LUND
UNIVERSITY

Knot Optimization with Recursive Partitioning in Functional Data Analysis

Master Thesis in Statistics, 15 ECTS
STAN40, Spring 2021

Mattias Jönsson
Marcus Kleverman

Supervisor
Prof. Krzysztof Podgorski

Abstract

Functional Data Analysis (FDA) is a field that has been growing rapidly over the last few decades, with much ongoing research and many recent publications. The focus for Functional Data Analysis is the study of so called functional data, that is data that is assumed to have been generated from a true, underlying function, where we can only observe a sequence of measurements.

We propose a method that can identify a suitable placement of knots purely based on the data, without any predefined basis functions.

For functional data specifically, some recent research has been published where the authors introduce tree based methods for knot placement. In this thesis we continue that research and investigate the method further by fully following the regression tree paradigm with evaluation of different methods to avoid over-fitting. Throughout this thesis we will call this method Knot Optimization with Recursive Partitioning (KORP).

We have evaluated our method on both simulated data sets and on the MNIST handwriting data set and compared both with uniform placement of knots and a genetic algorithm for identifying optimal placement of knots.

Our conclusion from the study of the proposed method, is that the method works very well, both for simple data sets and for functional data. It generally performs better than both Uniform placement and Genetic Algorithms.

Contents

1 Introduction

Functional Data Analysis (FDA) is a field that has been growing rapidly over the last few decades, with much ongoing research and many recent publications. It is considered to have been solidified as a proper branch of statistics in the late 1990's, although some of the techniques have been around for much longer (e.g. *functional principal component analysis*) (Kokoszka and Reimherr, 2017).

The focus for Functional Data Analysis is the study of so called functional data, that is data that is assumed to have been generated from a true, underlying function, where we can only observe a sequence of measurements. This can for example be (Kokoszka and Reimherr, 2017, p.1):

- Concentration of a certain protein in the blood of a patient at a particular time - where we can measure the level only a few times per year
- Strength of magnetic field at a certain location at a particular time of day - measured at a very small interval
- Price for a particular stock at a particular time of day, again measured at for example hourly or minute level

Within the field of Functional Data Analysis, it is therefore relevant to analyse data in order to understand the underlying continuous functions. This can for example be approached for example by smoothing of the observed curves. Although this often is achieved by using for example fourier transformations, another common method is using so called *splines*. However, splines are dependent on placing so called *knots* at particular points in the range of the curves, and how to identify some ideal placement of these knots is still an open question within research.

The objective of this thesis is therefore to bring another piece of this puzzle, by the introduction and validation of a new method for identifying the optimal placement of knots. We aim to compare this method to both the "naive" placement, i.e. uniform, and a more sophisticated method.

Our main focus in this thesis will be only on the knot placement and their implications for a very simple curve fit. There are of course many other topics with FDA that would be interesting to study, for example the implications on *functional principal component analysis* and *curve smoothing*, but we will not cover these in this thesis.

2 Preliminaries

In this section we will present some of the fundamental topics and techniques that are encountered and used in this thesis. As has been outlined in the introduction to this thesis, we are mainly focused on the area of Functional Data Analysis, which is a field of statistics where data and observations appear in the form of sequences or curves.

The most fundamental set of observations in this field would according to Kokoszka and Reimherr (2017, p.1) be

$$Y_n(x_{j,i}) \in \mathbb{R},\ x_{j,i} \in [X_1, X_2],\ i = 1, 2, \ldots, n,\ j = 1, \ldots, J_n \tag{1}$$

This means that a set of n curves that are observed on a common interval $[X_1, X_2]$. These are assumed to come from some smooth curve

$$\{Y_i(x)\ :\ x \in [X_1, X_2], i = 1, 2, \ldots, n\} \tag{2}$$

The values $Y_i(x)$ exist at any point x, but are only observed at selected points $x_{j,i}$, and of particular interest is the shape of the observed curves - in essence used to make inferences about the true, unknown function. One can note that perhaps a more common notation in functional data analysis is to use $X(t)$ or $Y(t)$. However, the evaluation in this thesis is performed on datasets that are not recorded as observations over time (but rather over some area) we will stick with this slightly more generic notation.

There is naturally much to say about this topic, but in the preliminaries of this thesis we will limit the focus to the early stages in a typical functional data analysis process.

2.1 Splines and Knots

The use of splines in mathematics is often said to have been introduced around mid 1900´s by Schoenberg (1946), and the term comes from pieces of long, flexible strips of material that was fixed in position at a number of points ("knots") along the strip. These devices were called splines and often used by for example ship builders to draw smooth curves, by using the strip's tension that was generated by these knots.

This idea is replicated well in mathematics, and is used as an interpolation and/or smoothing method. In this chapter we will highlight some of fundamental aspects of splines and their theoretical foundation.

2.1.1 Piecewise linear functions

To introduce the fundamentals of splines, we will initially outline general interpolation with polynomials of varying degree. A piecewise constant function would in this context be the most basic representation, and we will return to this briefly in section 2.1.4. Introducing additional degrees, we get a piecewise linear function. To perform interpolation we have some general interpolating function

$$y = g(x). \tag{3}$$

The interpolating function consists of piecewise linear functions

$$g(x) = g_i(x),\ x_i \leq x \leq x_{i+1} \tag{4}$$

with a given arbitrary set of points on $(x_0, y_0), (x_1, y_1),\ \ldots\ , (x_n, y_n)$.

For a piecewise linear interpolation as seen in Figure 1, function 3 is continuous although the derivative of the function is not. To determine these piecewise linear g_i functions, let us look at a basic linear function 5. Here a_i is the slope and b_i is a constant.

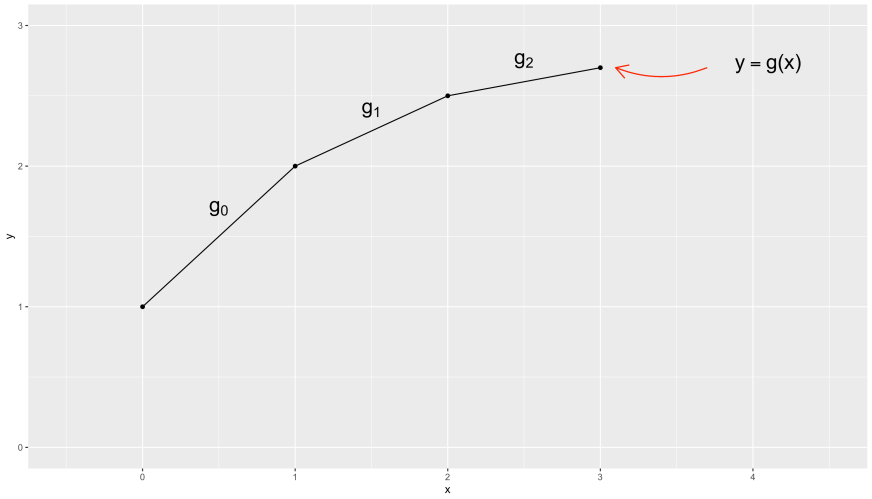


Figure 1: Example of piecewise linear functions

$$g_i(x) = a_i(x - x_i) + b_i \tag{5}$$

We know the value of $g_i(x_i)$. Since g_i has to go through the points as shown in the example in Figure 1, then $g_i(x_i) = y_i$. By the same principle we get that $g_i(x_{i+1}) = y_{i+1}$. This gives the two constraints

$$\begin{cases} g_i(x_i) = y_i \\ g_i(x_{i+1}) = y_{i+1}. \end{cases} \tag{6}$$

To solve for b_i in equation 5 we plug in x_i which together with the constraints 6 gives

$$b_i = y_i \tag{7}$$

and solving for a_i which is just the slope gives

$$a_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}. \tag{8}$$

2.1.2 Piecewise cubic functions

While X is one-dimensional, we can obtain a piecewise polynomial function $f(X)$ by dividing the domain of X into contiguous intervals and represent f by a separate polynomial in each interval.

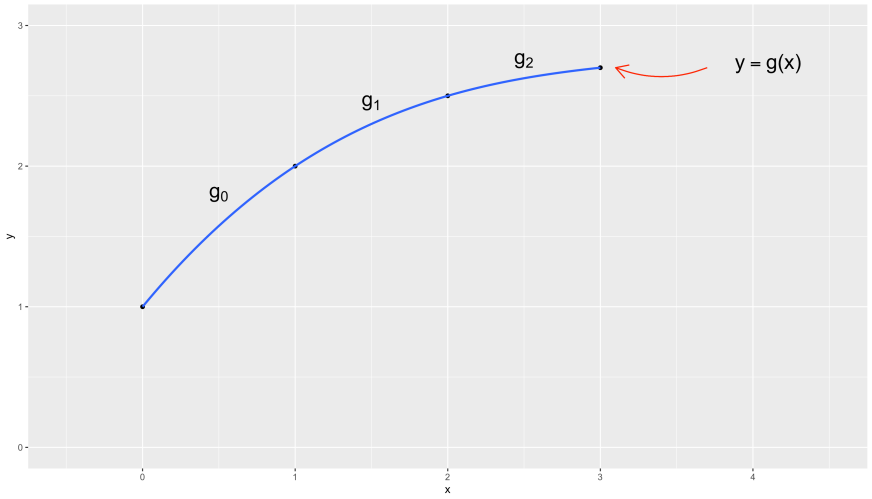


Figure 2: Example of piecewise cubic functions

$$g_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{9}$$

With $n + 1$ points we get n of these piecewise cubic polynomials as demonstrated in Figure 2. With 4 coefficients for each piecewise cubic polynomial we get $4n$ coefficients for this problem. To determine these $4n$ unknown coefficients we need $4n$ constraints. We have

- $g_i = x_i$ (n constraints),
- $g_{i+1} = x_{i+1}$ (n constraints),
- $g_i'(x_{i+1}) = g_{x_{i+1}}'(x_{i+1})$ (n-1 constraints),
- $g_i''(x_{i+1}) = g_{x_{i+1}}''(x_{i+1})$ (n-1 constraints),

which leaves us with $4n - 2$ constraints. The remaining 2 constraints can be solved with a system of equations which we will not go into detail in this thesis. For more details regarding the specifics of this topic, we recommend Boor (1978).

2.1.3 Basis expansions

A linear function is often a convenient approximation of the relationship of the data that one tries to model. It is quite easy to model and interpret although it is rarely that the true function $f(X)$ is linear. When moving beyond the linearity of piecewise functions we make use of what is called *basis expansions*. The main idea is to transform the original variables X into additional variables and then use linear models in these new input features.

Denote by $h_m(X) : \mathbb{R}^p \to \mathbb{R}$ the mth transformation of X, $m = 1, \ldots, M$. With this approach, once the basis functions h_m has been determined, the models are linear in these new variables and the relationship between X and Y can be modeled with

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X). \tag{10}$$

as a linear basis expansion in X.

A more in depth explanation of basis expansion can be found in Hastie, Tibshirani, and Friedman (2001).

2.1.4 B-splines

B-splines is short for *basis splines*. A spline function of order m is a piecewise polynomial of degree $m-1$ in a variable x. These piecewise polynomials meet at certain intervals known as *knots*. These knots are sections of the range that x spans and can be uniformly (equidistant) placed or by some other function depending on desired result or distribution of x. For a more in depth review of the theory described in this section, the reader is referred to Hastie, Tibshirani, and Friedman (2001) and Boor (1978).

Assume we have a number of internal knots $\xi_1, \ldots, \xi_K$. Add two boundary knots, ξ_0 and ξ_{K+1} where $\xi_0 < \xi_1$ and $\xi_K < \xi_{K+1}$. These boundary knots defines the domain over which we wish to evaluate our spline.

For an order m spline, add $m-1$ extra knots at each of the boundary knots. The values of these additional knots can be arbitrary but common practice is to make them the same value as the boundary knots. This can also be expressed as *knot multiplicity*, as described by Boor (1978), where the boundary knots are repeated M times which gives a total of $K+2M$ knots. Denote this knot vector by τ_i, $i = 1, \ldots, K+2M$ which is a non-decreasing sequence of real numbers. Denote by $B_{i,m}(x)$ the ith B-spline basis function of order m for the knot sequence τ with $m \leq M$ where $i = 1, \ldots, K+2M-m$.

If a knot in a sequence τ is duplicated and a B-spline sequence is generated as previously specified, the basis spans the space of the piecewise polynomials with one less continuous derivative at the duplicated knot. This is why the boundary knots are repeated M times which results in the splines becoming discontinuous and undefined beyond these points.

As described in Hastie, Tibshirani, and Friedman (2001), these are defined recursively in terms of divided differences as follows for a piecewise constant where $m = 1$, i.e, of degree 0. We have

$$B_{i,1}(x) = \begin{cases} 1, & \text{if } \tau_i \leq X < \tau_{i+1} \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

for $i = 1, \ldots, K+2M-1$ basis functions. These are also known as Haar basis functions, but we will not cover these in further detail in this thesis.

With higher order of smoothness, we furthermore have

$$B_{i,m}(x) = \frac{x-\tau_i}{\tau_{i+m-1}-\tau_i} B_{i,m-1}(x) + \frac{\tau_{i+m}-x}{\tau_{i+m}-\tau_{i+1}} B_{i+1,m-1}(x) \tag{12}$$

for $i = 1, \ldots, K+2M-m$ basis functions.

For a cubic B-spline basis function we will have $m = 4$ which will result in $B_{i,4}$, $i = 1, \ldots, K+4$ for the specified knot sequence. The recursion specified in equation 12 will generate the B-spline basis for any order spline.

From the definition in equation 11 and 12 we can deduce that the B-spline $B_{i,m}(x)$ depends only on the elements $\tau_i, \ldots, \tau_{i+m}$ in the knot vector τ. It is also positive within these elements and zero otherwise. This is visualized in Figure 4 with B-splines of order m $= 1, \ldots, 4$.

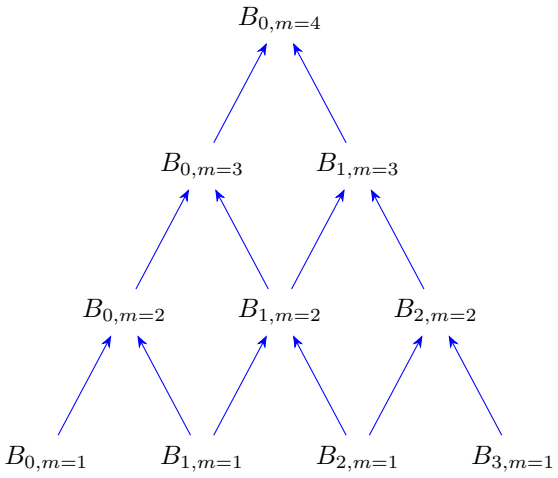


Figure 3: DeBoor

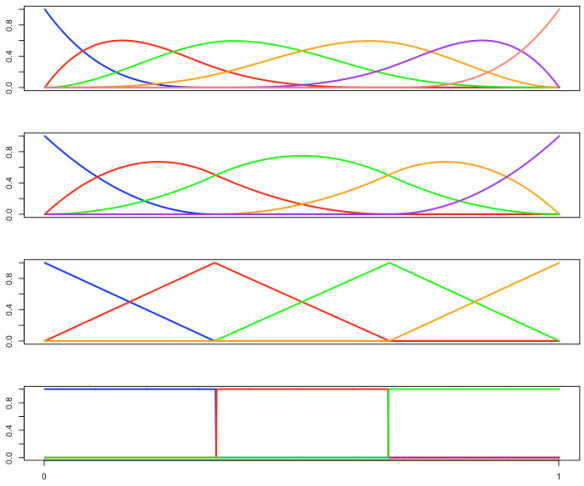


Figure 4: B-spline Basis Functions of orders 1 to 4

Equation 11 acts as a base case or switch function and as noted can take the value 0 or 1. The basis functions can be written as a triangular scheme which is visualized in Figure 3 where the top level $B_{0,m=4}$ is the product of all underlying levels. This is an important property of the recursion formula where division and multiplication by 0 is avoided and thus improves the computational cost.

