# Mixtures of Partially Linear Models with Monotone Shape Constraints

---

Master Thesis in Biostatistics (STA495)

by

Daniel Leibovitz

19-764-851

supervised by

Dr. Torsten Hothorn (UZH, Zürich)

Dr. Matthias Löffler (ETH, Zürich)

Zurich, April 2021

# Mixtures of Partially Linear Models with Monotone Shape Constraints

Daniel Leibovitz

Version May 3, 2021

# Contents

# Abstract

Mixtures of non-parametric monotone regressions are applicable to clustering problems where there is prior knowledge about appropriate shape constraints within the resulting model. The current standard for estimating such models involves fitting a series of non-parametric regression functions without shape constraints using an EM algorithm, as described by Zhang and Zheng (2018), followed by a monotonic estimate given the resulting latent variable classifications. In this paper, we propose to remove redundancy by incorporating the non-parametric monotone regression function estimate into the M-step of the EM algorithm. Moreover, we generalize the model to include any number of monotone or linear effects. We demonstrate the effectiveness of the algorithm when applied to both simulated data and real-world data on global life expectancy and GDP from the World Bank.

# Acknowledgments

I would like to thank: Drs. Holger Dette, Runze Li, and Mian Huang for providing the necessary code to run the Zhang et al. algorithm. Dr. Matthias Löffler for initially suggesting the topic of this manuscript, and for pushing me in the right direction on all the occasions when I seemed to have reached a dead end. The faculty of the Master's in Biostatistics at the UZH for two intense, creative and academically stimulating years, and for shepherding me on an academic journey from a state of statistical ignorance to a state of somewhat less statistical ignorance. Dr. Torsten Hothorn in particular for agreeing to sponsor this master thesis and thereby demonstrating significant trust that I would return with something useful. My family for supporting me through an unexpected number of video calls and a global pandemic. My friends in Zürich for making six months writing a thesis in home-office not only something I don't regret, but something I would do again.

Daniel Leibovitz
April 2021

# Chapter 1

# Introduction

A mixture of partially linear regressions with monotone shape constraints takes the following form:

$$Y = \begin{cases} \sum_{h=1}^{p} g_{h1}(Z_h) \ + \ \sum_{j=1}^{q} \beta_{j1} X_j \ + \ \epsilon_1, \text{ with probability } \pi_1; \\ \vdots \\ \sum_{h=1}^{p} g_{hk}(Z_h) \ + \ \sum_{j=1}^{q} \beta_{jk} X_j \ + \ \epsilon_k, \text{ with probability } \pi_k; \end{cases} \tag{1.1}$$

where the model has $K$ components, $\boldsymbol{X} \in \mathbb{R}^q$, $\boldsymbol{Z} \in \mathbb{R}^p$, $\beta \in \mathbb{R}^q$, and each function $g_{hk}()$ is assumed to be monotone. The error $\epsilon$ is assumed to be Gaussian distributed with mean zero and to be independent of the covariates $(\boldsymbol{X}, \boldsymbol{Z})$. The prior probabilities $\pi_k$ satisfy the conditions $\pi_k \in (0, 1)$ and $\sum_1^k \pi_k = 1$.

Such a model has broad applications for clustering of data in domains where monotone relationships are known *a priori*. These domains include, for example, epidemiology, where risk-exposure relationships may be modeled monotonically (Morton-Jones et al. (2000), Cai and Dunson (2007)); finance, where option pricing functions may be restricted to both monotonicity and and/or or convexity (Ait-Sahalia and Duarte (2003)); and biomedical research, where biochemical kinetics may be monotone over time (Oussalah et al. (2020)).

A similar type of model has previously been described by Zhang and Zheng (2018), and takes the following form for a $k$-component mixture:

$$Y = \begin{cases} g_1(Z) \ + \ \epsilon_1, \text{ with probability } \pi_1(Z); \\ \vdots \\ g_k(Z) \ + \ \epsilon_k, \text{ with probability } \pi_k(Z); \end{cases} \tag{1.2}$$

where $\boldsymbol{Z} \in \mathbb{R}^1$, and each function $g_k()$ is assumed monotone. The variables $\epsilon_k$ and $\pi_k$ are not constant, as in model [1.1], but are modelled as nonparametric functions $\epsilon_k(Z)$ and $\pi_k(Z)$. They nonetheless satisfy the same conditions as in model [1.1], namely, $\pi_k(Z) \in (0, 1)$ and $\sum_1^k \pi_k(Z) =$

1 for any $Z$, and $\epsilon_k(Z)$ is Gaussian distributed with $E(\epsilon_k|Z) = 0$ and $Var(\epsilon_k|Z) = \sigma_k(Z)$.

