoothing assumes there are errors in the observations.

It was stated in Equation (2) that it will be assumed that each observed value within the non-functional data has some error associated with it so smoothing techniques should be used to estimate our curves, not interpolation. This section will introduce two smoothing techniques which attempt to estimate the coefficients of our basis functions that give the best estimate of our function $f_1(t)$. These techniques will also be used more generally throughout the dissertation to estimate functions when there is no access to the function which describes the relationship between a variable and time. All methods discussed in the Finding Coefficients section can be applied for any choice of basis function, even choices not mentioned within this dissertation.

4.3.1 Regression Smoothing

The first method of finding the coefficients of our basis functions to be considered is smoothing using regression analysis.

This is the simplest method of smoothing. This method seeks to make the difference between the estimated function and the observed values (known as the residuals) as small as possible. It does this by rearranging Equation (2) as shown below and then minimising these residuals.

$$Y_i = f(t_i) + \epsilon_i$$
$$\Rightarrow \epsilon_i = Y_i - f(t_i)$$

It minimises these residuals by minimising the sum of squared errors as defined in Equation (9).

$$SSE(f) = \sum_{j=1}^{n} [\epsilon_i]^2 \tag{8}$$
$$\Rightarrow SSE(f) = \sum_{j=1}^{n} [Y_j - f(t_j)]^2 \tag{9}$$

where the $Y_j \; for \; j = 1, .., n$ are the n observed observations of the variable of interest and $f(t_j)$ is the function we are estimating, evaluated at time t_j (the time that Y_j is observed at).

This SSE can be rewritten in terms of basis functions and coefficients by substituting in the basis function decomposition of the function $f(t)$ given in Equation (4). This substitution is shown below.

$$SSE(f) = \sum_{j=1}^{n} [Y_j - f(t_j)]^2$$
$$\Rightarrow SSE(\boldsymbol{c}) = \sum_{j=1}^{n} [Y_j - \sum_{k=1}^{K} c_k \phi_k(t_j)]^2$$
$$\Rightarrow SSE(\boldsymbol{c}) = \sum_{j=1}^{n} [Y_j - \boldsymbol{\phi(t_j)'c}]^2$$

The reader may notice that the SSE changes from being a function of the function f to being a function of the coefficient vector ,$\boldsymbol{c}$, within this substitution. This is because the basis functions used to construct f are constant and have been pre-chosen. The only non-constant term within the SSE is the coefficient vector, $\boldsymbol{c}$

The vector $\boldsymbol{\phi(t_j)}$ gives the value of each of the basis functions evaluated at the observed time points t_j. All of the values of the basis functions evaluated at the time points that each observation was observed at are stored in a matrix $\boldsymbol{\phi}$ (often called the basis matrix). This matrix is structured such that each column of the matrix contains the basis functions evaluated at 1 observation of t, t_j. For example the first row of this matrix contains the values that the first basis function takes for every observed time point. This matrix is an $n \times K$ matrix.

In Example 1 where $f_1(t)$ of known analytical form is being estimated from simulated data, $\boldsymbol{\phi}$ can be found using the eval.basis command from the `fda` package. The first argument passed to this function is the time points which the user wishes to have their basis evaluated at and the second argument supplied is the basis functions, created using the `create.fourier.basis` or `create.bspline.basis` within this dissertation. The call to this function which gives the $\boldsymbol{\phi}$ matrix for this example is given below.

```r
phi_mat_ex_1 <- eval.basis(ex_1_data$Time,basis_ex_1)
```

Minimising the SSE function may seem quite difficult. This is a multivariate optimisation problem where there are many covariates to optimise. Taking our example as a case study, the number of coefficients to be estimated is 81 as there are 81 basis functions and the coefficients of these can be any number on the real number line.

This optimisation problem is not as difficult as it may appear. In typical non-functional regression analysis the sum of squared errors of the residuals is minimised to get regression coefficients. The SSE, as a function of the coefficients of the regression parameters, for non-functional regression analysis is given in Equation (10).

$$SSE(\boldsymbol{c}) = \sum_{j=1}^{n} [Y_j - \boldsymbol{X'c}]^2 \tag{10}$$

This is very similiar to the SSE for minimising the residuals of the function estimated from the observed data. The only difference is that in this SSE the $\boldsymbol{\phi}$ matrix takes the place of the design matrix, $\boldsymbol{X}$. The estimate of $\boldsymbol{c}$ that minimizes the SSE for non-functional regression analysis is given below in Equation (11).

$$\hat{\boldsymbol{c}} = (\boldsymbol{X'X})^{-1}\boldsymbol{X'Y} \tag{11}$$

When minimizing the SSE for finding the functional coefficients the design matrix, $\boldsymbol{X}$, can be exchanged with the basis matrix, $\boldsymbol{\phi}$, to get the estimate of the coefficient vector, $\hat{\boldsymbol{c}}$, given in Equation (12). This vector of coefficients is the vector of coefficients then used to estimate the function.