2.2 Tree Based Method

Several authors, for example Hastie, Tibshirani, and Friedman (2001) and James et al. (2013) presents comprehensive introductions to tree based methods for statistical learning, which is summarised briefly here.

Tree based methods for regression and classification are based on the idea is to segment the predictor space into smaller partitions which are more isolated and thus more simple. These methods are available in many different variants and with many extensions and applications, like Random Forests, Boosted Tree Ensembles, etc.

The idea is that the predictor space is partitioned into M distinct and non-overlapping regions $R_1, R_2, \ldots, R_M$. In each of these regions a constant $\hat{y}$ will be our prediction estimate, which basically means that the entire set of regions can be seen as a piecewise constant function. That means that we want to find the regions that minimise the total prediction error

$$\sum_{m=1}^{M} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2. \tag{13}$$

For growing regression trees we recursively identify the best cutpoint s such that the resulting regions $R_1(j,s) = \{X | X_j < s\}$ and $R_2(j,s) = \{X | X_j \geq s\}$ lead to the greatest reduction in RSS after a split, i.e.

$$\sum_{i:\ x_i \in R_1(m,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\ x_i \in R_2(m,s)} (y_i - \hat{y}_{R_2})^2. \tag{14}$$

Recursively performing these splits until nothing remain to split, will however means overfitting to the data. For this reason, one can use a pruning method to reduce a too large (and overfit) tree, but also an early stopping criterion that stops further partitioning based on some criteria.

Pruning can often be performed by calculating cost complexity metrics for each subtree, and eliminate subtrees with too high metric value. One common cost complexity metric is for example calculated as

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T| \tag{15}$$

where α is a penalty parameter and $|T|$ denotes the number of terminal nodes in the tree. A suitable parameter α is then identified either through cross validation or using a test/holdout data set.

2.3 Genetic Algorithms

In this section below, we will briefly introduce *genetic algorithms* - an optimization method (or perhaps rather a family of methods) that has been evaluated in this thesis, mainly to be used as a reference for the evaluation of the tree based method. The focus in this section will be to convey the main concepts and give the reader a general understanding of the reasoning behind the family of genetic algorithms.

Genetic algorithms is a family of optimization methods inspired by the evolution in nature. The methods try to capture principles like child inheritance from parents, overall survival of the fittest and random introduction of mutations. The method is stochastic and is not using gradients, and could therefore be suitable in cases where the underlying problem is either not differentiable, or where the function to optimize is not very well behaved (Goldberg, 1989, p.2).

Furthermore, Goldberg (1989) presents a thorough and comprehensive detailing of the concepts and this entire section is almost exclusively based on it. The main exception is the part with varying dimension, that we will get into shortly.

An overview of the method is as follows:

Algorithm 1: Genetic Algorithm

Result: A set of individuals/genomes
Let $f(x)$ be a function to minimize;
Let *pop* be a population of n individuals with a random set of x_i, representing the individual's genome in generation $g_{t=1}$;
Function `GeneticAlgorithm()`

1. Evaluate $f(x)$ for each individual genome
2. *Breed*() generates a new set of individuals/genomes to form generation g_t by
   - Randomly selecting individuals from g_t where the probability of selecting the individual is proportional to $f(x)$
   - Let two individuals create a new individual with a genome built up from different parts of its parents genome
   - Randomly alter the newly created individual to introduce some mutations
3. Repeat step 2 until convergence

The *Breed()* function exists in two slightly different versions in this thesis. Alternative 1 is when we have the function

$$f(x) \ : \ \mathbb{R}^p \ \rightarrow \ \mathbb{R} \tag{16}$$

which is used in this thesis in a way, where both parts of $\mathbb{R}^p$ are included in the optimization, i.e. we let p vary, and call this *varying dimension*. This is something that several authors have introduced, for example Brie and Morignot (2005) and Pawar and Bichkar (2015).

Alternative 2 is instead used when the dimension is fixed, and we have the discrete case of

$$f(x) \ : \ A \in \{0,1\}^d \ \rightarrow \ \mathbb{R}. \tag{17}$$

To go through some of these points in a little more detail, we can start with explaining the breeding method:

Let X_1 be the genome for the first parent and X_2 for the second.

Let X_{i_j} be the value of chromosome j for individual i

Let Φ be a vector of length $max(|X_1|, |X_2|)$ of Bernoulli random values with a given probability p

If Varying Dimension-variant:

Let P_{drop} be the probability that a chromosome in a genome is dropped.

Let P_{add} be the probability that a random chromosome is added to the genome.

Let P_{mutate} be the probability that a chromosome is mutated and Φ_m be a vector of length of the genome of the offspring containing Bernoulli random values with the probability P_{mutate}

Let $X_{mutation}$ be a vector of length of the genome of the offspring of random $N(0, \sum_i \sum_j Var[X_{i_j}])$ if X_{i_j} is continuous and random Bernoulli variables if X_{i_j} is discrete.

An offspring from two parents can then be calculated as

$$X'_{offspring} = \Phi X_1 + |\Phi - 1| X_2. \tag{18}$$

After an offspring is bred, we may want to be able to vary the number of dimensions used. Therefore, in this case a random number $U(0,1)$ is generated and if $U(0,1) < P_{drop}$ one of the available chromosomes are dropped. The same procedure is repeated for potentially adding a dimension.

The final offspring in the *varying dimension* configuration is then

$$X_{offspring} = X'_{offspring} \Phi_m X_{mutation}. \tag{19}$$

3 Knot Optimization with Recursive Partitioning

Previous research on general knot placement in splines (not specifically for functional data) has proposed methods for knot placement by for example using a Reversible-jump Markov Chain Monte Carlo algorithm (see Green (1995) for RJMCMC in general and Dimatteo, Genovese, and Kass (2001) for an application on fitting splines through knot placement). However, as far as we can tell, most research here has focused on the case where the choice of basis functions is already defined up front and known for the optimization algorithm. We want to instead propose a method that can identify a suitable placement of knots purely based on the data, without any predefined basis functions.

For functional data specifically, some recent research has been published (for example Basna, Nassar, and Podgorski (2021) and Nassar and Podgórski (2021)), where the authors introduce tree based methods for knot placement. In this thesis we continue that research and investigate the method further by fully following the regression tree paradigm, including for example evaluation of different methods to avoid overfitting. Throughout the thesis we will call this method Knot Optimization with Recursive Partitioning (KORP).

One of the first things to mention regarding our proposed additions to the method, is that we make a clear distinction between a *split point* in the tree based method, and the subsequent knot placement. This is something that Basna, Nassar, and Podgorski (2021) and Nassar and Podgórski (2021) does not do, and we assume that it is done at least partly due to properties of *Splinets*. However, in our work we utilize B-splines which are not orthogonal, and therefore potentially have slightly different implications on spline fit. However, we believe that knot placement with our proposed method enhancements could be beneficial also for *Splinets*.

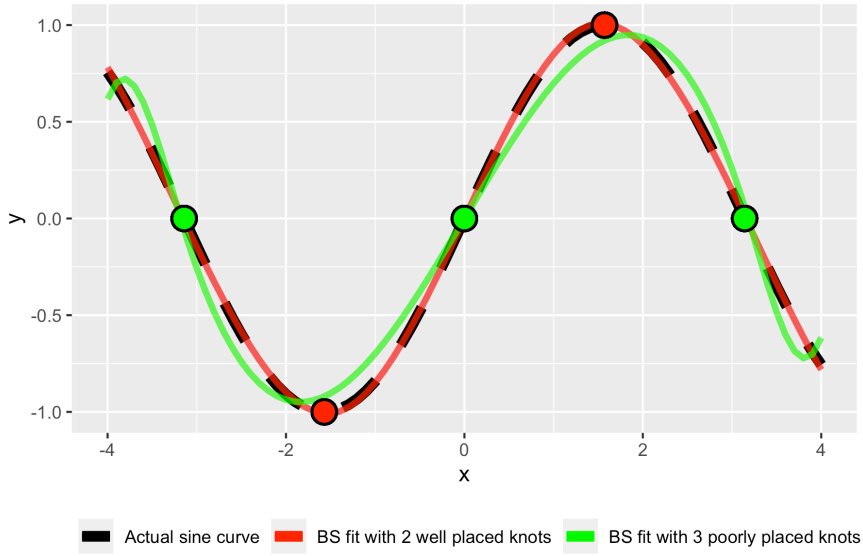


Figure 5: Difference in Knot Placement vs Tree Split Points

We also believe that the distinction between split point and knot placement is highly relevant, especially in a function data analysis context. One can show the importance of this in a simple example: say we have a simple sine curve without any randomness, and we try to fit a piecewise constant over a given range using a tree based method. The split points from a very small tree with for example three split points, would be placed approximately as the green points are in figure 5. The green line then represents a B-spline fit using those three split points as knots, and as one can see, the fit is certainly not very good. Our goal is instead to be closer to red knots in the same figure, where only two knots are needed to very accurately fit the curve.

Of course, in reality with more complex data sets, it is going to be more difficult to find some globally optimal knot placement, and it may not even be necessary in functional data analysis, since there are usually many other analysis techniques that follows the spline fits. However, we argue that fewer knots are generally better, and that we at least want the most reasonable knot placement that can easily be identified. In the case of knots being placed only at split points, to be able to achieve a decent fit in this toy example, it would require more than twice as many knots. There is more to say on this topic, and we will have the opportunity to return to this later in the thesis, for example in section 5.1.3.

Performing recursive splitting until each leaf contains a single data point, means a "complete" tree that likely to generalize very poorly to other samples (i.e. the full tree will have overfit to the data). On this topic, we also introduce a novel method to allow seemingly low value splits early in the tree while still avoiding unnecessarily complex branches in the tree. This we will go into more details in section 3.2.

Furthermore, we use only a training subset of the full dataset to fit the tree, and use another subset of data for validation (a test set). Evaluation is done using Mean Squared Error, performed for both training and test set, which can be used to choose a reasonable number of split points based on the MSE profile over the number of splits. Given the number and individual splits, knots can be placed relative to these points.

Our proposed method can be outlined as follows on a set of observations with with two components, say x and y, or in the case of functional data x_i and y_i (where i is the ith sequential measurement in an observation):

1. Perform recursive two-way splits, each one at the best location (details follows below), until a node contains too few observations.
2. For all individual splits identified in step 1, evaluate the increase in total mean squared error associated with the removal of the split and store the reduced model.
3. Repeat 2 using the reduced model, until no splits remain (i.e. the model is a constant)
4. For every model stored as part of step 2, calculate the overall total mean squared error for y and plot.
5. Evaluate the plot and look for a significant "elbow" in the curve, which can be seen as a suitable number of split points.
6. Given the identified number and placement of split points, place knots based on these. The knots are placed exactly at the midpoint between two splits.

3.1 Performing Recursive Splits

Recursive splits will be performed on a dataset with n functional observations to identify M distinct and non-overlapping regions $R_1, R_2, \ldots, R_M$. As outlined in section 2.2, in each of these regions y is estimated with a constant $\hat{y}$, that is

$$\hat{y}(x) \;=\; \sum_{m=1}^{M} c_m I(x \in R_m). \tag{20}$$

The piecewise constant can as Basna, Nassar, and Podgorski (2021) and Nassar and Podgórski (2021) points out, also be seen as a spline of degree 0 (i.e. order 1). Here, the c_m constant per region R_m is simply calculated by the average across the region and across all observations. That is

$$c_m \;=\; \frac{1}{n}\sum_{i=1}^{n}\frac{1}{n_{R_m}}\sum_{j=R_{m_L}}^{R_{m_U}}(y_i(x_j)) \tag{21}$$

where R_{m_L} is the index of x that that is the lower bound of R_m, R_{m_U} is the upper bound, and n_{R_m} is the total number of the functional observations measurements in x that are in the region R_m, i.e. $R_{m_U} - R_{m_L}$. Looking at other authors, for example Kokoszka and Reimherr (2017, p.49), it seems reasonable to here replace equation 13 with *total mean squared error*, calculated as

$$MSE_{Total} \;=\; \frac{1}{n}\sum_{i=1}^{n}\int (y_i(x) - \hat{y}(x))^2\,dx \;=\; \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J}(y_i(x_j) - \hat{y}(x_j))^2. \tag{22}$$

For growing the tree we recursively identify the best split point s such that the resulting regions lead to the greatest reduction in total mean squared error (MSE_{Total}). Expressed as a function we have

$$g(\{R_1, \ldots, R_M\}, s) \;=\; \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J}(y_i(x_j) - \hat{y}(x_j))^2 - \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J}(y_i(x_j) - \hat{y}^*(x_j))^2 \tag{23}$$

where $\hat{y}^*(x)$ is the piecewise constant prediction when including the new split point s, for a new set of regions $R_1, R_2, \ldots, R_M, R_{M+1}$.

3.2 Avoiding Excessively Deep Branches

Including an excessive number of splits in the tree means that the algorithm has overfit to the data. Therefore we want to limit the complexity, i.e. number of splits in the tree. Two common ways to do this are *early stopping* and *pruning*. Since we want to visually evaluate the MSE_{Total} decrease and look for an "elbow" in the curve, we will in this work focus on early stopping.

A very common way to do this early stopping is to use a threshold value for minimum MSE_{Total} improvement to perform a split. This is for example something that Basna, Nassar, and Podgorski (2021) proposes to use in the *DDK* method. However, it is known that this may result in valuable splits being missed (Hastie, Tibshirani, and Friedman, 2001, p. 308). Other options for avoiding complex trees, is to limit the number of levels (Boehmke and Greenwell, 2019). However, seeing that trees may be very skewed, especially so when applied to functional data, a single fixed limit to the number of levels for all branches, may be too inflexible.

Based on these ideas, we propose using a method of an adaptive penalty α, that we can use in the following way. Again using total mean squared error (MSE_{Total}), we have a value for this prior to a certain split, and the value this would take if the split was performed (call this MSE^*_{Total}. We can then have a condition to only perform the split if

$$\frac{MSE^*_{Total}}{MSE_{Total}} \leq (1-\alpha)^{level}. \tag{24}$$

Since this means the condition will be more and more restrictive as more splits are performed we have an adaptive threshold, that results in a better possibility to allow some splits early in the tree with a small MSE_{Total} reduction, whereas the further we get in building the tree, the less likely we are to accept these small MSE_{Total} reductions. We believe this is a novel idea, and has in our experimentation proven to be a very good way to reduce the amount of (unnecessary) computations, while still ensuring that early splits are performed.