The model and estimator of Zhang et al. has four drawbacks:

1. The model proposed by Zhang et al. cannot be generalized to a semiparametric approach, i.e., one cannot include linear effects in the mixture components. Being able to include linear effects in the mixture components is conducive to two distinct purposes:

   (a) First, it can be used when the data to be clustered has multiple independent variables that are not of primary interest, but which the user would still like to include in the model. As Zhang et al. point out, including such varibles with nonparametric effects can explode the complexity of the algorithm beyond usability (see Section [4.0.5]). Including such variables as linear effects is a alternative that keeps the complexity of the algorithm tractable.

   (b) Second, users who are mainly concerned with linear effects of components within a mixture model can flexibly control for one or more nuisance variables that are suspected of being monotone in effect.

2. The Zhang et al. estimator fits mixture components in two, sequential steps, first fitting an unconstrained function and then applying monotone constraints once the components membership has been calculated. This approach introduces a potential bias when components are not clearly identifiable.

3. The Zhang et al. estimator requires a tuning parameter for the fitting of each unconstrained mixture component function, which is difficult in practice and adds computational complexity.

We propose model [1.1] and a new estimator as an alternative, more generalized form of the approach suggested by Zhang et al. that avoids the drawbacks mentioned above. Specifically, our model accepts any number of non-parametric monotone or linear effects within each mixture component; our estimator fits the monotone functions within each component in a single step; and our estimator does not require the calibration of any tuning parameters.

The remainder of this paper is organized as follows. In Section 2, we discuss previous research in the domains of mixture models, partial linear models, and isotonic regression. In Section 3, we discuss the components of the proposed model. In Section 4, we describe the theoretical structure of the proposed model (model [1.1]) as well as the estimation algorithm, followed by the model's asymptotic properties and empirical complexity. In Section 5, we apply the proposed model to simulated data as well as World Bank data on global life expectancy through the end of the 20th century. In the final section, we discuss implications and future work.

# Chapter 2

# Previous Work

The model proposed in this article draws from several branches of statistical research. In this section, we briefly discuss the history and current state of said research.

### 2.0.1 Regression with shape constraints

Parametric regression models are constrained in their shape by construction. They can be further constrained, often trivially, by estimating their shape within a limited parameter space. A univariate linear model $E(Y) = X\beta$, for example, can be constrained to be non-decreasing by estimating $\hat{\beta}$ to be non-negative, i.e., $\hat{\beta} \in [0, +\infty)$. When estimating nonparametric regressions, this triviality is lost and one must reconsider how to estimate constrained functions. The resulting estimators are often applications of classical techniques such as maximum likelihood, and are often free of tuning parameters, making them attractive alternatives to typical, unconstrained nonparametric estimators (Guntuboyina and Sen (2018)).

Several types of constraint have been considered over the years. Frisen described unimodal regression, i.e., the case where for $E(Y) = f(X)$, $f()$ has a unique local maximum or local minimum (Frisen (1986)). Convex/concave regression, where the first derivatives of the estimated functions are non-decreasing/non-increasing respectively, was first given a least-squares point estimate by Hildreth (1954). The Hildreth estimate was later proved to be consistent by Hanson and Pledger (1976), while its rate of convergence was established by Mammen (1991b). More recently, convex estimation has been considered by Seijo and Sen (2011), Mazumder et al. (2015), Kuosmanen (2008), and Groeneboom et al. (2001).

Isotonic, or monotone, regression and its variants have received the most attention in the statistical literature, perhaps due to their continuing relevance. Monotone regression can be defined most simply as a mapping $f : x \rightarrow y$ where $f$ is non-decreasing or non-increasing over the range of $x$. Monotone regressions have seen diverse applications across research domains; They have been used by Hu et al. (2005) to analyze dose-response in bioinformatics, by Luss et al. (2012) to estimate gene-gene interactions, and by Diggle et al. (1997) to estimate disease risk as functions of spatial exposure, to name just a few examples.

Isotonic regression maximum likelihood estimators with no smoothness requirements were first explored by Brunk (1958) and Grenander (1956). The asymptotic distribution of these estimators was later established by Wright (1981). The Pool Adjacent Violators algorithm for the estimation of the isotonic regression MLE with Gaussian distributed errors was first suggested by Ayer et al. (1955).