$$\hat{\boldsymbol{c}} = (\boldsymbol{\phi'\phi})^{-1}\boldsymbol{\phi'Y} \tag{12}$$

The `fda` package can conduct the process of minimising the SSE to find the coefficients of the basis functions that is shown above. The function `smooth.basis` can be used to conduct this form of regression smoothing as well as smoothing using a roughness penalty which is introduced in the next section. This function to only finds the coefficients by minimising the SSE as laid out above but also fits the resulting functional data object (being the linear combination of basis functions and coefficients). In order to conduct regression smoothing the user must pass in 3 arguments to the function. The first is the time points which the observed data are observed at, the second is the observations of the data in data matrix for similiar to that shown in (1) for the blood pressure example and the final argument is the basis functions that the user wishes to use to estimate the function. The code below conducts the regression smoothing on the data simulated for Example 1 and fits the resulting functional data object.

```r
ex_1_smooth_basis <- smooth.basis(ex_1_data$Time,ex_1_data$Y,basis_ex_1)
```

This call to the `smooth.basis` function does not just return the functional data object. It also returns the value of the SSE that is given by the estimated coefficient vector as well as a few other values which will be discussed in the next section. the most important of the objects returned by the function is the functional data object itself which is stored as `fd` and can be accessed using the $ operator as shown, for Example 1, below.

```r
ex_1_fd_obj <- ex_1_smooth_basis$fd
```

This function can be easily plotted using the `plot` function found in base `R`. This is shown in the call to the function below. The points argument included overlays the original data from which the function was estimated and the lines argument overlays a red line which gives the true shape of the function which is to be estimated. The output of this call to the plot function is shown in Figure 2

```
plot(ex_1_fd_obj)
par(new=TRUE)
points(ex_1_data$Time,ex_1_data$Y)
lines(ex_1_data$Time,function_ex_1(ex_1_data$Time),col="red")
```

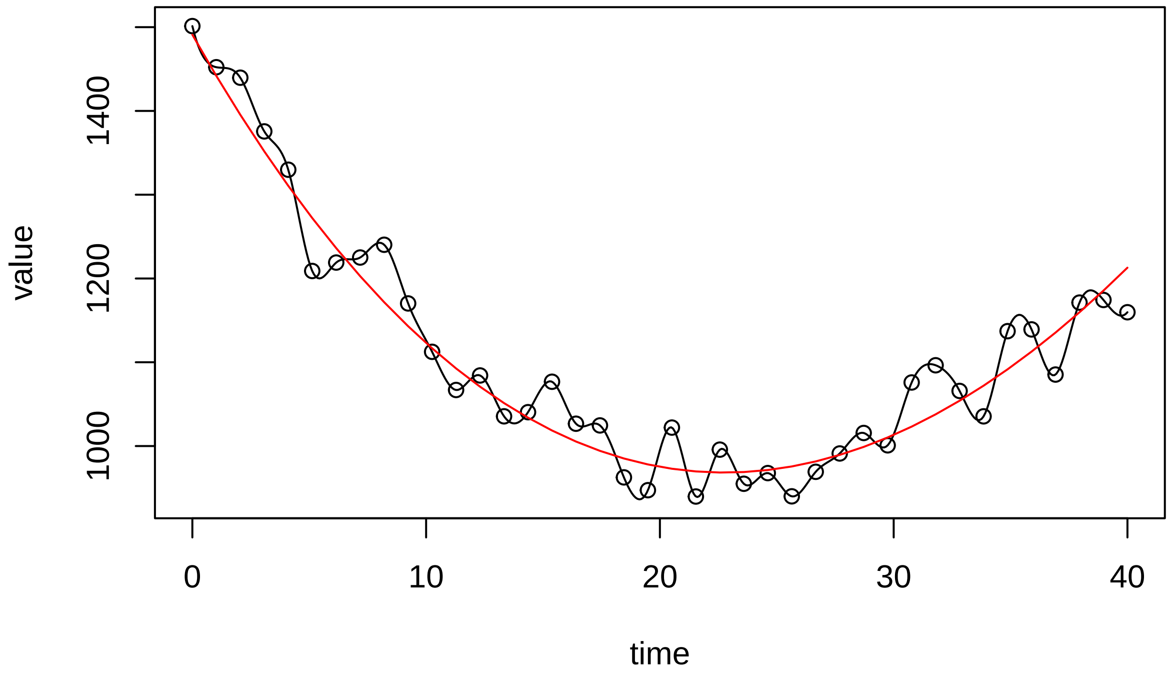


Figure 2: Estimate of Function With True Function (Red Line) and the Data Used to Estimate It Overlayed

It can be seen from Figure 2 that this method of finding the coefficients of a functional data object has a few drawbacks. This method assumes that there is no error in the observations and thus overfits the data as it only minimises the differences between the estimated function and the observed data (this is often refered to in literature as the function being too rough). This method also tends to give very unstable estimates of the derivatives of the curves near the boundaries of other data which is not useful as one of the advantages of functional data analysis is that it allows for analysis of the derivatives of the functional aspects of the data.