3.3 Identifying Optimal Split

In the description above, the function $g(\{R_1, \ldots, R_M\}, s)$ calculates the decrease in MSE_{Total} at a certain split point s, given an existing set of M split points. To go into more detail how this is applied in practice, we have two different scenarios:

3.3.1 x is discrete or x is continuous with low cardinality

If x is discrete with values

$$x \in \{x_1, x_2, \ldots, x_j\} \tag{25}$$

the possible split points are only

$$x_{split} \in \{\frac{x_1 + x_2}{2}, \frac{x_2 + x_3}{2}, \ldots, \frac{x_{j-1} + x_j}{2}\}. \tag{26}$$

In this case $g(\{R_1, \ldots, R_M\}, s)$ can be evaluated for each possible x_{split} at each recursive split. Often, functional data would likely fall in this category (Kokoszka and Reimherr, 2017, p.1)

3.3.2 x is continuous with high cardinality

If $x \in \{x_1, x_2, \ldots, x_j\}$ is observed with large j, it may be computationally prohibitive to evaluate $g(x)$ for each possible x_{split} at each recursive split. In that case, we could see minimizing $g(\{R_1, \ldots, R_M\}, s)$ as an optimization problem.

However, it is not a very straightforward optimization task. The function $g(\{R_1, \ldots, R_M\}, s)$ is first of all non-convex as is presented in Figure 6. As can also be noted in the same figure, the function is also not smooth (seen most clearly around the lowest points in the troughs). That means that using the derivative $g'(\{R_1, \ldots, R_M\}, s)$ as in some quasi-Newton method, would most likely pose great challenges.

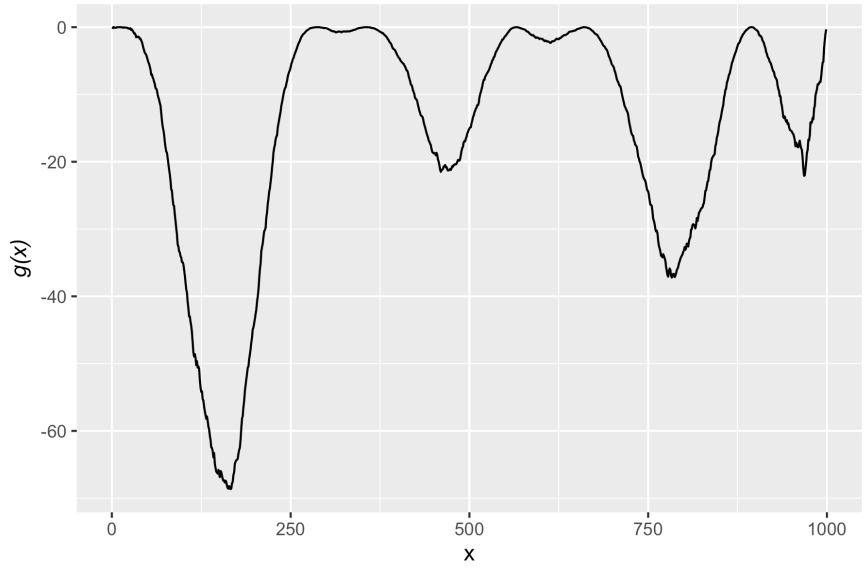


Figure 6: $g(M, s)$ for the first split of a simulated dataset

Instead we have to look at other optimization methods, particularly ones that do not use gradients. In our development of this method, we have evaluated several of these methods and have found most promising results (computational efficiency and ability to find a global minimum) with an algorithm called Particle Swarm Optimization (PSO), originally introduced by Kennedy and Eberhart (1995). Some amount of detail about the PSO algorithm can be found in Appendix B, but we will not delve deeper into this subject, since it is likely not a big issue with functional data.

3.4 Algorithm for Recursive Splits

Using the defintions of $g(\{R_1, \ldots, R_M\}, s)$ and MSE_{Total} from above, our proposed algorithm for performing recursive splits of the data is presented in Algorithm 2:

Algorithm 2: Recursive Splits

Result: A recursive tree with split points in each node
Let α be a parameter that controls the penalty for stopping greedy search for split points.
Let *nDataPts* be a number of data points where splitting stops.
Level $\leftarrow$ 0
Function `Split(`*x, y, level*`)`
  Let splitpt $\leftarrow \arg\max_s g(M, s)$
  Let lh.x $\leftarrow \{x \mid x < splitpt\}$
  Let lh.y $\leftarrow \{y(x) \mid x < splitpt\}$
  Let rh.x $\leftarrow \{x \mid x \geq splitpt\}$
  Let rh.y $\leftarrow \{y(x) \mid x \geq splitpt\}$
  if $\frac{MSE^*_{Total}}{MSE_{Total}} \leq (1-\alpha)^{level}$ **then**
    Let LH $\leftarrow$ null
    Let RH $\leftarrow$ null
    if $n_{lh} \geq nDataPts$ **then**
      Let LH $\leftarrow$ Split(lh.x, lh.y, level+1)
    end
    if $n_{rh} \geq nDataPts$ **then**
      Let RH $\leftarrow$ Split(rh.x, rh.y, level+1)
    end
    return {splitpt, LH, RH}
  else
    return null
  end

3.5 Pruning the Tree

We are in this thesis mainly interested in the case of functional data (where we have a curve of measurements over a range of x), with the purpose of knot placement. It is likely that the full tree identified in the previous step, contains too many split points and would likely not generalise well to for example a test set. We therefore propose that the full tree is pruned back to a smaller set of split points, by iteratively eliminating the split point with the smallest impact of $\hat{y}(x)$ until every split point has been excluded.

We can describe this as a function $g^*(\cdot)$, as

$$g^*(\{R_1, \ldots, R_M\}, s^*) \;=\; \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J}(y_i(x_j) - \hat{y}(x_j))^2 - \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J}(y_i(x_j) - \hat{y}^\dagger(x_j))^2 \tag{27}$$

where $\hat{y}^\dagger(x_j)$ is the prediction constant of the tree, when the cut point s^* has been removed. This function is then iteratively used as

$$\arg\max_{s^*} g^*(\{R_1, \ldots, R_M\}, s^*) \tag{28}$$

in order to eliminate the least valuable split point s^*. To be very clear here, at this point any of the split points used to build the tree can be removed, not only leaves.

3.6 Identifying suitable number of splits and placement of knots

Using the iteratively calculated values for $g^*(\cdot)$ from previous steps, we can plot the MSE_{Total} over the number of splits in the tree. In this we should find an "elbow", which gives a suitable number of splits. We can however look at an example of how a B-spline fit improves as more knots are added. As can be seen in Figure 7 (which is a variation on the plot presented later in Figure 19 in section 5.1.2), we get a good fit of a B-spline already with reasonably few knots. This means one would probably want to opt for an early "elbow", rather than a later. However, this is in contrast with the context of functional data, where an excessive number of knots is no a major issue, due to other methods following the spline fit.

Figure 7: MSE per number of splits, and MSE of B-spline fit with corresponding knots

Given the number of splits, we also need to decide where to place the knots in relation to the split points. In this work, we evaluate both

- Knots at split points
  i.e given a set S of $M-1$ split points, we have knots at $\tau_i = S_i$.
- Knots centered between split points
  i.e given a set S of $M-1$ split points, we have knots at $\tau_i = \frac{S_{i-1}+S_i}{2}$.

4 Evaluation of Proposed Method

There is no obvious comparable benchmark to evaluate against. Instead we will evaluate our method with two different approaches:

- **Simulation**
  When evaluating against simulated data, we will use identified knot placements to fit splines to then evaluate the goodness of fit. Evaluation will be done mainly with visual inspection of the spline fit. Furthermore, although the goal is for this method to be applied in a functional data context, the evaluation in this step will be of more fundamental nature. The generated datasets will be well structured and simple to validate that the method works for the simple cases, before moving on to more advanced situations. Therefore, the dataset generated can be seen as a single functional observation, with a large number of measurements.
- **MNIST Handwriting Data**
  To validate the method's applicability in the main domain of interest, we will apply it to functional data. We will use the method to represent a set of images of handwritten zeroes, using splines based on the identified knots. We will also compare this fit with splines based on uniform knots.

In the development of the method and subsequent experimentation for evaluation, all code has been built in R, using only basic, built-in functions.

4.1 Evaluation by Simulation

Evaluation on simulated data will be performed for a number of different datasets. Through this simulation, we will generate a functional observation with 1000 measurements of $y(x)$ to form a "training" dataset, and another observation with 100 measurements to act as a "test" set. We will simulate the following:

4.1.1 Piecewise Constants

Our x will be random $Unif(0, 1)$, and y will be piecewise constant with the following structure:

$$\begin{cases} y = \mathcal{N}(\mu = 0.3, \sigma = 0.1) & x \leq 0.3 \\ y = \mathcal{N}(\mu = 0.5, \sigma = 0.1) & 0.3 < x \leq 0.5 \\ y = \mathcal{N}(\mu = 0.7, \sigma = 0.1) & 0.5 < x \leq 0.7 \\ y = \mathcal{N}(\mu = 0.5, \sigma = 0.1) & x > 0.7 \end{cases} \tag{29}$$

The data is simulated, and one example is seen in Figure 8.

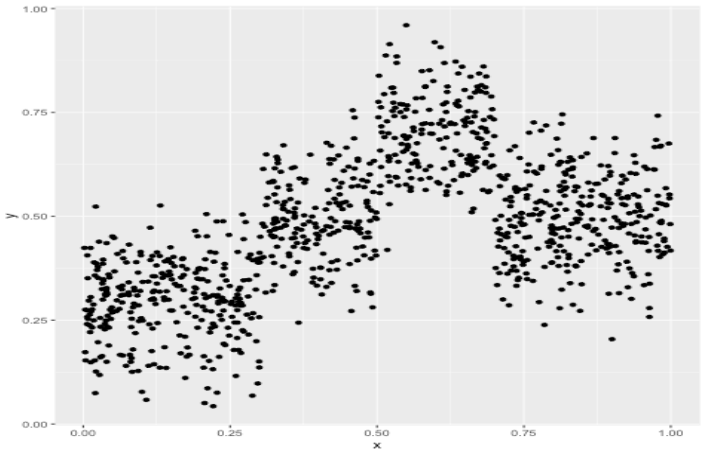


Figure 8: Dataset - Piecewise Constant

4.1.2 Piecewise Linear

Our x will be random $Unif(0, 1)$, and y will be piecewise linear with the following structure:

$$\begin{cases} y = 2x + \epsilon & x \leq 0.3 \\ y = -0.5x + 0.7 + \epsilon & 0.5 < x \leq 0.7 \\ y = 1.3x - 0.4 + \epsilon & x > 0.7 \end{cases} \tag{30}$$

where $\epsilon \sim \mathcal{N}(\mu = 0, \sigma = 0.1)$

The data is simulated, and one example is seen in Figure 9.

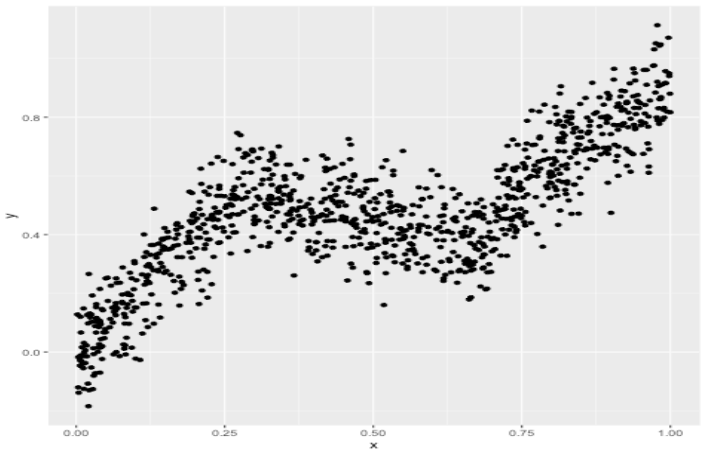


Figure 9: Dataset - Piecewise Linear

4.1.3 Piecewise Exponential

Our x will be random $Unif(0, 1)$, and y will be piecewise constant with the following structure:

$$\begin{cases} y = e^{5x} + \epsilon & x \leq 0.5 \\ y = e^{-5x+5} + \epsilon & x > 0.5 \end{cases} \tag{31}$$

where $\epsilon \sim \mathcal{N}(\mu = 0, \sigma = \frac{1}{2} + \frac{\log y}{2})$

The data is simulated, and one example is seen in Figure 10.

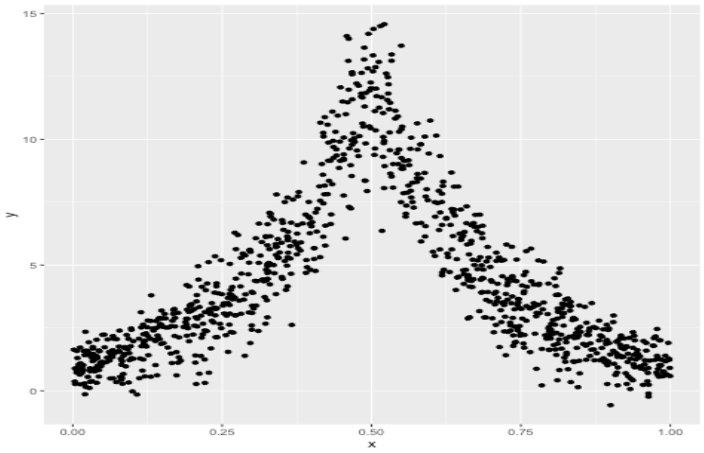


Figure 10: Dataset - Piecewise Exponential

4.1.4 Sine curve with constant frequency

Our x will be random $Unif(0, 1)$, and y will be sine wave with the following structure:

$$y = sin(20x) + \epsilon_i \tag{32}$$

where $\epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 0.4)$

The data is simulated, and one example is seen in Figure 11.

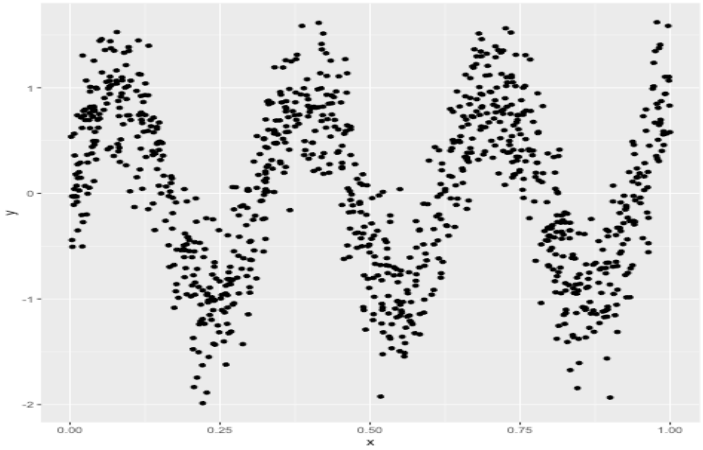


Figure 11: Dataset - Sine wave

4.1.5 Two sequential sine curves with different frequencies

Our x will be random $Unif(0, 1)$, and y will be two combine sine waves with the following structure:

$$\begin{cases} y = sin(20x) + \epsilon_i & x \leq 0.85 \\ y = sin(50x) + \epsilon_i & x > 0.85 \end{cases} \tag{33}$$

where $\epsilon_i \sim \mathcal{N}(\mu = 0, \sigma = 0.4)$

The data is simulated, and one example is seen in Figure 12.