The literature regarding isotonic regression diverges at the point of choosing an estimator based on the crucial assumption of whether the resulting function is smooth. For applications in which a step-wise function is acceptable, the estimators of Brunk, Grenander, Ayer, etc., have an exact solution and no tuning parameter. However, for applications in which a smooth function is required, alernate approaches have had to be developed, and a two-step procedure has become common. These two-step procedures either estimate a smooth function and apply a monotonic constraint on the resulting function (Friedman and Tibshirani (1984)), or estimate a monotonic function and then smooth the resulting estimate (Cheng and Lin (1981)). Mammen compares the asymptotic behaviours of these approaches in a comprehensive treatment of smooth, monotonic nonparametric regression. Although these procedures differ in their asymptotic behaviours, all of them require the selection of a tuning parameter (Mammen (1991a)).

### 2.0.2 Partial Linear Models

The Generalized Additive Model, or GAM (see equation [2.1]), was first suggested by Hastie and Tibshirani (1986), and can be seen both as a generalization of the GLM to include nonparametric terms, and as a generalization of additive models to models with error terms from the exponential family. GAMs provide a large amount of flexibility to the nonparametric modeling of multivariate data, but avoid the slow convergence – the curse of dimensionality – associated with multivariate non-parametrics by imposing an additive structure between terms.

$$g(E(X)) \;=\; \beta_0 + \sum_{i=1} f_i(x_i) \tag{2.1}$$

Partial Linear Models, or PLMs, predate GAMs by several years, having been introduced in 1986 by Engle et al. (1986) for the modelling of weather and energy-use. PLMs can nonetheless be considered more productively as a slightly more restrictive subset of GAMs, wherein some proportion of the model terms are required to be linear.

Both GAMs and PLMs have often been fit using the "backfitting" algorithm, first introduced by Breiman and Friedman (1985) and used in the first descriptions of GAMs by Tibshirani and Hastie, though several alternatives have been developed over the years. Notably, Speckman introduced the profile least-squares estimator in 1988 Speckman (1988); Hua proposed the penalized spline estimator Liang (2006); and Hamilton and Truong proposed the local linear estimator Hamilton and Truong (1997). A thorough review of these various estimators, including numerical comparisons, was performed by Liang (2006).

### 2.0.3 Mixture Models

Model-based clustering, or mixture models, are a common method for producing models of latent clusters with probabalistic or "soft" cluster assignment. The history of mixture models is particularly long, with the first such models having been implemented more than a century ago by Newcomb (1886) and Pearson (1894). In much of the following century, progress in the theory of mixture models and their estimation stalled due to a lack of computational power and for lack of an efficient estimating strategy.

This dry spell was called to a close by the introduction by Dempster et al. (1977) of the EM algorithm for estimation of models with supposed latent variables, which vastly simplified the estimation of mixture models. Since then, development and research in mixture modelling has proceeded rapidly, with implementations of Bayesian mixture models (Marin et al. (2005)), infinite mixture models (Rasmussen (2000)), *must-link* and *cannot-link* constraints (Kiri Wagstaff (2000)), and many variations of normal mixture models (McLachlan and Peel (1999), Fraley and Rafter (2012)) all having been published within the last two decades.

Yet more recently, mixtures of regressions have received increased attention (Grün et al. (2008), Viele and Tong (2002), Hurn et al. (2003)) as easily interpretable clustering methods that specify a dependence structure amongst observed covariates without modelling the distributions of the covariates themselves.

### 2.0.4 Mixtures of Nonparametric Regressions

Both Xiang and Yao (2016), and Huang et al. (2013) have considered mixtures of nonparametric regressions. The model of Xiang & Yao estimates mixing proportions and the variance of each component as constants, while allowing the mean of each component to be a nonparametric function of the data. Huang et al., by contrast, propose a model where mixing proportions, mean and variance within each mixture component are all estimated nonparametrically.

Various attempts have been made at generalizing the above approaches to include linear effects, i.e., to model mixtures of partially linear models (PLMs) or generalized additive models (GAMs). Wu and Liu (2017) propose a structure for estimating mixtures of PLMs with a univariate nonparametric effect and arbitrary linear effects per component. Most recently, Zhang and Pan (2020) extended this model to accept arbitrary nonparametric effects.