4.3.2 Smoothing using a Roughness Penalty

The generally preferred method of finding the coefficients for a functional data object is smoothing using a roughness penalty. This method is similar to that used in smoothing using regression analysis in that it uses a the sum of squared errors. However, this method adds a roughness penalty to this sum of squared errors in order to stop the functional data object from over fitting the observed data. The sum of squared errors with the new roughness penalty term, denoted by $F(\boldsymbol{c})$, is shown in Equation (13).

$$F(f) = \sum_j [Y_j - f(t_j)]^2 + \lambda \int_{t_0}^{t_1} [Lf(t)]^2 dt \tag{13}$$

where $f(t)$ is the estimated function evaluated at t, L is a linear differential operator (which will be discussed in the next section), λ is known as the smoothing parameter and $t \in [t_0, t_1]$.

This can be rewritten using the vector expansion $f(t) = \boldsymbol{c'\phi(t)}$ given in Equation (4) to get Equation (14).

$$F(\boldsymbol{c}) = \sum_j [Y_j - \boldsymbol{c'}\boldsymbol{\phi}(t_j)]^2 + \lambda \boldsymbol{c'}[\int_{t_0}^{t_1} L\boldsymbol{\phi}(t)L\boldsymbol{\phi}'(t)dt]\boldsymbol{c} \tag{14}$$

There are two unknowns in $F(\boldsymbol{c})$ those being the linear differential operator L and the smoothing parameter λ and these are chosen based on what features we wish our estimate of the function to have.

4.3.2.1 Choosing an Appropriate Linear Differential Operator Remembering that the most desirable set of coefficients, $\boldsymbol{c}$, will minimise $F(\boldsymbol{c})$, the linear differential operator penalises a function depending on how close to 0 a certain linear differential operator of the estimated function is to 0 across all time points. This is a way of eliminating a level of roughness in the estimated function. Typically it is useful to choose the linear differential operator, L, to be a simple derivative. The general rule is to use the derivative that is 2 higher than the derivative of the function that is to be used in analysis. Throughout this dissertation, no derivatives of the function will be used and so the linear differential operator is chosen to be the second derivative, D^2. This is because it is generally agreed that a straight line has no roughness, e.g. it only changes in one direction on the axis, and the second derivative of a straight line is 0. If the second derivative of a function is to be used it would be desirable for the second derivative of the function to be similarly smooth and this is why the derivative of two orders higher is used, e.g. the 4th derivative.

4.3.2.2 Choosing an Appropriate Smoothing Parameter λ The generalized cross-validation (GCV) criteria was developed by Craven P. and G. Wahba (Craven and Wahba (1978)) to assist in finding the most appropriate value of λ to use in smoothing using a roughness penalty. The need for this is clear as choosing a λ close to 0 means that $F(\boldsymbol{c})$ becomes the SSE used to find the coefficient for regression smoothing without a roughness penalty however if the λ chosen is too large the estimated function will typically approach a straight line.

The GCV for choosing the smoothing parameter λ is defined as shown in Equation (15) :

$$GCV(\lambda) = \left(\frac{n}{n - df(\lambda)}\right)\left(\frac{SSE}{n - df(\lambda)}\right) \tag{15}$$

where the SSE is defined as in the Regression Smoothing section, $df(\lambda)$ is the degrees of freedom for the function fitted to the data object using the λ specified and n is the number of observations in the non-functional data points used to estimate the function. The degrees of freedom of the fit of an observation of functional data are as defined in page 65 of Ramsay J. O. (2009).

The value of λ that gives the lowest GCV is the most preferable. The GCV produced by a certain value of λ can be found by accessing the GCV argument of the functional data object created in R.

To fit a function to non-functional data using regression smoothing using a roughness penalty in R the `smooth.basis` function is again used. To perform regression smoothing using a roughness penalty with this function similar arguments are passed to the `smooth.basis` function. The first two arguments are the same as regression smoothing without a roughness penalty, being the time points of the observations along with the observations. The third argument is different from regression smoothing. The third argument should be an `fdPar` object defined using the `fdPar` function in the `fda` package. The `fdPar` object can be thought of as supplying the `smooth.basis` function with the linear differential operator, basis functions and the value of lambda that should be used in the regression smoothing. The `fdPar` object can either take a linear differential operator defined through the method contained within the `fda` package or can take a single number. Supplying a single number to the `fdPar` function to define the linear differential operator defines the linear differential operator to be the derivative of the function corresponding to that number, e.g. if the number 2 is given then the second derivative of the function is used as the linear differential operator.