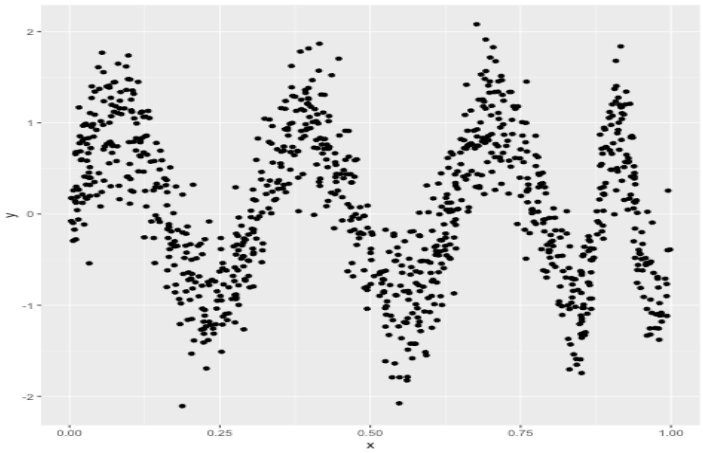


Figure 12: Dataset - 2 Sine Waves

4.1.6 Comparison with Uniform and Genetic Algorithms

We will also for one of these datasets, compare the tree method with Uniform knot placement, and knot placement performed by a genetic algorithm.

In this part of the evaluation, the *varying dimension* variant will be used. This means that in the fitness function (i.e. the function $f(\cdot)$ to optimize) has the following mapping

$$f \ : \ \mathbb{R}^p \ \rightarrow \ \mathbb{R} \tag{34}$$

and we let p represent the number of knots and the p dimensional real valued vector contains the knot placements.

The specific $f(\cdot)$ used in this evaluation is almost equivalent to MSE_{Total} presented earlier, but here, a penalty term is included to avoid an excessive number of knots. The penalty term is simply based on the

number of knots. We have

$$f(\cdot) \;=\; \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{J}(y_i(x_j) - \hat{y}(x_j))^2 + \alpha k \tag{35}$$

where n is the number of observations, k is the number of knots and α is a penalty factor. $\hat{y}(x)$ is just as before calculated as the total mean value for the region m, for all observation measurements and all observations (see equations 20 and 21). Furthermore, in this thesis the penalty factor is manually chosen so that the number of knots proposed by the genetic algorithm matches the number identified with the tree based method, in order to evaluate only the actual placement.

4.2 Evaluation on MNIST Handwriting Data

To evaluate the method on functional data, we will apply it on the MNIST database (Modified National Institute of Standards and Technology Database) of handwritten digits. This database contains 70.000 instances of handwritten digits, originally made available by LeCun et al. (1998). The images in the databases are grayscale images, with a resolution of 28*28 pixels. Some examples of what these digits look like can be seen in Figure 13. Furthermore, in this study we will only study the observations that represent zeroes in the dataset.

Figure 13: MNIST Database Sample

Each observation consists of a $28 * 28$ matrix of pixel intensities in the range $[0, 255]$. However, in order to make this feasible and suitable for our proposed method, we will transform the matrix from $28 * 28$ to $1 * 784$. This will give us a one dimensional structure and an example of the curve representing an observation can be seen in Figure 14.

Now, given the set of n observations (curves) $y_1^*(x),\ y_2^*(x),\ ...,\ y_n^*(x)$, where x denotes the pixel ordinal, we will preprocess each $1 * 784$ observation by subtracting the mean.

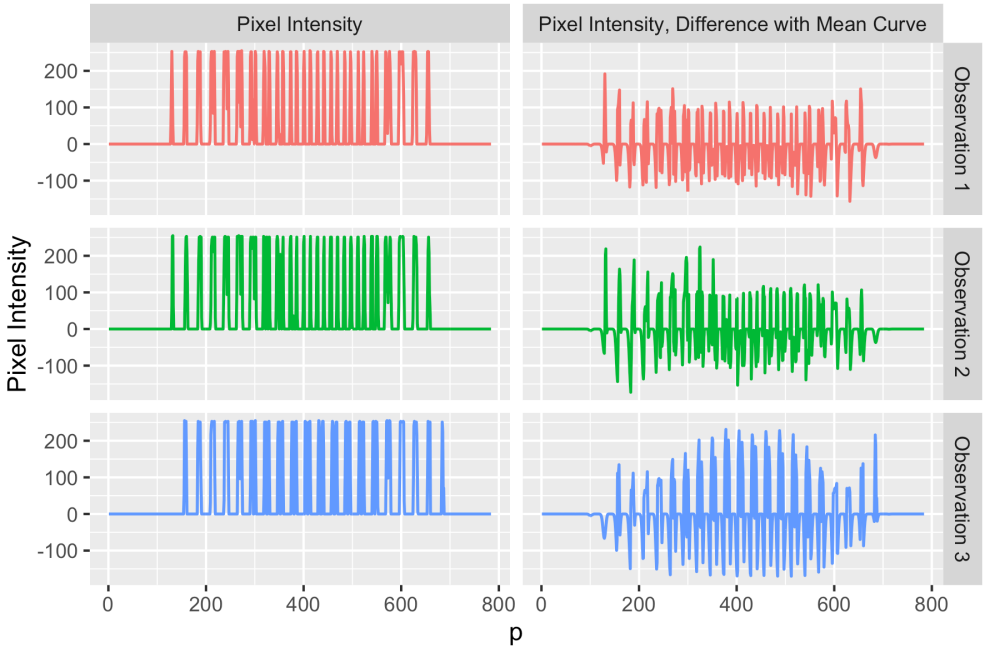


Figure 14: MNIST - Three observations flattened and mean differenced

$$y_i(x) = y_i^*(x) - \mu(x) \tag{36}$$

Where the mean curve is

$$\mu(x) = \frac{1}{n}\sum_{i=1}^{n} y_i^*(x) \tag{37}$$

In essence, this gives us the result as is indicated in Figure 15. At the left, we have an example of a digit in the dataset. In the middle, we have the average zero in the dataset overall, and to the right, one can see the observation difference from the mean. Here, the positive differences are highlighted in green, and negative differences are highlighted in red.

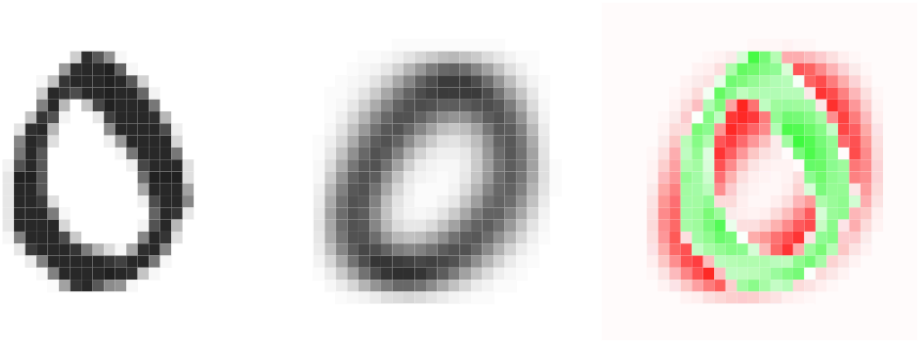


Figure 15: Mean "Zero" and Difference between Observation 1

After the mean curve has been subtracted, we will fit the knot tree according to the proposed method, and a reasonable number of knots will be selected. Finally, we will visualise the B-spline fits for a number of observations, and also compare the difference in MSE_{Total} between our method, a genetic algorithm and an equal number of knots, placed uniformly over the range of $[1, 784]$.

The genetic algorithm is here the discrete version and will be using a fitness function (i.e. the function $f(\cdot)$ to optimize) with the following mapping

$$f \ : \ A \in \{0,1\}^{784} \ \rightarrow \ \mathbb{R} \tag{38}$$

where $A \in \{0,1\}^{784}$ contains a 1 if a knot is placed on this point in the range of x, and 0 if not. That means we can use the same $f(\cdot)$ as in the continuous case, i.e. equation 35 since $A \cdot \begin{bmatrix} 1 & 2 & \dots & 784 \end{bmatrix}$ is going to give us the set of split points needed to calculate c_m according to equation 21. Again, the penalty term in equation 35 is chosen to identify the same number of knots as is identified in the tree based method.

5 Results

5.1 Simulation

Going through every step in our proposed method of every simulated dataset will result in excessive detail and much repetition. Instead, a complete listing of all plots for all simulated datasets are available in Appendix A. In this section we will instead focus only on some of the most relevant findings. Relevant plots will however still be included here, to make the reading of this section easier for the reader.

5.1.1 Building the Full Tree

When we first of all build the full trees, doing so without any penalty term for number of splits would become very large with excessive splits. For this reason, each full tree has been fitted with suitably identified penalties, that however still retain a reasonably large number of splits. The reason for the conservative penalty, is to ensure that the rest of the method still is applicable.

We can now also visualise how the number of split points is affected by the value of the penalty, and we see that even for low values of α we often get some reduction, see Figure 16

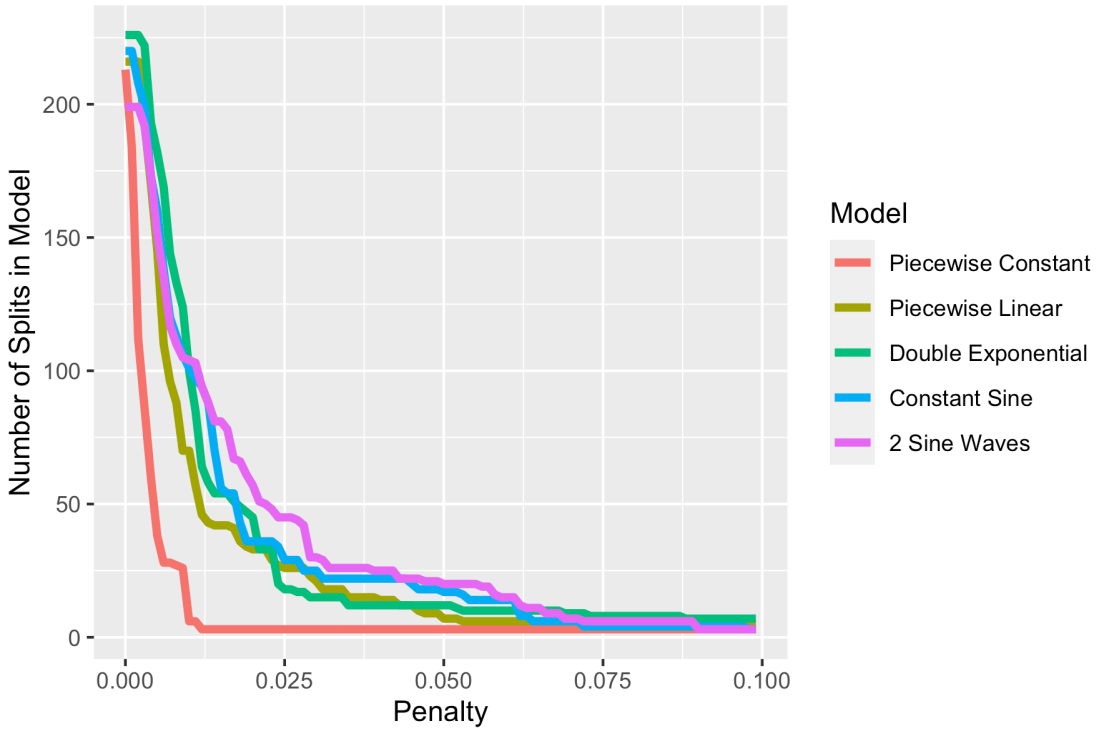


Figure 16: Number of Splits per Penalty value

As we can also see in Figure 16, the reduction in split points is relatively similar across the datasets. This implies that it may be possible to have some suggested standard value of α that is suitable at least as a starting point for the researcher. The penalty terms that have been used in fitting the full trees in this work are presented in 1, and may give some hint towards a reasonable suggestion.

If we study some of the plots of the dataset with the predictions from each respective full tree, we can for example in Figure 17, see that the full model tree produces many splits, but of course splits that follow the

Dataset	Penalty
Piecewise Constant	0.001
Piecewise Linear	0.005
Double Exponential	0.001
Constant Sine	0.005
2 Sine Waves	0.005

Table 1: Penalties Applied in Full Tree

data very well. The many short red line segments in the plot indicate the piecewise constant function that the full tree gives us.

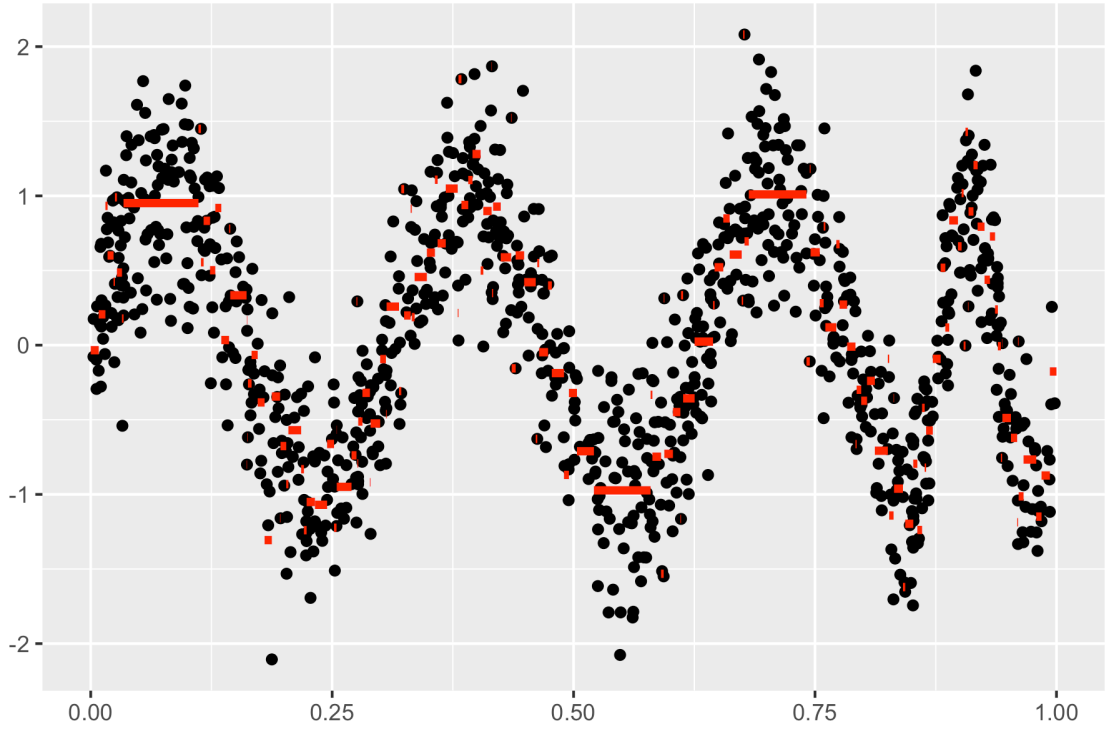
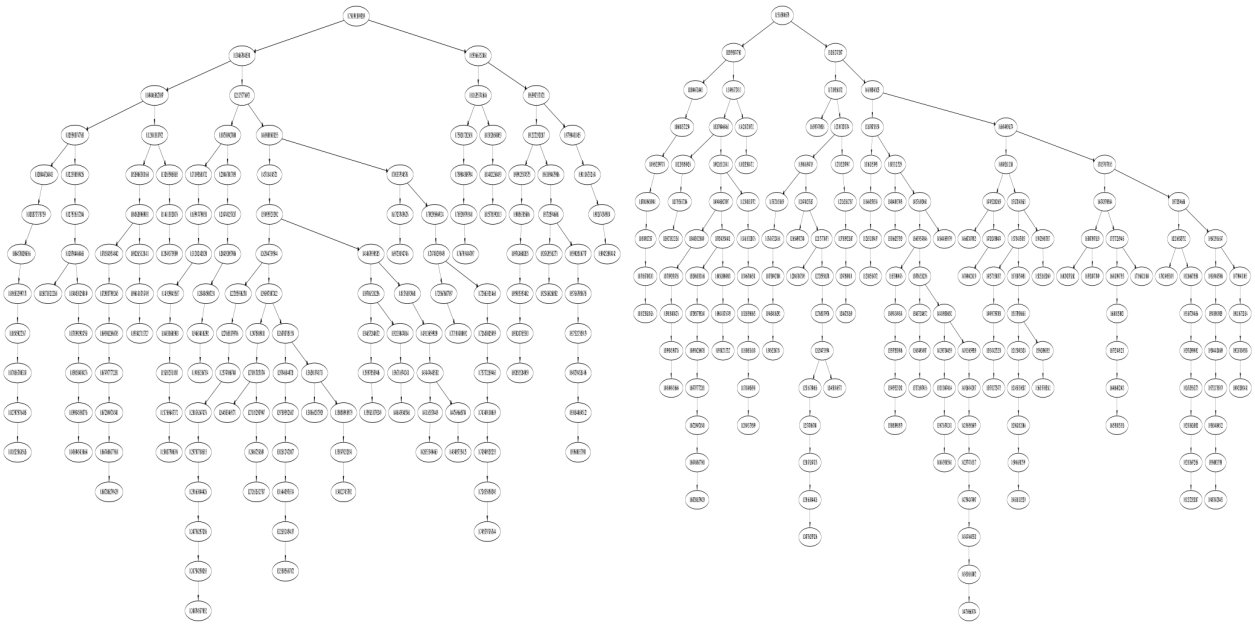