Within the smaller subset of mixtures of non-parametric regressions, one may frequently encounter situations in which one would like to place shape constraints on some, or each, of the components in our mixture model. There has been relatively little previous work in the modelling of mixtures of specifically monotone nonparametric regressions, with the publication by Zhang et al. standing out as the only treatment of this particular issue.

# Chapter 3

# An Overview of Contributing Models and Estimators

### 3.0.1 Mixture Models and the EM Algorithm

#### General Finite Mixture Models

At its most basic, a finite mixture of $K$ distributions for some positive integer $K$ can be represented by its additive distribution:

$$p(X) \;=\; \sum_{k=1}^{K} \pi_k \cdot p(X|\boldsymbol{\theta}_k) \tag{3.1}$$

where the distribution of each component $k$ is parametrized by some set of parameters $\boldsymbol{\theta}_k$, and the number of observations generated by component $k$ is proportionate to $\pi_k$. Necessarily, all $\pi_k \in (0,1)$, and $\sum_{k=1}^{K} \pi_k = 1$. We assume that each observation generated by the mixture is generated uniquely by one component, such that if we are given some number $n$ of observed values $X_1, ..., X_n$, we can denote their categorization within the components of a mixture by a latent (i.e., unobserved) variable $\Lambda$ such that $\Lambda_i \in \{1, .., K\}$ for all $i \in \{1, ..., n\}$. Then, the distribution of $\Lambda$ can be denoted by equation [3.2], and the additive distribution in equation [3.1] can be deconstructed to the conditional distribution in equation [3.3].

$$p(\Lambda = k) \;=\; \pi_k \tag{3.2}$$

$$p(X_i|\Lambda_i = k) \;=\; p_k(X) \;=\; p(X|\theta_k) \tag{3.3}$$

Typically, however, one is simultaneously estimating the number of components $K$, the mixing proportions $\pi_k$, the parameters $\boldsymbol{\theta}_k$, and the latent variable $\Lambda$, at which point it becomes more useful to consider $\boldsymbol{\Lambda}$ an $n \times k$ matrix of sequentially updated probabilities respresenting the probability of observation $n$ having been generated by component $k$. $\boldsymbol{\Lambda}$ must then meet the condition that $\sum_{k=1}^{K} \boldsymbol{\Lambda}_{ik} = 1$ for all $i \in \{1, ..., n\}$.

This is an extremely flexible basic model structure, with $p(\cdot)$ only needing to be a valid probability density function. Thus, one may have a mixture of gaussians (normals), of binomials, of poissons, etc. These distributions may also be multivariate, and again, there is no restriction on the presumed distribution belonging to each dimension within each cluster $k$. Some specific such models are known by alternate names. For example, with some minimal restrictions added to the mixture of multivariate gaussians, namely, when all component covariances are diagonal, equal, and the variances are infinitesimal or the observations are given “hard assignments” (i.e., $\mathbf{\Lambda}_{ik} \in \{0, 1\}$), one has the well-known $k$-means model.

## Finite Mixtures of Regressions

If one further specifies the distributions $p_k()$ in equation [3.1] as regression functions, one is left with a finite mixture of regressions. The structure of such a mixture can be described without specifying the exact form of either the regression model $Y = f_k(\cdot|\vec{X}) + \epsilon_k$ or the distribution of the random component, $\epsilon_k$, as long as there is a weighted procedure for estimating $\hat{f}(\cdot)$.

Suppose then that, instead of univariate $X$, we observe $Y_i, ..., Y_n$ and associated $\vec{X}_i, ..., \vec{X}_n$. As before, we assume that each observed set $(Y_i, \vec{X}_i)$ belongs to one of $\{1, ..., k\}$ unobserved components for some positive integer $k$, and we denote this by a matrix of probabilities $\mathbf{\Lambda}$. We further assume some vector of regression model parameters $\boldsymbol{\theta}_k$.

Thus equation [3.1] becomes equation [3.4], in which the distribution of $Y$ is conditioned on the associated covariates $\vec{X}$. The likelihood of this model is written out in equation [3.5]. When the likelihood is maximized and parameters are estimated, the model provides the following:

1. The previously mentioned $n \times k$ matrix $\mathbf{\Lambda}$ representing the posterior probability of each $(Y_i, \vec{X}_i)$ belonging to each of $K$ components.