For Example 1, the derivatives of the function are not required so the standard linear differential operator, D^2, will be used in smoothing. An example of a possible function fit to the data in Example 1 is given below. The value of lambda used was chosen randomly.

```
lambda <- 10^-4

fd_par_obj_ex_1 <- fdPar(basis_ex_1,2,lambda)

ex_1_smooth_basis_rough_pen <- smooth.basis(ex_1_data$Time,ex_1_data$Y,fd_par_obj_ex_1)

ex_1_functional_observation <- ex_1_smooth_basis_rough_pen$fd
```

The first argument given to `fdPar` is the set of basis functions to be used, the second is the linear differential operator and the final is the chosen lambda.

This is not the optimal lambda that minimises the *GCV* function. To find this lambda many functions must be fit to the data using different values of lambda and the GCV's of these must be compared. This comparison can either be done graphically or computationally. To conduct this method computationally a function must be created that takes lambda as an input and outputs the GCV of the function fitted using that lambda. This function for this method of smoothing is given below.

```
GCV_func <- function(log_lambda,basis,observations,time_points,penalty){

  lambda <- 10^log_lambda

  fd_par_obj <- fdPar(basis,penalty,lambda)

  smoothbasisobj <- smooth.basis(time_points,observations,fd_par_obj)

  return(sum(smoothbasisobj$gcv))
}
```

It can be seen that this function fits a function to the observed data which is input using the linear differential operator, basis functions and lambda that are also input to the function and outputs the GCV of this function. This function takes the log of lambda as an input instead of lambda. This is because the behaviour of the GCV as a function of lambda can often have steep inclines making the optimisation challenging computationally. Taking the log of lambda flattens out these steep inclines so that the estimate of lambda used can be more accurate. This function can then be optimised using the `optimise` function found in base `R` and the minimising value of log lambda can be extracted. This process of finding the lambda which minimises the GCV is shown for Example 1 below.

```
optimised_function <- optimise(GCV_func,
                               lower=-4,upper=4,
                               basis=basis_ex_1,
                               observations=ex_1_data$Y,
                               time_points=ex_1_data$Time,
                               penalty=2)

minimum_log_lambda <- optimised_function$minimum

minimum_lambda <- 10^minimum_log_lambda
```

The lower and upper values of the call to this function specify the range of log lambda that the minimum should be found over. In this case it was 1 to 10. The quickest way to assess the range that is needed to be looked at is plotting some values of the GCV function graphically. An example of how to create a data frame which stores many values of the GCV along with their associated log lambda is given below.

```
loglambda <- seq(-4,4,0.25)

list_of_gcvs <- c()

for (log_lam in loglambda){

  lambda <- 10^log_lam

  fd_par_obj <- fdPar(basis_ex_1,2,lambda)

  smoothbasisobj <- smooth.basis(ex_1_data$Time,ex_1_data$Y,fd_par_obj)

  list_of_gcvs <- c(list_of_gcvs,sum(smoothbasisobj$gcv))

}

GCV_log_lambda_df <- data.frame(
                   loglambda=loglambda,
                   lambda=10^loglambda,
                   GCV=list_of_gcvs
                 )
```

A plot of this GCV function is given in Figure 3. It can be seen from this that the minimising value of lambda is in the region of log lambda in -4 to 4 so these are input as the lower and upper boundaries of the region to be searched to find the minimum of the GCV function.

Once the value of log lambda is found that minimises the GCV all components needed for smoothing are found and the process used to fit functions to the data shown above are used to get the optimal fit to the function for this method of smoothing. This is shown for Example 1 in the code below.

```
lambda <- minimum_lambda

ex_1_fd_par_obj <- fdPar(basis_ex_1,2,lambda)

ex_1_functional_object <- smooth.basis(ex_1_data$Time,ex_1_data$Y,ex_1_fd_par_obj)
```

`ex_1_functional_object` now contains the estimate of the function $f_1(t)$ found through regression smoothing using a roughness penalty. The purpose of estimating functions from observed data is to analyse them, so in a typical example the object `ex_1_functional_object` would now be the functional sample with which functional data analysis would be conducted on. Example 1, however, has been used to demonstrate the effectiveness of this smoothing technique in estimating functions from data when the functions analytical form is already known. Therefore, the fit of the estimated function should be assessed.