Figure 17: Full Tree Predictions - 2 Sine Waves

In many areas of the plot, there are splits that are introduced that are overfitting to the data, but that is what is expected in the full tree. This tendency to overfit, can be seen even more clearly in the full tree predictions on the Piecewise Constant-dataset (see plot in Appendix A).

An interesting consequence of the structure of the dataset and how the tree is recursively split, is that when the dataset is very symmetric or very sparse, the tree tends to become skewed. One can see that this is the fact because a split in the middle of a symmetric dataset produces a $\hat{y}_{R_1} \approx \hat{y}_{R_2}$. The best split is more likely to happen somewhere around either low or high values of x.

In Figure 18 we can see indications of this, where we in the piecewise linear tree, have a slightly more symmetric tree with both Left Hand and Right Hand splits, whereas we in the sine wave tree more of a tendency to place most of the remaining data in the Right Hand splits.



(a) Piecewise Linear Tree plot

(b) Sine Tree Plot

Figure 18: Tree Plots

However, this result is not overly worrying. In our thesis, we are only really interested in the actual split points themselves. The order in which they are looked at in the tree, actually does not matter much for the current application.

5.1.2 Pruning the Tree

We continue by looking at the method for pruning the tree back to some smaller, less complex tree that is likely to generalise better (and also generate better knot placements).

Using the procedure described in section 3.5, we can easily plot MSE_{Total} over the number of split points. We can also perform this same process for the test dataset, to include this in the same plot. Important to note about the plot, is that the first point in the plot represents one constant value of $\hat{y}$, i.e. no split points. The full set of plots, we see in Appendix A, but we can for example highlight the result in the piecewise/double exponential (see Figure 19. Here we see somewhat of an elbow at around 10, but already at 4 we have a decent change in reduction between k's.

We also see in Figure 19 that the MSE_{Total} for the test data is slightly increasing from its lowest value somewhere just under 20. We are not very surprised about being increasing (after all, the model has overfit to the training data) - we are however noting that the increase is only very minor. We assume that the minor increase is based on the fact that the datasets are very simple with a clear structure.

It is also important to note that the actual predictions that the model makes, are of course based on the available training data. However, these predictions are not necessarily entirely relevant or of interest to us, since we are mainly interested in finding where to place knots. We are instead just interested in the fact that the models makes a split at a certain point. However, also this can of course also overfit.

We can also briefly investigate the case of a constant sine wave, which can be seen in Figure 20. Here, we have a smoother curve, and the ”elbow” is not as clear - or appears significantly later in the curve - in this case, a reasonably clear elbow is seen at around $k = 20$. However, the magnitude of the decrease between

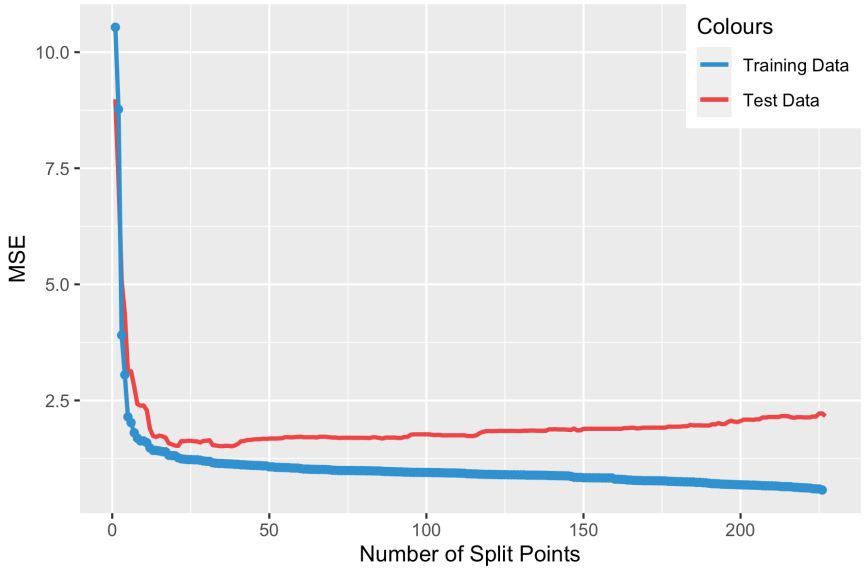


Figure 19: MSE Increase over k - Piecewise Exponential

k's is significantly reduced at $k = 7$. Hence, a selection of either 6 or approximately 20 split points can seem reasonable, but we recommend opting for the lower number in this case - particularly since we already know that the dataset is quite "well behaved".

The selection of number of split points has been performed manually by visual inspection. A table of the selected number of split points for each dataset follows in Table 2.

	Number of Split Points
Piecewise Constant	3
Piecewise Linear	4
Piecewise/Double Exponential	4
Constant Sine	6
2 Sines with different frequencies	7

Table 2: Number of Knots in reduced model

5.1.3 Placing the Knots

Given that we have a suitable number of split points, we now move our attention to the placement of knots. Here, one question could be if the knots should be placed precisely on the split points or exactly in the middle. To visualise the impact of this choice, we can look at two different datasets (see Figure 21. Here we have fitted a Knot Optimization with Recursive Partitioning model and reduced according to Section 5.1.2. We then generate B-Splines on the data with the internal knots from the split points.

For both datasets, we see that when knots are placed at the split points (in plot (a) and (c)) we have a poorer fit of the spline. In the sine wave dataset, this effect is considerably significant. Placing the knots in the midpoints does mean we need to include an additional knot, but given the potential large difference in fit, this is an acceptable tradeoff.

Another important finding relates to the optimality of the knot placement. We can visualise this by looking at a B-Spline fit for the dataset with two sine waves (Figure 22). In this example, we have knot placements

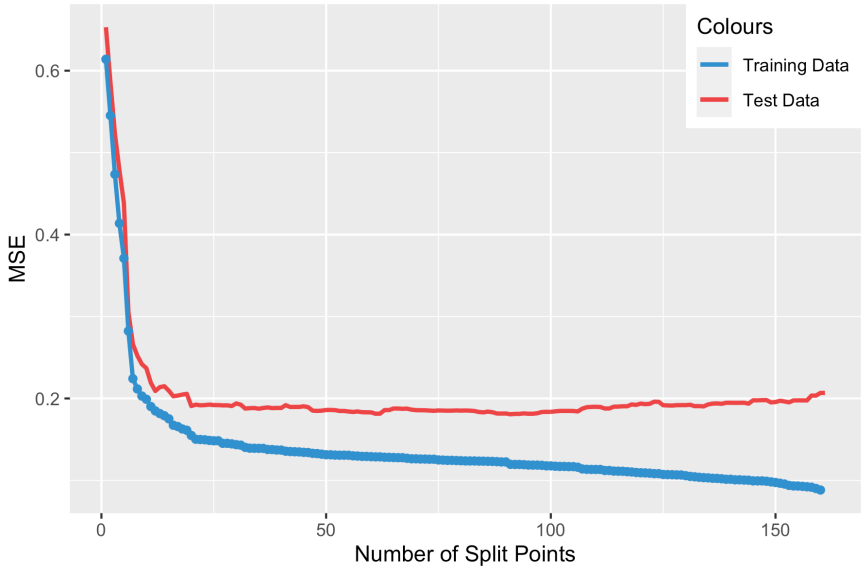


Figure 20: MSE Decrease over k - Constant Sine Wave

identified at each minimum and maximum of y. Seeing that this was very suitable for a constant sine wave, we would expect it would suit also the two sine waves dataset. However, although the knots are placed at these points, we see that the fit towards the upper end of the function is not particularly good.

Instead, we need to increase the number of knots. However, since we through our proposed method can not control which specific knots are added in first, we need to add enough knots to make sure additional ones are included in the relevant range. In Figure 22b, we can for example see the fit with 13 knots instead of 8, which looks considerably better.

This is however, not a significant problem for the method. Our purpose is to find knots for further Functional Data Analysis, and as such we would actually want to be more generous with the number of knots anyway.

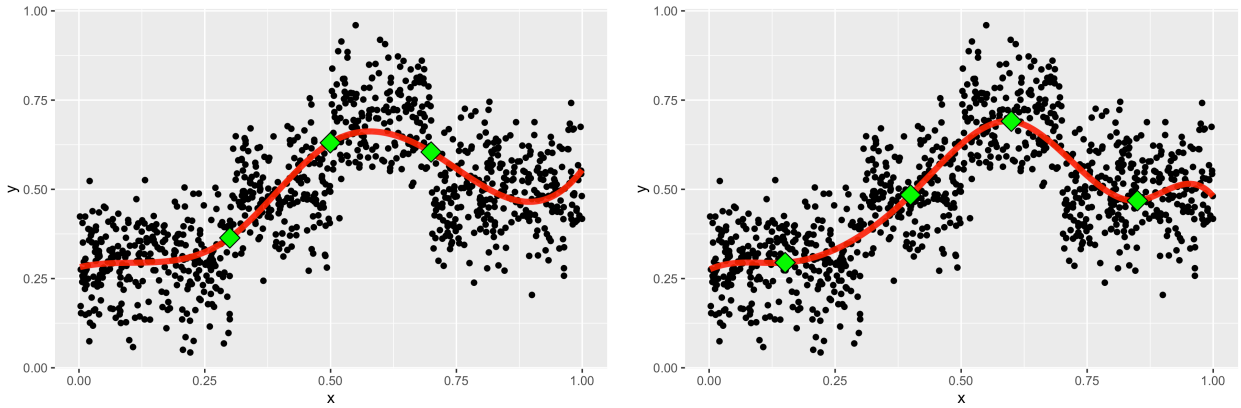
5.1.4 Comparing results with Genetic Algorithm

We have also run a genetic algorithm using the "2 Sine Waves" dataset to understand the behavior of this algorithm, in relation to the KORP method. The Genetic Algorithm variant run here, is the Varying Dimension variant, since we are here trying to find both optimal knot placements and optimal number of knots simultaneously.

As can be seen in Figure 23a, it is obvious that the methods give very similar results (with a Pearson correlation of approximately 0.99 for the actual knot placements).

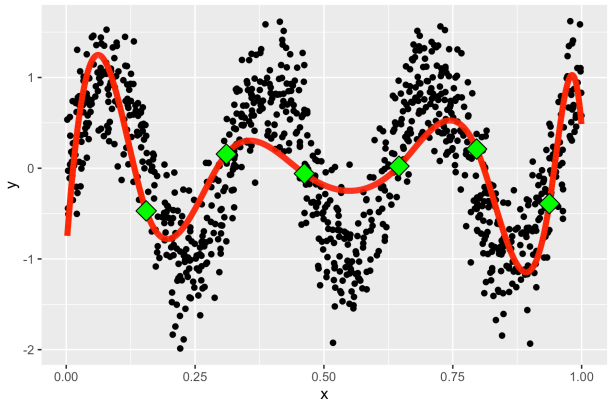
In Figure 23, we have regressed y on a B-spline fit for x, once per knot placement method. In the plot, we look at the residuals from each observation, and we can immediately again note a very strong correlation ($R^2 \approx 0.92$).

Given these results, and the fact that the Varying Dimension Genetic Algorithm takes considerable more time to run, we see no clear benefits with using this instead of the KORP method. However, seeing that our results are almost equal between the two methods, we can only conclude that it seems our proposed method is actually providing reasonable results.