2. A vector $\pi_1, ..., \pi_k$ of prior probabilities representing the mixing proportions of each component in the larger mixture model

3. A set of parameters $\boldsymbol{\theta}_k$ for each regression component $k$

$$p(Y) = \sum_{k=1}^{K} \pi_k p(Y = y \mid X = x, \boldsymbol{\theta}_k) \tag{3.4}$$

$$L(\boldsymbol{\pi}, \boldsymbol{\theta}) = \prod_{i=1}^{n} \sum_{k=1}^{K} \pi_k p_k(y_i \mid \vec{x_i},\ \boldsymbol{\theta}_k) \tag{3.5}$$

In contrast with other mixture models of multivariate data, this is in fact a 1-deminsonal mixture of conditional normals. The convenience of this structuring is that no assumptions or restrictions are placed on covariates $\boldsymbol{X}$. As a result, none of the parameters of the mixture model external to the model components, i.e., neither the priors $\boldsymbol{\pi}$ nor the estimated posterior matrix $\mathbf{\Lambda}$, depend on covariates $\boldsymbol{X}$ except through $Y$. One could choose to include this dependence by modelling the priors $\boldsymbol{\pi}$ as a function of the covariates, i.e., $\boldsymbol{\pi}(\boldsymbol{X})$. The result is what is sometimes called a mixture-of-experts model, with $\boldsymbol{\pi}(\boldsymbol{X})$ being called *gating coefficients* or *gating networks* (Gormley and Frühwirth-Schnatter (2018)).

### The EM Algorithm

The EM algorithm is a method for discovering the parameter estimates that maximize the likelihood of a model which incorporates unobserved variables representing latent clusters. If we denote such a problem as incorporating observations $x_1, ..., x_n$ and associated latent variable $z_1, ..., z_n$ where $z_i \in \{1, ..., k\}$, then the EM algorithm allows us to maximize equation [3.6], which is the general likelihood of a single observation $x$. Note that this equation already appears quite similar to the likelihood of a mixture model. A set of prior probabilities $\pi_k$ is not required, as here we have assumed that each $x_i$ has a *true* cluster assignment, i.e., a "hard" assignment.

$$\ell(\theta) \;=\; \log \sum_{k=1}^{K} p_k(X = x, Z = z|\theta) \tag{3.6}$$

The EM algorithm allows us to avoid determining the maximum of equation [3.6] directly, and instead allows us to iteratively maximize a lower bound of the log-likelihood. To demonstrate this, we introduce an arbitrary distribution over $Z$ called $q(Z)$. This allows us to reformulate the likelihood equation as below:

$$\ell(\theta) \;=\; \log \sum_{k=1}^{K} q(Z) \frac{p_k(X = x, Z = z|\theta)}{q(Z)} \tag{3.7}$$

Since the log-likelihood function is concave, Jensen's inequality (equation [3.8], Jensen (1906)) applies, which specifies that the expectation over a convex function of $X$ is greater or equal to the convex function of the expectation. This gives us the inequality in equation [3.9].

$$E(f(X)) \geq f(E(X)), \text{ for convex function } f() \tag{3.8}$$

$$\log \sum_{k=1}^{K} q(Z) \frac{p_k(X = x, Z = z|\theta)}{q(Z)} \;\geq\; \sum_{k=1}^{K} q(Z) \log \frac{p_k(X = x, Z = z|\theta)}{q(Z)} \tag{3.9}$$

The function $J(q, \theta) = \sum_{k=1}^{K} q(Z) \log \frac{p_k(X=x,Z=z|\theta)}{q(Z)}$ therefore serves as a lower bound to the likelihood. Given this formulation of $J(q, \theta)$, the expectation and maximization steps of the EM algorithm, in equations [3.10] and [3.11] respectively, can be seen as complementary maximizers of $J(q, \theta)$. Since each step only maximizes $J(q, \theta)$ with respect to either $q$ or $\theta$, the steps can only increase $J(q, \theta)$ (or keep it constant), thus giving us the inequalities in equations [3.12] and [3.13]. Moreover, since the lower bound on the likelihood can only increase, the likelihood can only increase, thus demonstrating that the EM algorithm converges to at least a local maximum.