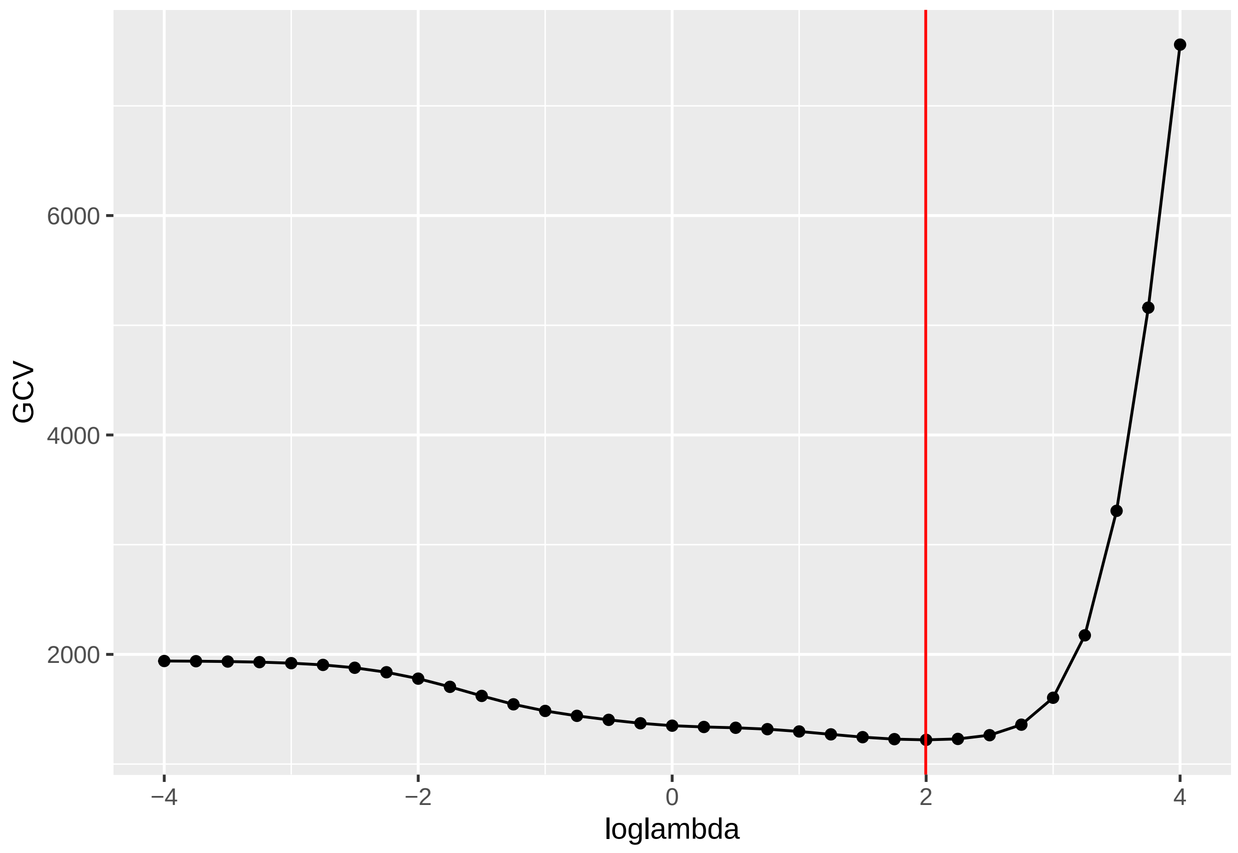
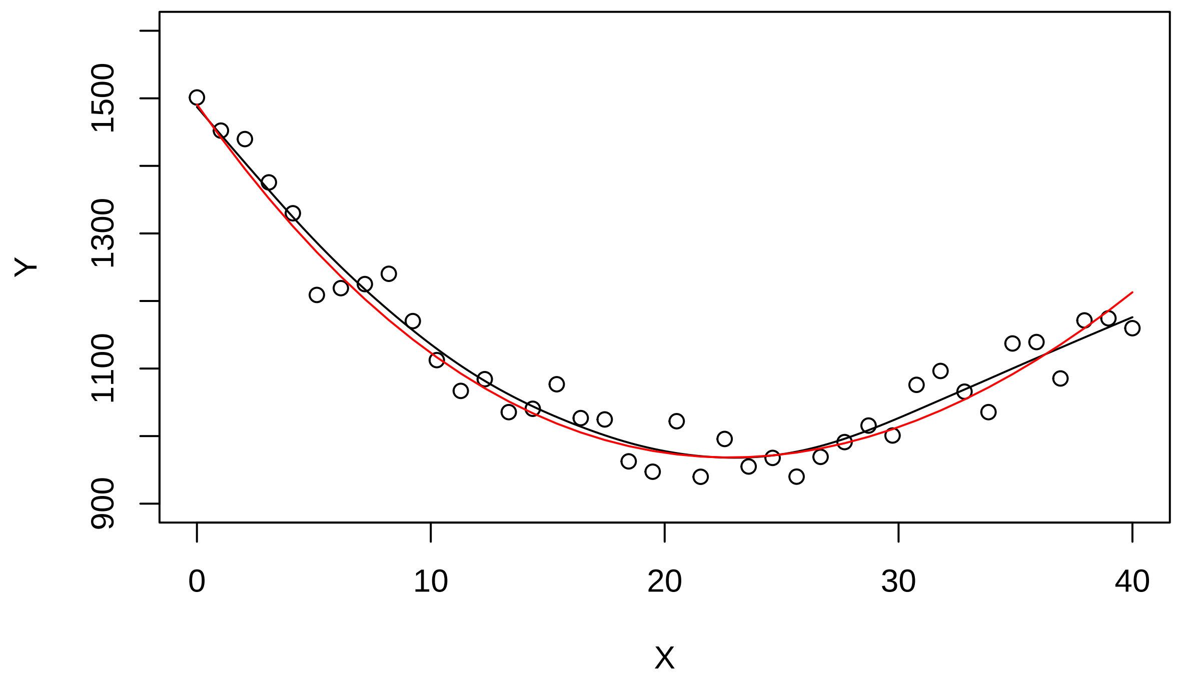