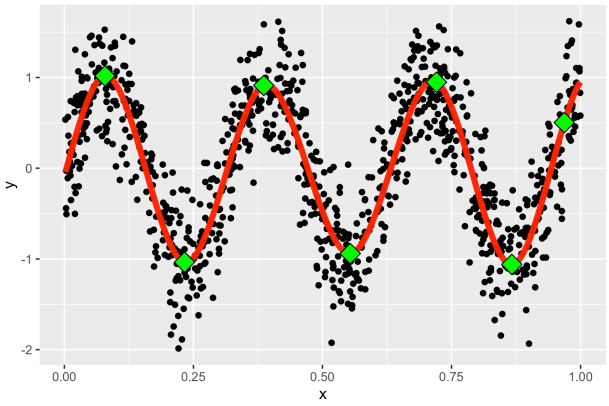


(a) Piecewise Constant - Knots at split points

(b) Piecewise Constant - Knots between split points



(c) Sine Wave - Knots at split points

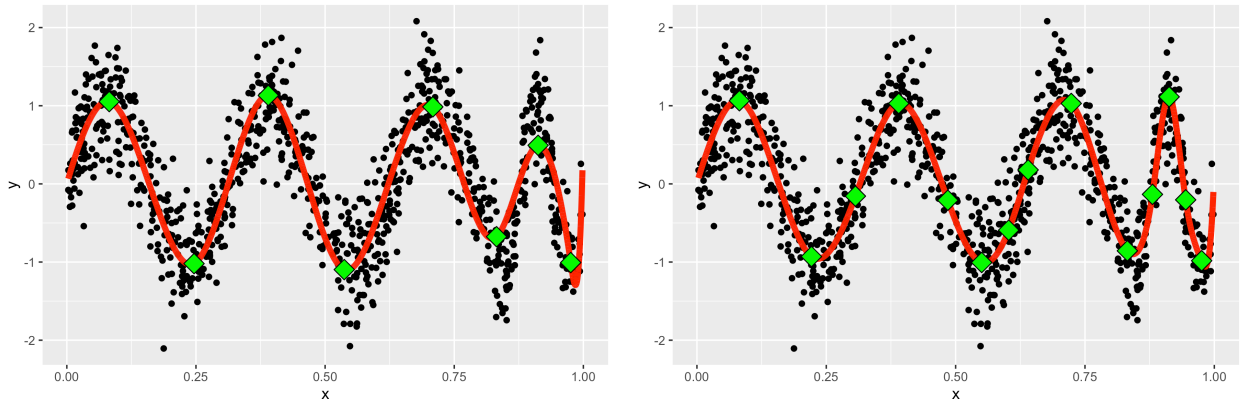


(d) Sine Wave - Knots between split points

Figure 21: Different Knot Placements

5.2 MNIST Handwriting Data

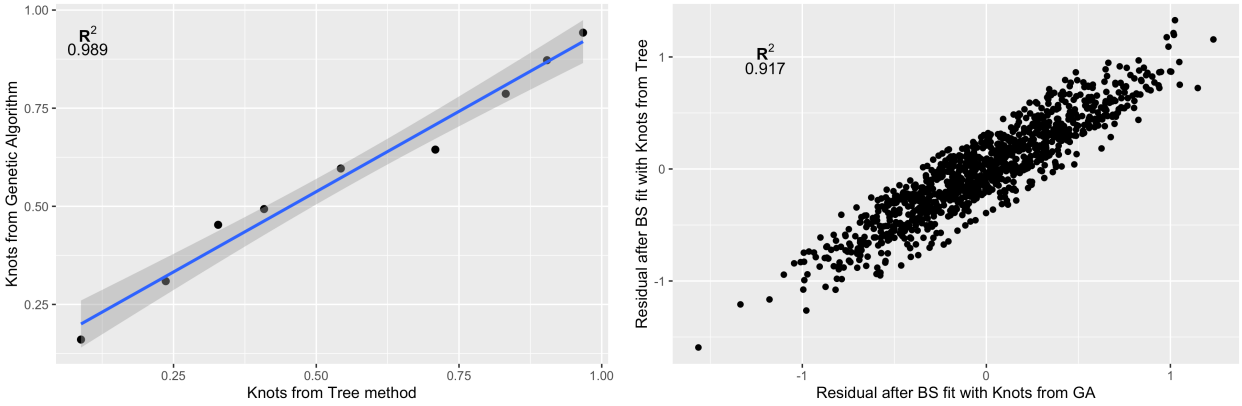
As indicated in section 4.2, we will first of all perform mean curve subtraction, which is used for fitting the Tree. However, we now need to visualise the tree slightly different to before, since we have many observations that are entire curves. Plotting every curve in a single plot makes it very difficult to interpret the contents, so in Figure 24 we have taken a random sample of 50 observations and plot these, and the predictions from the full tree. As the reader can see, it is already with 50 observations quite hard to distinguish each curve from the others. Also important to note is that we are of course only sampling 50 of the thousands of observations available. Hence, we can probably not make too bold claims about the visual fit. One thing that can be noted however, is that generally, the white "bands" are usually not populated with any split points, which at least gives some confidence that the method has produced relevant results. For the sections the range of p where we do have lots of activity, we can note that it is quite hard to distinguish the splits. This is dues to the fact that the full tree gives unique predictions for almost every p where something is happening in the function.



(a) 8 Knots

(b) 13 Knots

Figure 22: Excessive number of knots needed



(a) Correlation, Knots from Tree and Genetic Algorithm

(b) Correlation, Residuals from BS Fits

Figure 23: Knots from GA and Tree methods

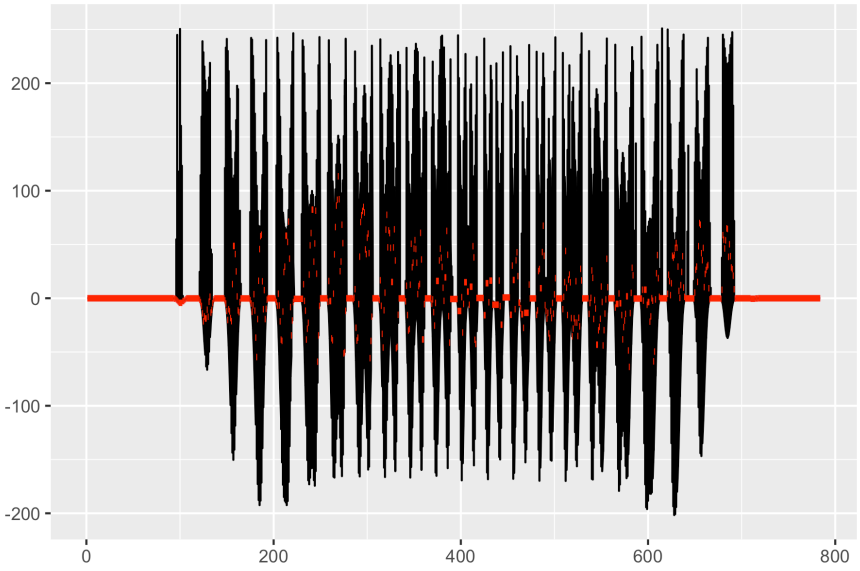


Figure 24: 50 observations, Difference from Mean Curve and Full Tree

We can also have a brief look at the MSE plot in Figure 25. We see that it is tapering off significantly slower than in the simulated datasets, probably because of the complexity of the data. Here, we may therefore argue for a need to keep many split points in order to accurately represent the curves, perhaps as many as 200 or even more.

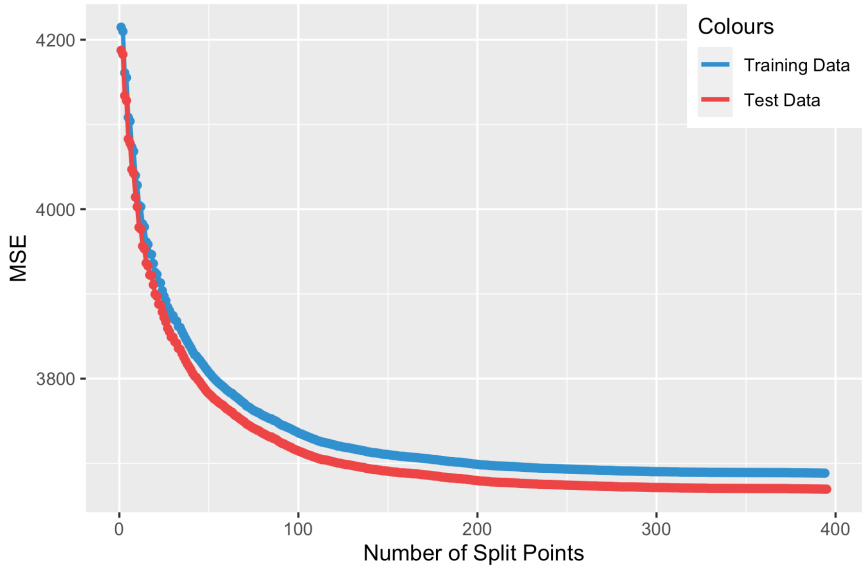


Figure 25: Reduction of MSE over number of knots

In order to get some insight into the appropriateness of the knot placements, we will visualise a series of observations including B-spline fits for each. The B-spline fits will be using the knots identified through the tree method in the previous step, but also a uniform placement of knots, where the number of uniform knots is set to be equal to the number of tree knots.

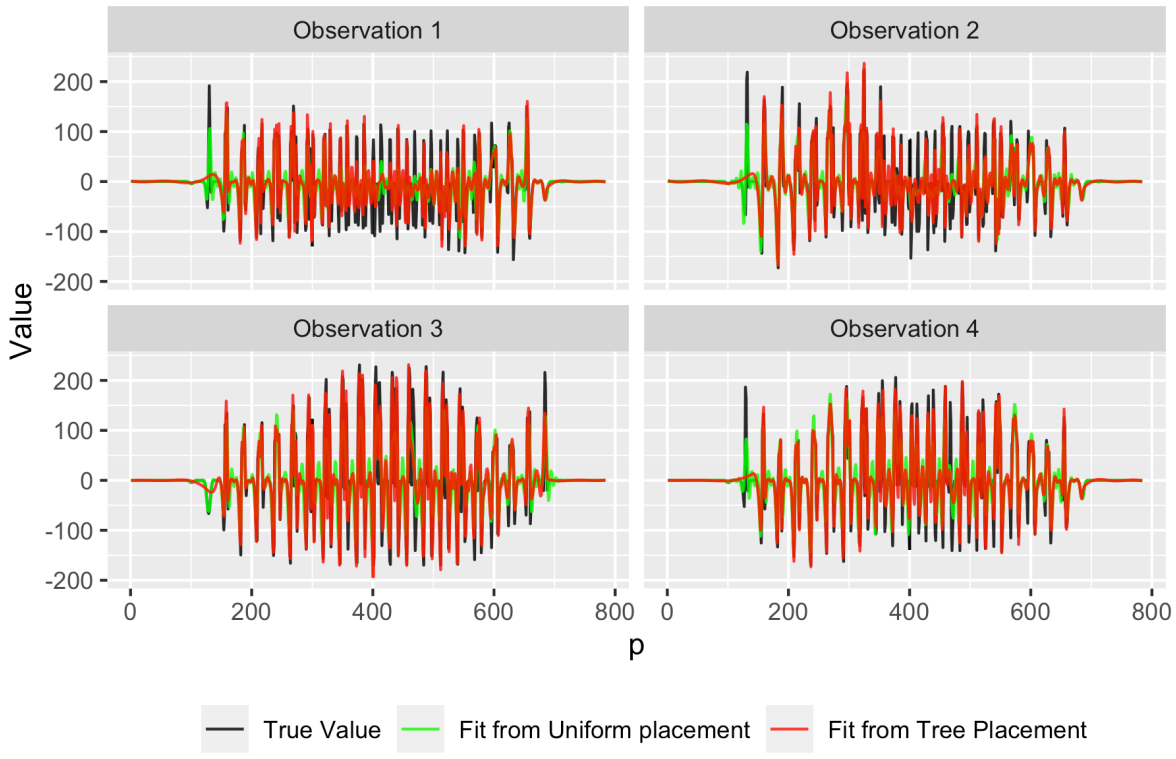


Figure 26: 4 Observations with B-spline fits

In Figure 26, we see 4 observations with the corresponding Spline fits. Although not very clear from these plots, partly because only 4 out of 5923 observations are plotted, and partly due to the very localized variation in the data, we still consider the fit to visually be better for our knot placement method. We can also see this far more clearly in Figure 27, where we present only a part of the range of p in order to more clearly see the differences.

Figure 27: Part of Observations with B-spline fit

Finally, we will also have the opportunity to return to this question shortly, with a more quantitative method to assess the performance.

5.2.1 Comparing with Genetic Algorithms

Using the binary genetic algorithm variant, we can generate a set of knot point proposals, and compare with the knots proposed by the tree method. As can be seen in Figure 28, we have significant curvature. This indicates that the genetic algorithm has placed the knots over a slightly larger area (i.e. more towards the beginning and end of the range of x), whereas the Tree method focuses on placement in the center of the range, where more things are happening.

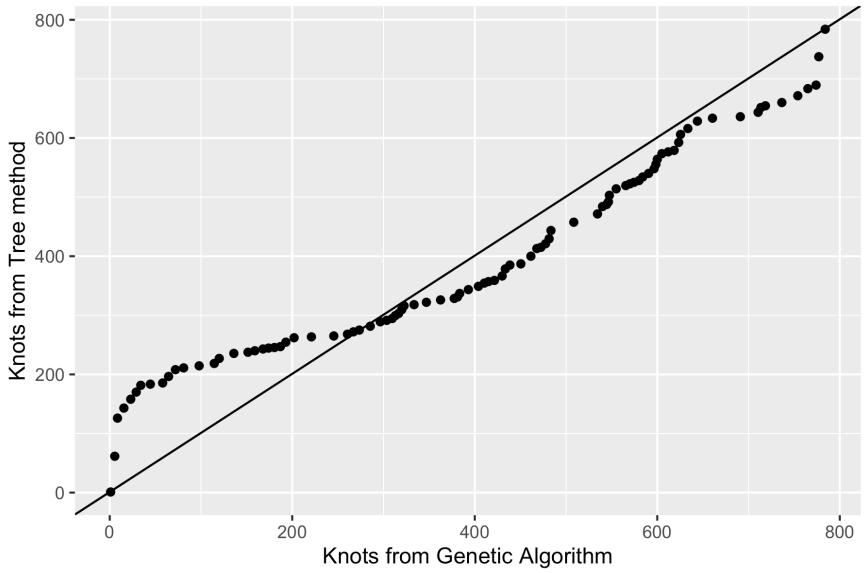


Figure 28: Knots from Tree and GA

We can also also quantify the quality with MSE from B-Spline fit. We can for each observation calculate MSE_{tree}, $MSE_{uniform}$ and MSE_{GA} and compare the differences, for example in a boxplot, as in Figure 29.

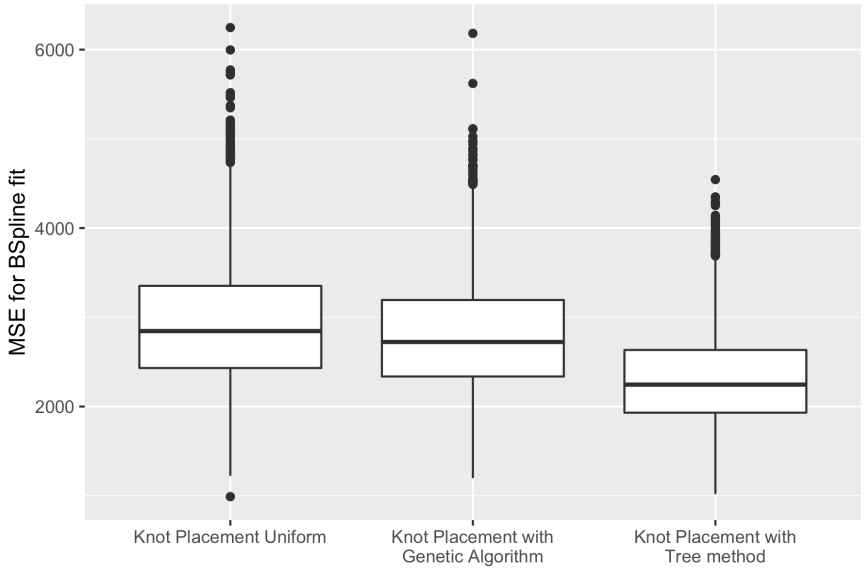


Figure 29: Boxplot of MSE of BSpline Fits

From the boxplot it looks like there is very little difference, but since we have quite large samples even a small difference could be significant. A formal ANOVA test is formulated the following way. We test the null hypothesis

$$H_0 = \mu_{Tree} = \mu_{Unif} = \mu_{GA} \tag{39}$$

versus the alternative hypothesis

$$H_1 = \textit{Any of the means are not equal.} \tag{40}$$

Result of the test is presented in an ANOVA table as follows:

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Method	2	1248830278.69	624415139.35	1696.69	0.0000
Residuals	17766	6538253309.84	368020.56		

This shows that there is a significant difference (at well below $p < 0.05$) in means for at least one of the methods, which means we can reject the null hypothesis. By also performing a Tukey HSD Post-Hoc test, we can see that the Tree method seems to have the best performance overall, and significant results for each comparison (all well below $p < 0.05$).

	diff	lwr	upr	p adj
Genetic Algorithm - Uniform	-134.45	-160.58	-358.32	0.00
Tree method - Uniform	-617.41	-643.54	-591.29	0.00
Tree method - Genetic Algorithm	-482.97	-509.09	-456.84	0.00

Table 3: Tukey's HSD Post-Hoc Test for residuals from three methods

6 Conclusions

In this thesis we have continued previous research on tree based methods for knot placement in functional data, by fully following the regression tree paradigm. We have evaluated the method on both simulated datasets and on the MNIST handwriting dataset and compared both with uniform placement of knots and a genetic algorithm for identifying optimal placement of knots.