$$q^{(t)} \;=\; \underset{q}{\operatorname{argmax}}\, J(q, \theta^{(t)}) \tag{3.10}$$

$$\theta^{(t+1)} = \underset{\theta}{\operatorname{argmax}}\, J(q^{(t)}, \theta) \tag{3.11}$$

$$J(q^{(t)}, \theta^{(t)}) \geq J(q^{(t-1)}, \theta^{(t)}) \tag{3.12}$$

$$J(q^{(t)}, \theta^{(t+1)}) \geq J(q^{(t)}, \theta^{(t)}) \tag{3.13}$$

### Model-Based Generation and Prediction in Mixtures of Regressions

In typical mixture model frameworks, the model estimate permits the calculation of unconditioned joint distributions of the data and marginal distributions of any given variable. By extension, such typical models permit the generation of new pseudo-observations that conform with the model structure. The mixture of regressions model does not permit this type of generativity unless the distribution of the covariates is known *a priori*, since the distribution of the covariates is not estimated as a part of the model. The mixture of regressions model thus requires a covariate vector $\vec{x}$ in order to produce the marginal distribution of $Y$.

Similarly, whereas typical mixture models can classify new observations in a Bayesian manner by summing over the product of the mixture prior and the density at the new observed data (equation [3.14]), a mixture of regressions model must classify by summing over the density *given* the observed data (equation [3.15]).

$$p(Y = y) = \sum_{k=1}^{K} \pi_k p_k(\vec{Y} = \vec{y} \mid \theta_k) \tag{3.14}$$

$$p(Z = j | X = x, \theta) = \frac{\pi_j p_j(y \mid X = x, \theta_j)}{\sum_{k=1}^{K} \pi_k p_k(y \mid X = x, \theta_k)} \tag{3.15}$$

This is simply the standard application of Bayes' theorem (equation [3.16]), where the densities of $y|X = x$ for each component $k$ are multiplied by the uninformed priors $\pi_k$ and normalized over the marginal distribution of $y|X = x$.

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\int_\theta p(X|\theta)p(\theta)} \tag{3.16}$$

### 3.0.2 Partially Linear Models

#### Additive Linear Models

The general partial linear model is an additive regression model with some finite combination of linear and non-linear components, which can be denoted thus:

$$Y = \sum_{h=1}^{p} g_h(Z_h) + \sum_{j=1}^{q} \beta_j X_j + \epsilon \tag{3.17}$$

where the model has $h$ non-linear covariates and $j$ linear covariates, where each $g_h()$ is some nonparametric function of $Z_h$, and where $\epsilon$ is a random variable with mean 0.

This model overlaps broadly with Generalized Additive Models (GAMs), but differs critically in that we place no restriction on the smoothness of the non-linear components. GAMs are, however, instructive in that they are partly motivated by the difficulty in estimating non-additive, non-parametric models, and the additive structure of our partially linear model is similarly motivated.

More specifically, although nonparametric methods exist for multiple regression, they are faced with the well-known "curse of dimensionality" (Geenens (2011)). The slow convergence associated with the curse of dimensionality is avoided by fitting the less general, additive model, where each term is fit within one dimension. Both GAMs and partial linear models can be estimated using the backfitting algorithm (Breiman and Friedman (1985)), which iteratively estimates the individual terms of an additive model and, up to a user-specified threshold, determines an optimal solution.

## Partially Linear Models with Monotone Constraints

The partial linear model that we apply in the proposed model of this paper can be denoted thus:

$$Y = \sum_{h=1}^{p} g_h(Z_h) \; + \; \sum_{j=1}^{q} \beta_j X_j \; + \; \epsilon \tag{3.18}$$

where the model has $h$ non-linear covariates with monotone shape constraints and $j$ linear covariates, and where $\epsilon \sim Normal(0, \sigma^2)$. The parameters $\vec{\beta}$ and the functions $g_1(\cdot), ...g_p(\cdot)$ are determined as the minimizers of the quadratic loss function, shown in equation [3.19]. The estimation of the functions $g_1(\cdot), ...g_p(\cdot)$ is a problem of additive isotonic regression, discussed in the next subsection.

$$\{\hat{\vec{\beta}}, \hat{g}_1, ..., \hat{g}_p\} = \underset{\vec{\beta}, g_1:g_p}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \sum_{h=1}^{p} g_h(z_{ih}) \; - \; \sum_{j=1}^{q} \beta_j x_{ij})^2 \tag{3.19}$$

The MLE of the entire partial linear model is obtained via the backfitting algorithm, iterating through the two-step process (equations [3.20], [3.21]) until convergence.