Figure 3: The GCV function for Example 1 with The Optimal Value of Lambda Found Computationally Indicated By The Red

5 Assessing the Fit of An Estimated Function



```
function_ex_1 <- ex_1_functional_object$fd

residuals <- ex_1_data$Y - as.vector(eval.fd(ex_1_data$Time,function_ex_1))

shapiro.test(residuals)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals
## W = 0.98139, p-value = 0.7411
```

6 The Open Air Dataset

7 Summary Statistics of Functional Data

8 Functional Principal Components Analysis (fPCA)

9 Functional Regression

10 Appendix

10.1 Derivation of A Second Order B-Spline Basis

To demonstrate the meaning of these terms we will look at the B-spline basis function. B-spline basis functions are defined by a recursion relation. This recursion relation states that for a B-Spline of order K+1, notated by $N_{i,k+1}(x)$ with *knot vector* $U = \{t_0, .., t_k\}$ is defined by :

$$N_{i,k+1}(x) = \omega_{i,k}(x)N_{i,k}(x) + (1 - \omega_{i+1,k}(x))N_{i+1,k}(x)$$

where:

$$N_{i,1}(x) = \begin{cases} 1 & if \;\; x \in [t_i, t_{i+1}) \\ 0 & elsewhere \end{cases}$$

and :

$$\omega_{i,k}(x) = \begin{cases} \frac{x-t_i}{t_{i+k}-t_i} & where \;\; t_{i+k} \neq t_i \; and \; t_{i+k} \in U \\ 0 & elsewhere \end{cases}$$

This recursion relation can be quite difficult to understand for someone unfamiliar with B-spline basis and therefore a worked through example of finding some B-spline functions of order 2 can be found in the appendix of this paper.

There are a few key features of B-splines that are notable. One of these features is that the first and last spline must equal 1 at the boundaries (the first and last knot value). This is because at any points in the domain established by the knots the sum of all the B-splines is equal to 1 such that $\sum_{i=1}^{K} N_{i,k} = 1$. This feature of B-splines means that when a linear combination of these is taken the coefficient of any B-spline function is approximately equal to the value of the B-spline at it's peak. Due to the first and last spline being equal to exactly one at the boundaries the coefficient of these splines is exactly equal to the value of these functions at their peak. Another feature is the relation between the number of knots, the order and number of basis functions. This relation is given by the following:

Say we wish to find a series of B-Spline basis functions of order two and our *knot vector* $U = \{0, 1, 3, 4\}$. Let us first define a B-spline basis of order 1 for the knot vector. This basis would be defined given by the functions:

$$N_{0,1}(x) = \begin{cases} 1 & if \;\; x \in [0, 1) \\ 0 & elsewhere \end{cases}$$

$$N_{1,1}(x) = \begin{cases} 1 & if \;\; x \in [1, 3) \\ 0 & elsewhere \end{cases}$$

$$N_{2,1}(x) = \begin{cases} 1 & if \ \ x \in [3,4) \\ 0 & elsewhere \end{cases}$$

These functions are shown graphically in Figure 4 below:

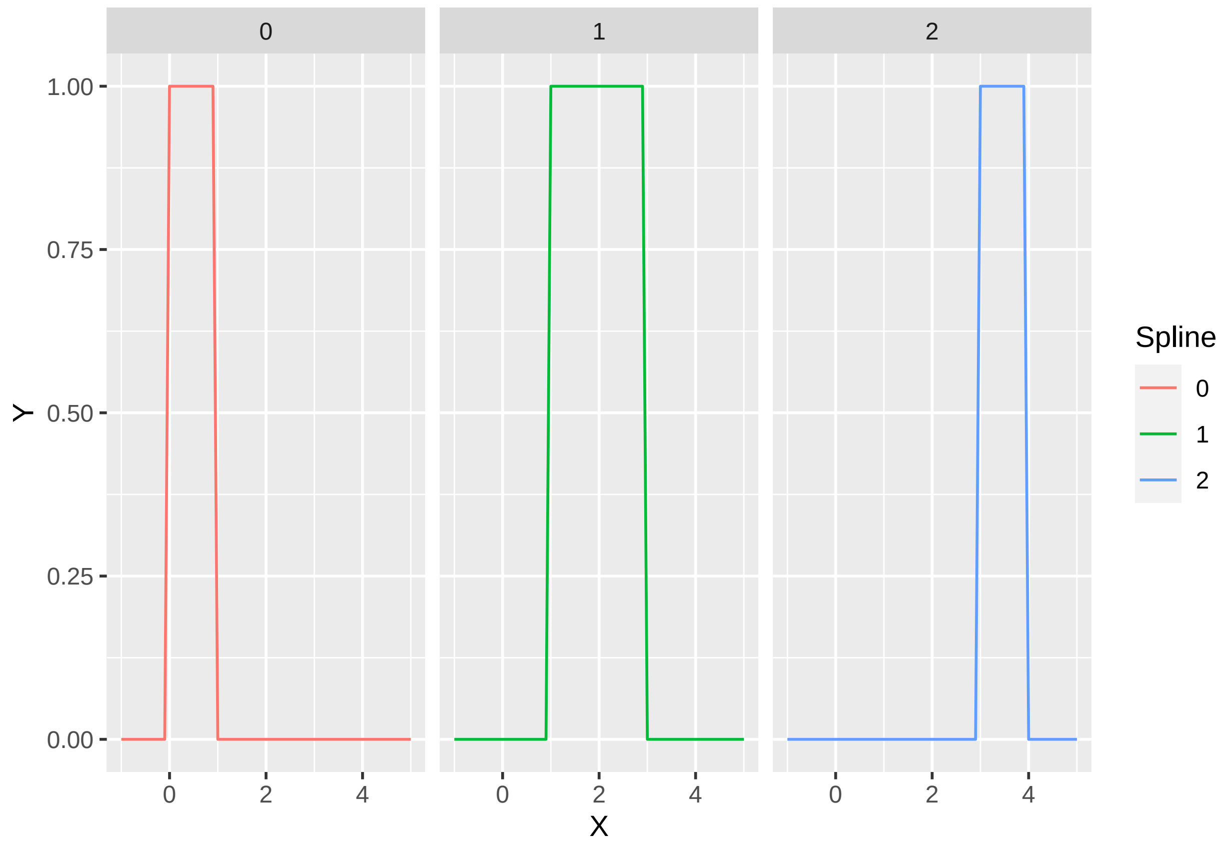


Figure 4: 1st Order B-Splines Represented Graphically

These order 1 functions can be used in the recurrence relation to find the splines of any desired order. In this case the derivation for only one of these order two basis functions will be shown but the same method can be followed to find the other two basis functions. We will find the second basis function $N_{1,2}$. First we must start by finding $\omega_{1,1}$. We know $i = 1$ and $k = 1$ (meaning $i + k = 2$) in this case so:

$$\omega_{1,2}(x) = \begin{cases} \frac{x-t_1}{t_2-t_1} & where \ \ t_2 \neq t_1 \ and \ t_2 \in U \\ 0 & elsewhere \end{cases}$$

We know $t_2 = 4 \neq 1 = t_1$ so there is only one case meaning:

$$\omega_{1,1}(x) = \frac{x-1}{3-1} = \frac{1}{2}x - \frac{1}{2}$$

Using the same method we find that:

$$\omega_{2,1}(x) = x - 3$$

Therefore, using the recursion relation:

$$\begin{aligned}N_{1,2}(x) &= \omega_{1,1}(x)N_{1,1}(x) + (1-\omega_{2,1}(x))N_{2,1}(x)\\ &= \left(\frac{1}{2}x - \frac{1}{2}\right)N_{1,1}(x) + (4-x)N_{2,1}(x)\end{aligned}$$

$$\Rightarrow N_{1,2}(x) = \begin{cases} \frac{1}{2}x - \frac{1}{2} & if \;\; x \in [1,3) \\ 4-x & if \;\; x \in [3,4) \\ 0 & elsewhere \end{cases}$$