Our conclusion from the study of the proposed method, is that the method works very well, both for simple datasets and for functional data. It generally performs better than both Uniform placement and Genetic Algorithms. The former is along what one could reasonably expect, but the latter is at least a little more surprising. The difference between the two is certainly not massive, but at this point large enough, at least for functional data, to not motivate its usage directly. However, seeing that the research on Genetic Algorithms is fairly extensive, there are likely many improvements that can be made to our simple approach in order to achieve better results. It is most likely the case that with only some modifications, the Genetic Algorithms can be on par with the proposed KORP method also for functional data.

However, the Genetic Algorithm has a big advantage. In this study we have had a principle of not introducing any prior knowledge about the basis functions or splines in the knot proposal. If this principle is relaxed, it is certainly something that could be easily incorporated into the Genetic Algorithm. That is, instead of minimising some form of penalized MSE, we could minimise for example the sum of MSE of a B-Spline fit for every functional observation. This could potentially be more accurate than our currently proposed method, since recursively building the tree would not be able to do this, at least for B-Splines or other non-orthogonal splines.

This is also similar to some previously identified methods on the knot placement subject, for example using RJMCMC (Dimatteo, Genovese, and Kass, 2001). Therefore, a thorough evaluation and comparison between using RJMCMC and the KORP method would certainly be relevant for further research.

References

[Sch46] I. J. Schoenberg. "CONTRIBUTIONS TO THE PROBLEM OF APPROXIMATION OF EQUIDISTANT DATA BY ANALYTIC FUNCTIONS: PART A.—ON THE PROBLEM OF SMOOTHING OR GRADUATION. A FIRST CLASS OF ANALYTIC APPROXIMATION FORMULAE". In: *Quarterly of Applied Mathematics* 4.1 (1946), pp. 45–99. ISSN: 0033569X, 15524485. URL: http://www.jstor.org/stable/43633538.

[Boo78] Carl d. Boor. *A Practical Guide to Splines*. New York: Springer Verlag, 1978.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[Gre95] Peter J. Green. "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination". In: *Biometrika* 82.4 (1995), pp. 711–732. ISSN: 00063444. URL: http://www.jstor.org/stable/2337340.

[KE95] J. Kennedy and R. Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95 - International Conference on Neural Networks*. Vol. 4. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[LeC+98] Yann LeCun et al. "Gradient-based learning applied to document recognition". English (US). In: *Proceedings of the Institute of Radio Engineers* 86.11 (1998), pp. 2278–2323. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[DGK01] Ilaria Dimatteo, Christopher Genovese, and Robert Kass. "Bayesian curve-fitting with free-knot splines". In: *Biometrika* 88 (Dec. 2001), pp. 1055–1071. DOI: 10.1093/biomet/88.4.1055.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[BM05] Alexandru Horia Brie and P. Morignot. "Genetic Planning Using Variable Length Chromosomes". In: *ICAPS*. 2005.

[Jam+13] Gareth James et al. *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. URL: https://faculty.marshall.usc.edu/gareth-james/ISL/.

[PB15] Sunil Nilkanth Pawar and Rajankumar Sadashivrao Bichkar. "Genetic algorithm with variable length chromosomes for network intrusion detection". In: *International Journal of Automation and Computing* 12.3 (2015), pp. 337–342. DOI: 10.1007/s11633-014-0870-x. URL: https://doi.org/10.1007/s11633-014-0870-x.

[KR17] P. Kokoszka and M. Reimherr. *Introduction to Functional Data Analysis*. Chapman & Hall / CRC numerical analysis and scientific computing. CRC Press, 2017. ISBN: 9781498746342. URL: https://books.google.se/books?id=HIxIvgAACAAJ.

[BG19] Brad Boehmke and Brandon Greenwell. *Hands-On Machine Learning with R*. Nov. 2019. ISBN: 9780367816377. DOI: 10.1201/9780367816377.

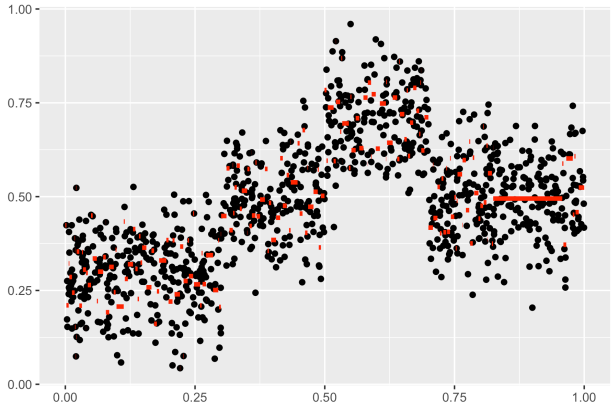
[BNP21] Rani Basna, Hiba Nassar, and Krzysztof Podgorski. "Machine Learning Assisted Orthonormal Basis Selection for Functional Data Analysis". In: (Mar. 2021).

[NP21] Hiba Nassar and Krzysztof Podgórski. "Empirically Driven Orthonormal Bases for Functional Data Analysis". In: *Numerical Mathematics and Advanced Applications ENUMATH 2019*. Ed. by Fred J. Vermolen and Cornelis Vuik. Cham: Springer International Publishing, 2021, pp. 773–783. ISBN: 978-3-030-55874-1.