$$(I) \qquad \{\hat{g}_1, ..., \hat{g}_p\} = \underset{g_1:g_p}{\operatorname{argmin}} \sum_{i=1}^{n} \left( y_i - \sum_{j=1}^{q} \beta_j x_{ij} - \sum_{h=1}^{p} g_h(z_{ih}) \right) \qquad \textit{holding} \;\; \vec{\beta} \;\; \textit{fixed} \tag{3.20}$$

$$(II) \qquad \hat{\vec{\beta}} = \underset{\vec{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left( y_i - \sum_{h=1}^{p} g_h(z_{ih}) - \sum_{j=1}^{q} \beta_j x_{ij} \right) \qquad \textit{holding} \;\; \{g_1, ..., g_p\} \;\; \textit{fixed} \tag{3.21}$$

This model and estimator is thoroughly explored by Cheng (2009), where it is shown that, under certain conditions (see section [A.1] of the appendix), $\hat{\beta}_n$ is $\sqrt{n}$-consistent (equation [3.23]), while the estimates $\{\hat{g}_1, ..., \hat{g}_p\}$ converge in distribution as to a two-sided brownian motion plus a parabola (equation [3.22]).

$$\begin{gathered} n^{1/3}\frac{(2p_{Z_h}(z_h))^{1/3}}{\sigma^{2/3}g_h(z_h)^{1/3}}[\hat{g}_h(z_h) - g_h(z_h)] \xrightarrow{d} GCM(Z(t)+t^2) \\ \text{where } p_{Z_h}(z_h) \text{ is the density of } Z_h \text{ evaluated at } z_h, \\ GCM(Z(t)) \text{ is the greatest convex minorant of } Z(t) \\ \text{and } Z(t) \text{ is a two-sided Brownian motion} \end{gathered} \tag{3.22}$$

$$\begin{gathered} \sqrt{n}(\hat{\beta}_n - \beta_0) \xrightarrow{d} N(0, \Sigma) \\ \text{where } \Sigma = \sigma^2[E(X - \sum_{h=1}^{p} E(X|Z_h))^{\otimes 2}]^{-1} \\ \text{and } \sigma^2 = Var(\epsilon) \end{gathered} \tag{3.23}$$

The asymptotic distribution of $\hat{\beta}$ evidently has a larger variance than in the case of the ordinary least squares estimate. As Cheng points out, this can be considered the cost of including non-parametric terms in the regression model.

The rate of convergence for the non-parametric terms is slower than that of the linear terms, as expected, but significantly faster than would be the case if the nonparametric terms were multivariate monotone rather than univariate monotone and additive. Moreover, Cheng demonstrates that in the additive case, the estimators $\{\hat{g}_1, ..., \hat{g}_p\}$ possess the oracle property, meaning each $\hat{g}(\cdot)$ can be estimated as well as it could be if all other components, including all other $\hat{g}(\cdot)$, are known. Thus the rate of convergence of any $\hat{g}(\cdot)$ is not diminished by the inclusion of other monotone terms.

### 3.0.3 Isotonic Regression

#### Univariate Isotonic Regression

At the most basic level, with univariate $x$ and $y$ and a simple ordering amongst $x$ such that $x_1 \leq x_2 \leq ... \leq x_n$ for all $x_i \in X$, isotonic regression determines a non-decreasing function $g(\cdot)$ such that $g(x_1) \leq g(x_2) \leq ... \leq g(x_n)$ and for which $\hat{g}(\cdot) = \text{argmin}_g \sum_{i=1}^{n} ||g(x_i) - x_i||_L$ for some loss function $|| \cdot ||_L$. If observations are weighted, the objective function becomes $\hat{g}(\cdot) = \text{argmin}_g \sum_{i=1}^{n} w_i||g(x_i) - x_i||_L$ for weights $w$. In the so-called antitonic case, the function $g(\cdot)$ is non-increasing such that $g(x_1) \geq g(x_2) \geq ... \geq g(x_n)$. Without loss of generality, from this point on we are only concerned with the non-decreasing case.