In this case we will end up with 3 more basis functions of order 2 those being $N_{-1,2}(x)$, $N_{0,2}(x)$, $N_{1,2}(x)$ and $N_{2,2}(x)$ which are given by:

$$N_{-1,2}(x) = \begin{cases} 1-x & if \;\; x \in [0,1) \\ 0 & elsewhere \end{cases}$$

$$N_{0,2}(x) = \begin{cases} x & if \;\; x \in [0,1) \\ \frac{3}{2} - \frac{1}{2}x & if \;\; x \in [1,3) \\ 0 & elsewhere \end{cases}$$

$$N_{2,2}(x) = \begin{cases} x-3 & if \;\; x \in [0,1) \\ 0 & elsewhere \end{cases}$$

Graphical representation of these functions is given below in Figure 5:

References

Craven, Peter, and Grace Wahba. 1978. "Smoothing Noisy Data with Spline Functions." *Numerische Mathematik*. https://doi.org/10.1007/BF01404567.

Ramsay J. O., Graves S, Hooker G. 2009. *Functional Data Analysis with r and MATLAB*. 233 Spring Street, New York: Springer Science; Business Media.

Riccio, Donato. 2022. "Functional Data Analysis: A Solution to the Curse of Dimensionality." *Toward Data Science*. https://towardsdatascience.com/functional-data-analysis-a-solution-to-the-curse-of-dimensionality-f83dd19fa6e8.

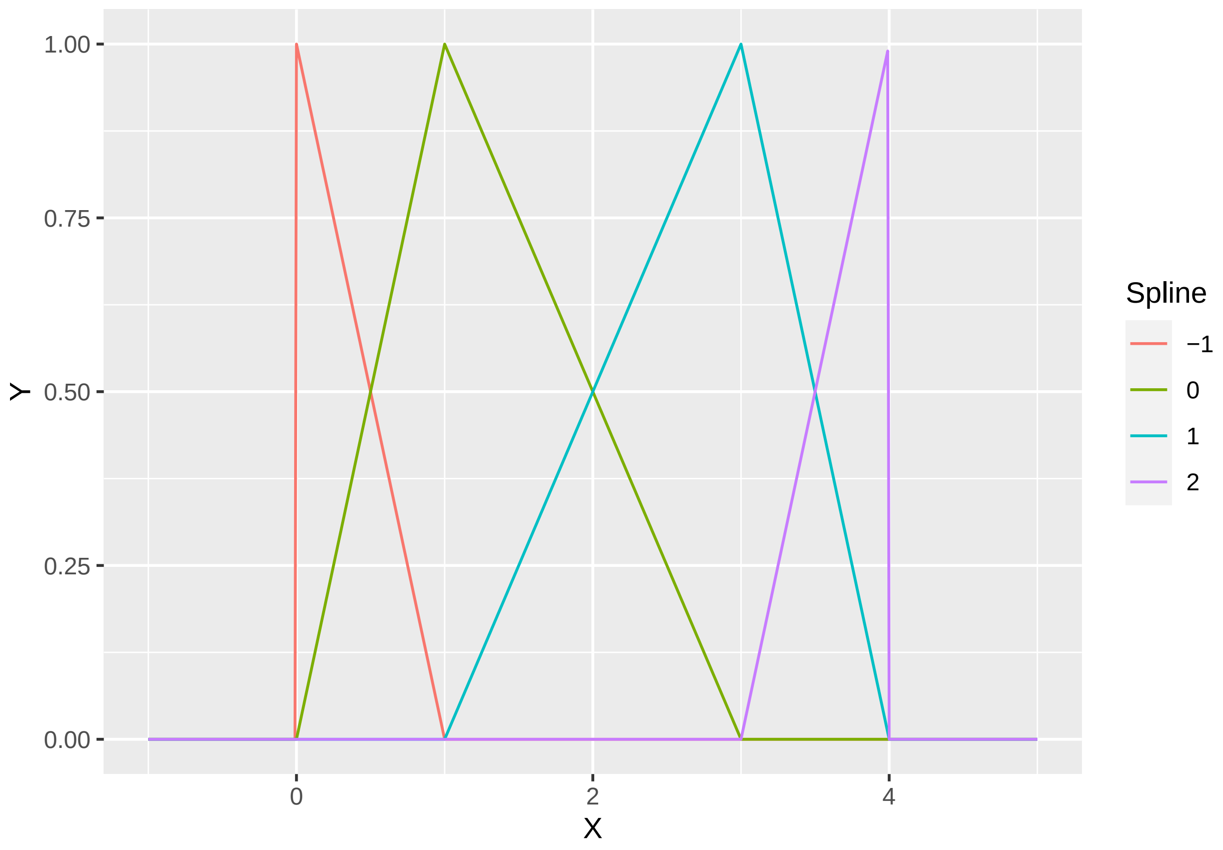


Figure 5: 2nd Order B-Spline Functions Represented Graphically