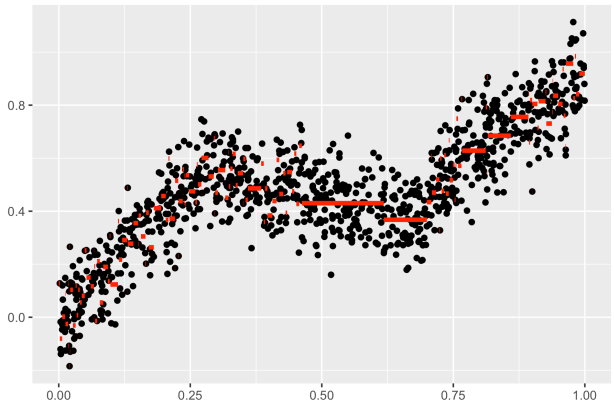
Appendices

A Complete Results from Evaluation

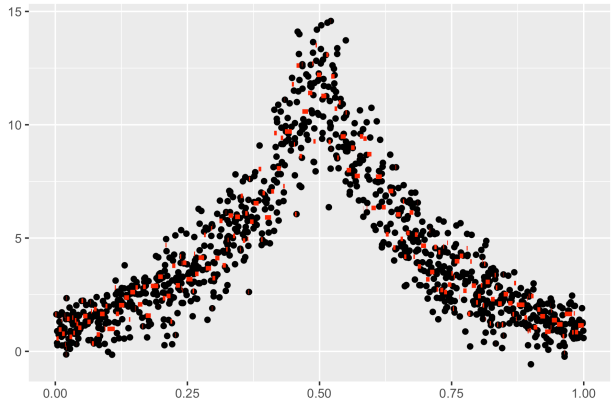
A.1 All Full Trees



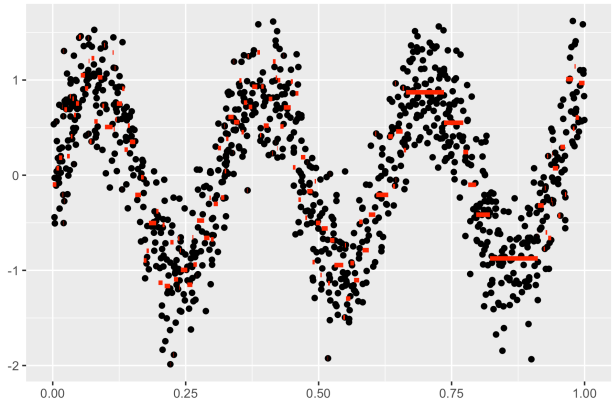
(a) Piecewise Constant



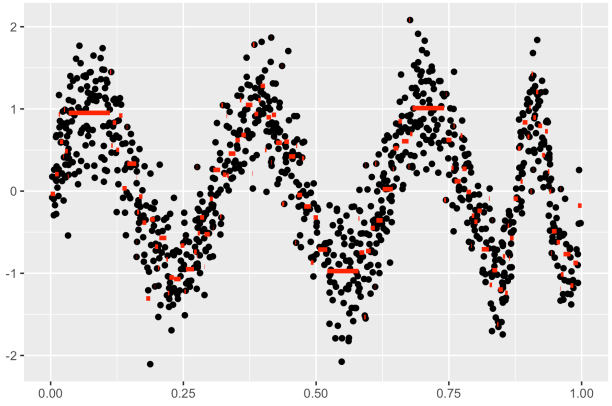
(b) Piecewise Linear



(c) Piecewise/Double Exponential

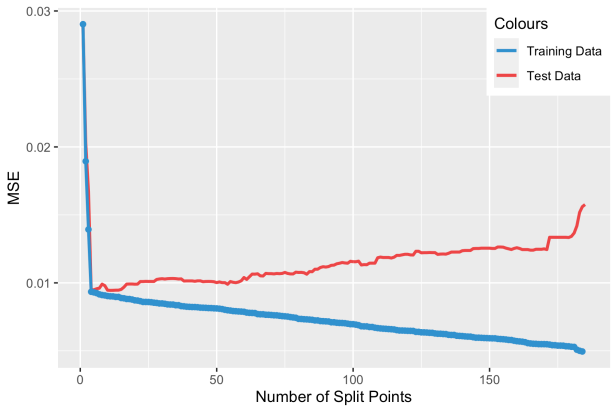


(d) Constant Sine

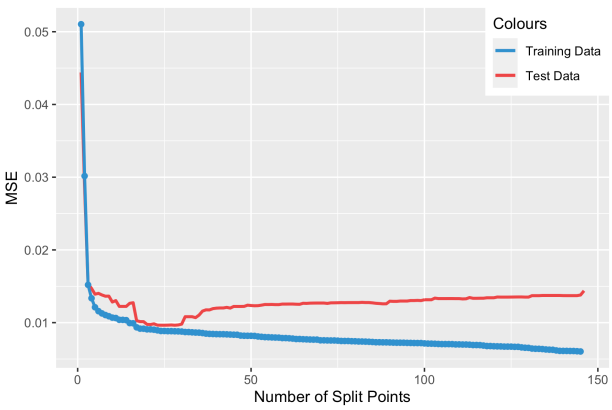


(e) 2 Sines with different Frequencies

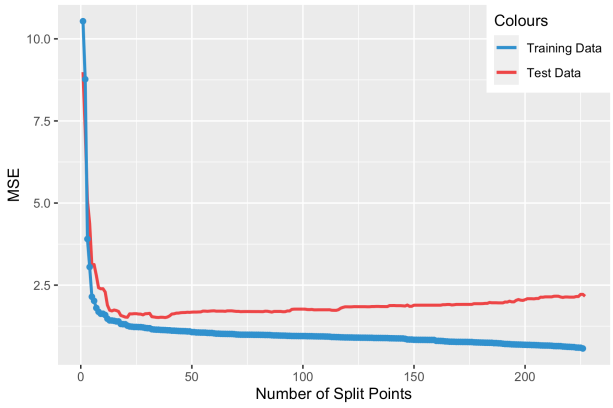
A.2 All MSE plots



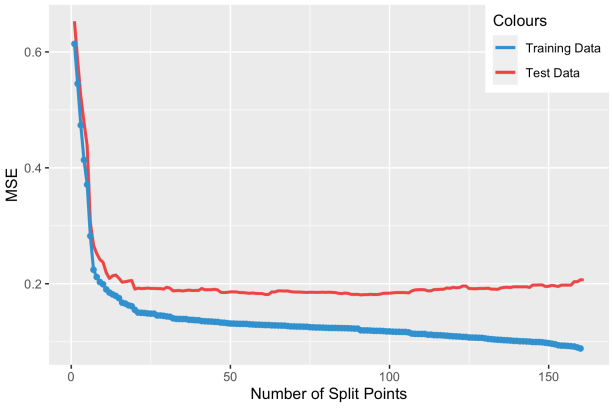
(a) Piecewise Constant



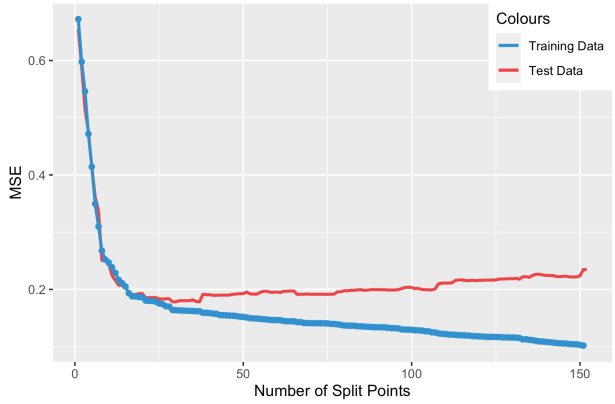
(b) Piecewise Linear



(c) Piecewise/Double Exponential

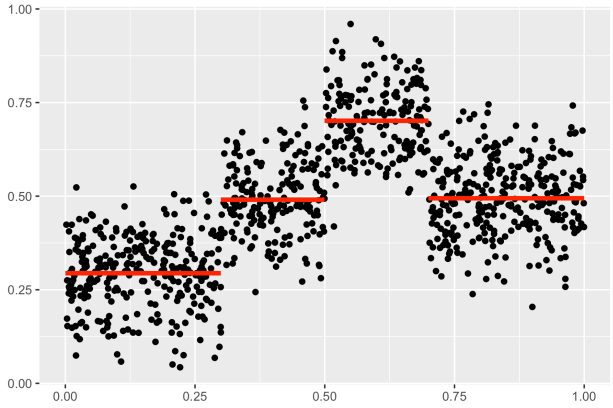


(d) Constant Sine

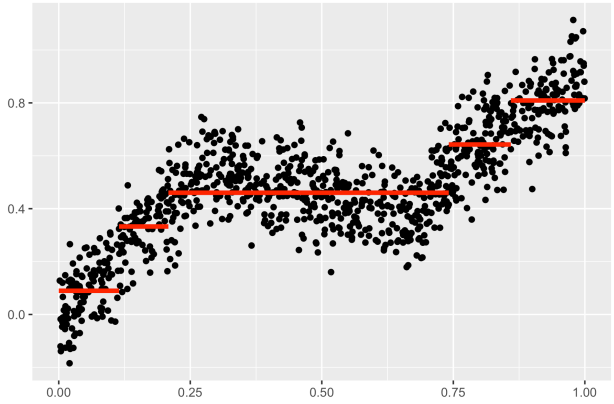


(e) 2 Sines with different Frequencies

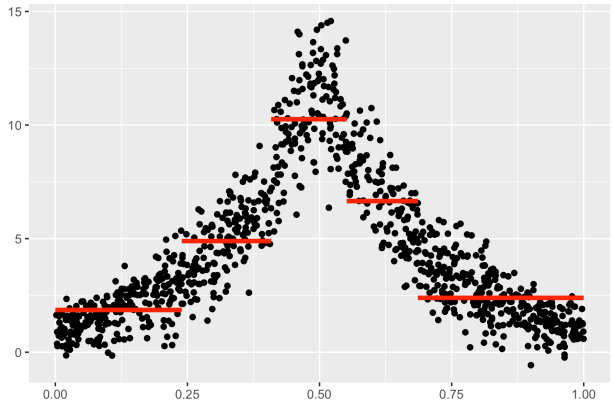
A.3 All Reduced Trees



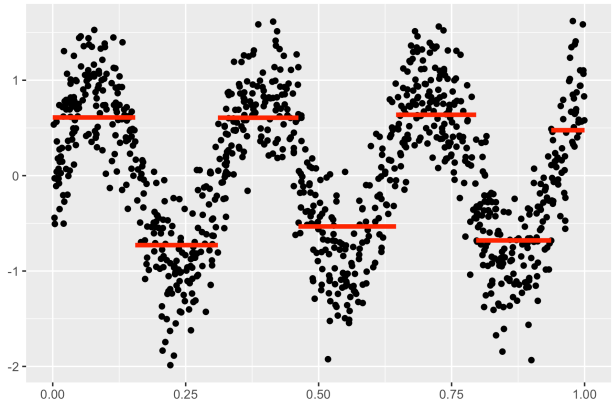
(a) Piecewise Constant



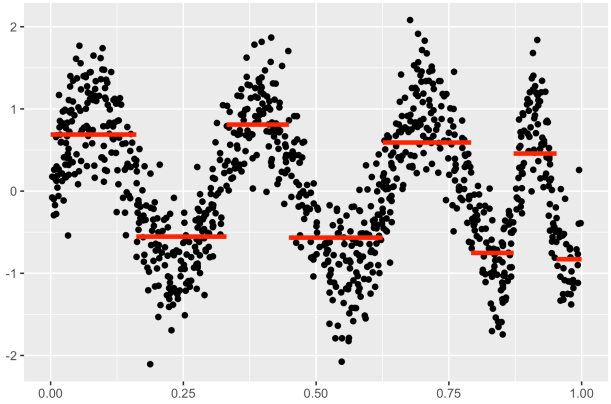
(b) Piecewise Linear



(c) Piecewise/Double Exponential

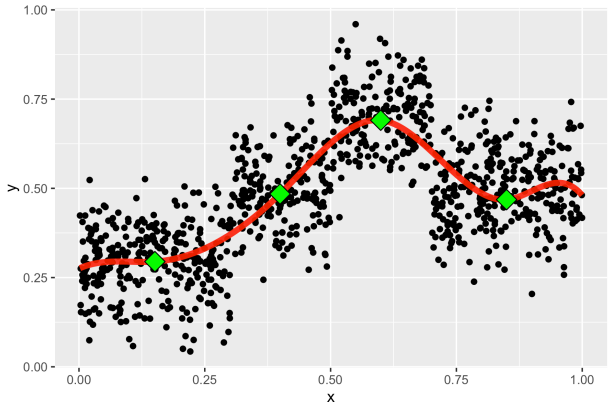


(d) Constant Sine

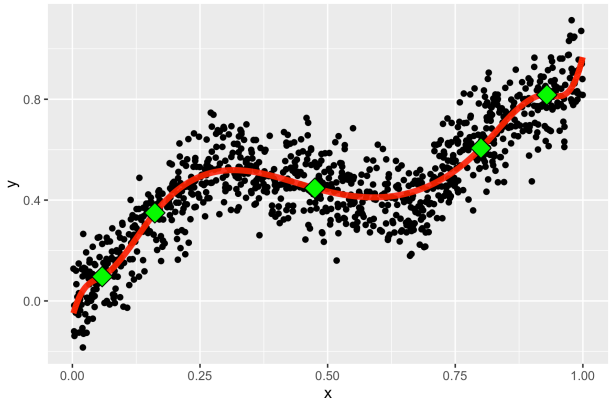


(e) 2 Sines with different Frequencies

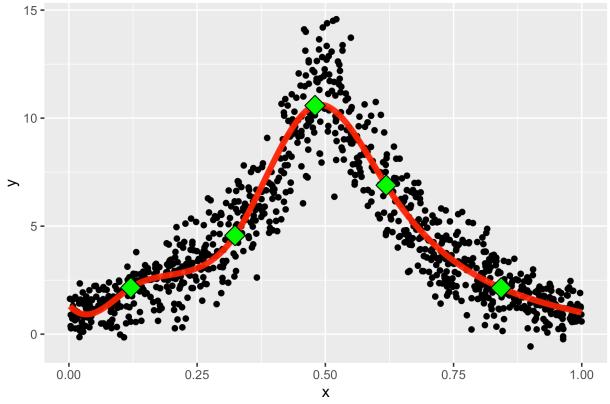
A.4 All B-Spline plots



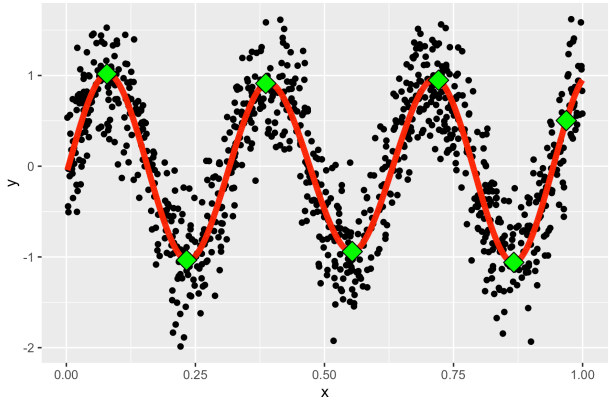
(a) Piecewise Constant



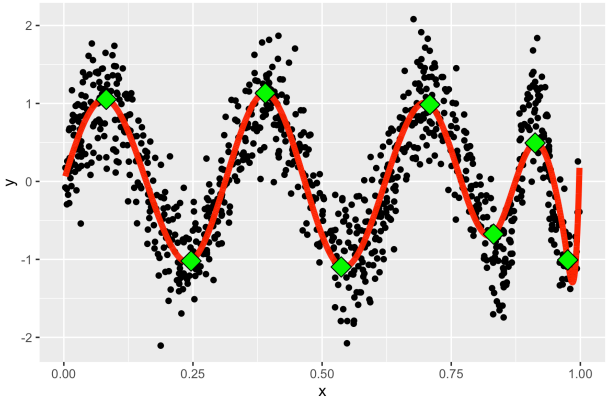
(b) Piecewise Linear



(c) Piecewise/Double Exponential



(d) Constant Sine



(e) 2 Sines with different Frequencies

B Particle Swarm Optimization

Particle Swarm Optimization was originally a method developed to simulate social behaviour, but can also be used for optimization purposes. Here, the process is outlines as follows:

Let $f(x)$ be a function to minimize

- Initiate a swarm of n particles with a random set of x_i, representing the particles' position in the search space.
- Initiate n random velocities for particles with a random set of v_i, representing the particles' velocity in space.
- Evaluate $f(x)$ for each individual genome
- Do until convergence:
  - Update state of global best and each particle's personal best
  - Update the position and calculate new velocities

A first random initialization of particles and velocities could look as in Figure 34.

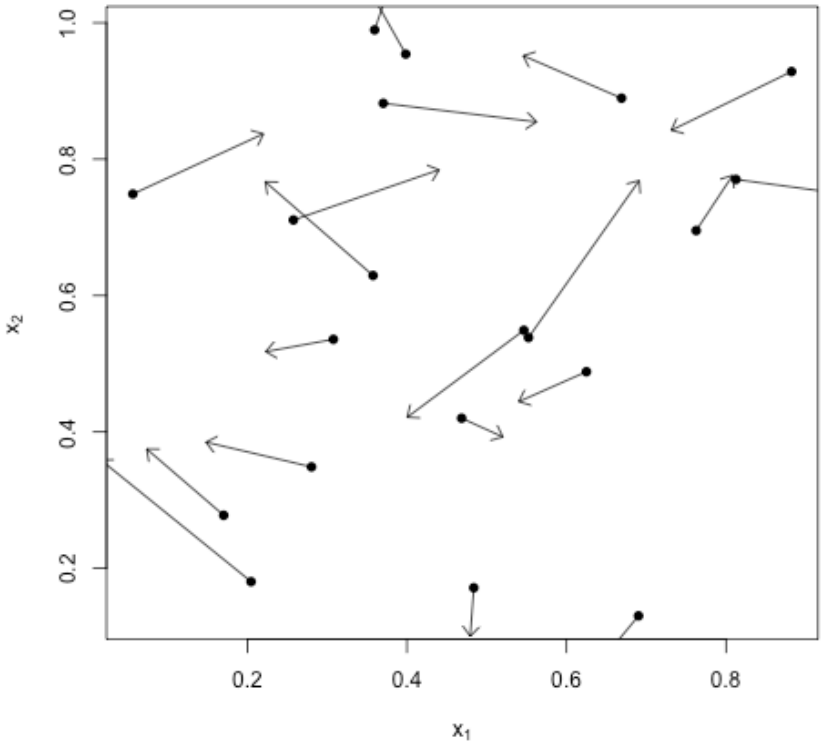


Figure 34: Particle Swarm Optimization - After Init

In the step above where positions and velocities are updated, the following calculations are performed:

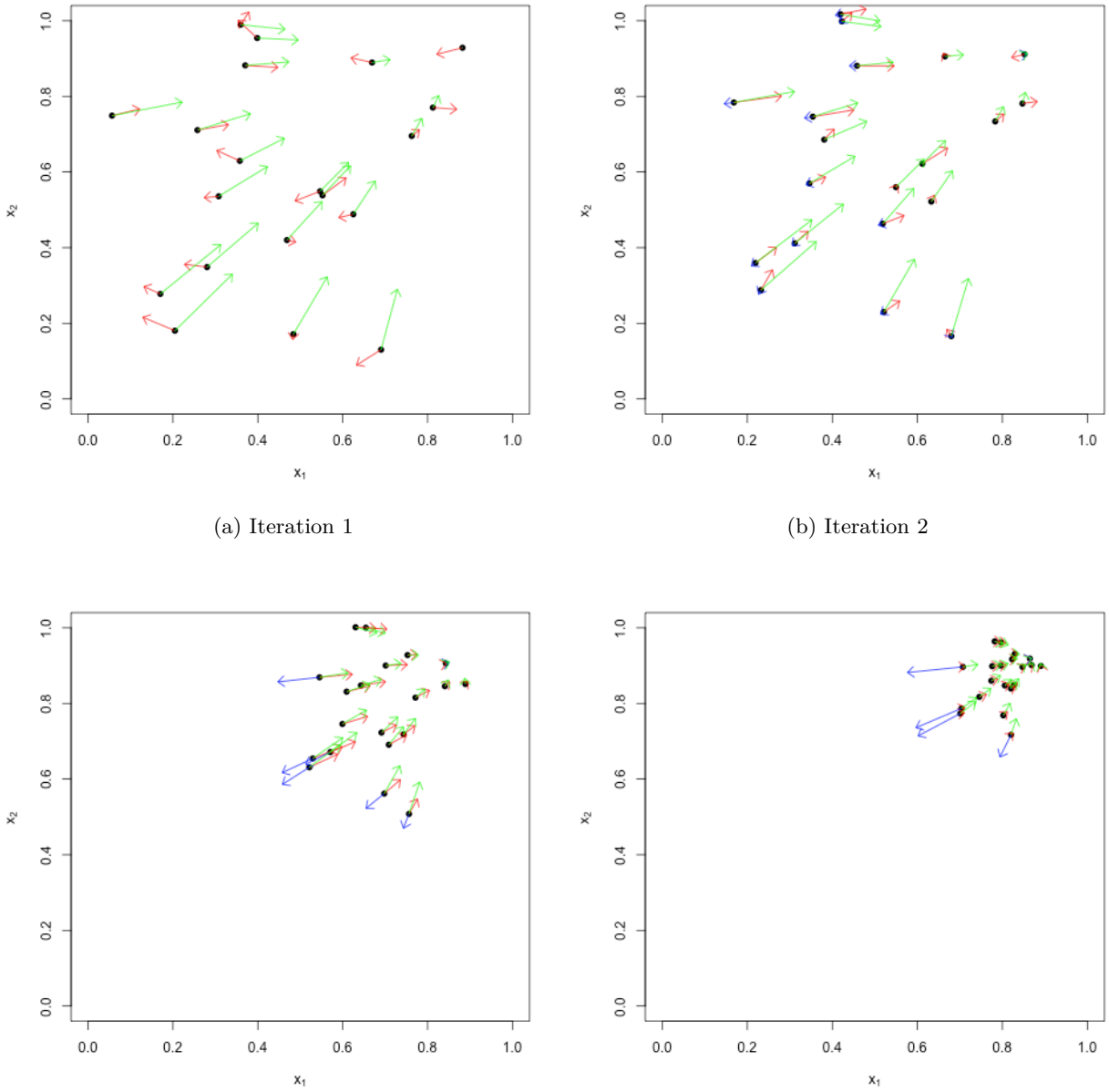
$$V_{i_{t+1}} = \alpha V_{i_t} + \beta(X_i - X_{i_{p.best}}) + \gamma(X_i - X_{g.best})$$

where $X_{i_{p.best}}$ and $X_{g.best}$ represents the personal best for the particle itself, and the global best particle respectively. When new velocities are calculated, new positions are calculated as

$$X_{i_{t+1}} = \psi V_{i_{t+1}} X_{i_t}$$

Taking a few snapshots from the running of an optimization, we see in Figure 35 the gradual convergence of particles to the global minimum. Each particle (dot) will move according to its total trajectory, which is a combination of global best (green arrow), personal best (blue arrow) and trajectory of last iteration (red arrow).

As can be seen in Figure 35, the general direction of the best particle is identified early, and in the first iteration, some of the initial velocity still remains. By iteration 2 the velocity has almost completely changed. In Iteration 5 and 10, we see that the velocities of some particles still are impacted by a previous personal best.



(a) Iteration 1

(b) Iteration 2

(c) Iteration 5

(d) Iteration 10

Figure 35: Particle Swarm Optimization Illustration