If we consider specifically the squared error loss, our risk function and weighted risk function become equations [3.24] and [3.25] respectively. Equation [3.25] can alternately be written explicitly in the form of a min-max formula, as in equation [3.26] (Jordan et al. (2019)), giving it a characterization which facilitates the study of the properties of the estimator $\hat{g}(\cdot)$. Equation [3.26] breaks the function $g(\cdot)$ into non-decreasing "blocks", and assigns to each block the weighted mean of the values $x$ contained in that block.

$$\hat{g}(\cdot) = \underset{g}{\operatorname{argmin}} \sum_{i=1}^{n} (g(x_i) - x_i)^2 \tag{3.24}$$

$$\hat{g}(\cdot) = \underset{g}{\operatorname{argmin}} \sum_{i=1}^{n} w_i (g(x_i) - x_i)^2 \tag{3.25}$$

$$\hat{g}(x_i) = \min_{j \geq i} \max_{k \leq j} \frac{\sum_{k=k}^{j} w_k x_k}{\sum_{k=k}^{j} w_k}, \; i = 1, ..., n \tag{3.26}$$

One can see from equation [3.26] that determining the estimate $\hat{g}(\cdot)$ for the least square isotonic regression returns a step function, and requires no tuning parameter.

### The Pool Adjacent Violators Algorithm

A well-known and efficient way to obtain the least square estimate of $g()$ is through the Pool Adjacent Violators Algorithm (PAVA). The PAVA – for univariate monotone regression (equation [3.27]) – returns a step-function fit without either having to select a bandwidth or having to set a congergence tolerance parameter. For multivariable monotone regression (equation [3.28]), one must take a different approach, suggested by Bacchetti, called the Cyclic Pool Adjacent Violators Algorithm (CPAV). Within the CPAV, one iterates through each univariate function sequentially and update univariate monotone functions until convergence, returning the additive model of equation [3.28].

$$Y = g(X) \; + \; \epsilon \tag{3.27}$$

$$Y = \sum_{h=1}^{p} g_h(X_h) \; + \; \epsilon \tag{3.28}$$

# Chapter 4

# Proposed Model

### 4.0.1 Model Definition

The model proposed in this article has the following structure:

$$Y = \begin{cases} \sum_{h=1}^{p} g_{h1}(Z_h) \ + \ \sum_{j=1}^{q} \beta_{j1} X_j \ + \ \epsilon_1 \text{ with probability } \pi_1; \\ \vdots \\ \sum_{h=1}^{p} g_{hk}(Z_h) \ + \ \sum_{j=1}^{q} \beta_{jk} X_j \ + \ \epsilon_k \text{ with probability } \pi_k; \end{cases} \tag{4.1}$$

where $\pi_k$ represents the prior probability of mixture component $k$; $g_{hk}(\cdot)$ represents the monotone function of variable $h$ within component $k$; $\beta_{jk}$ represents the linear effect of variable $j$ within mixture component $k$; and $\epsilon_k$ represents the error associated with component $k$. All $\epsilon_k$ are assumed to be normally distributed with mean 0, and are assumed to be independent of the covariates $(\boldsymbol{X}, \boldsymbol{Z})$ . All $\pi_k$ are assumed to be unknown constants, and all $\pi_k \in (0, 1)$ such that $\sum^K \pi_k = 1$.

There is no requirement that the $K$ regression functions in model [4.1] be identical. Specifically, $g_{hk}()$ for any $h \in p$ and any $k \in K$ can be set as monotone non-increasing, monotone non-decreasing, or absent, regardless of the other $g_{hk}$. Likewise, the number of linear effects $\beta_j$, including the intercept $\beta_0$, need not be same across different components $k$.

### 4.0.2 Model Estimation

The proposed model is obtained from a series of nested, iterative algorithms, described below. Algorithm 1 describes the EM algorithm for fitting mixture priors and observation posteriors. Algorithm 2 describes the weighted partial linear regression for the fitting of each component within each M-step of the EM algorithm. Algorithm 3 describes the weighted, cyclic pool adjacent violators algorithm for cases where there is more than one monotone function fit within a single partial linear regression. Algorithm 4 describes the weighted pool adjacent violators algorithm for fitting a single monotone regression. In all cases, convergence thresholds are set by the user.

**Algorithm 1** EM algorithm for Finite Mixtures of Regressions

```
Require:
    $x$ — an $n \times p$ matrix (independent variables with no shape constraint)
    $z$ — an $n \times q$ matrix (independent variables with monotone shape constraint)
    $y$ — an $n \times 1$ matrix (dependent variable)
    $k$ — a positive integer representing the number of categories of latent variable L
Result:
    $\Lambda$ — an $n \times k$ matrix representing the posterior probability of observation $i = 1, ..., n$ belonging to latent category $j = 1, ..., k$. Additionally, for all $i = 1, ..., n$ and $j = 1, ..., k$, $\Lambda_{ij}$ is a real number in the range $[0, 1]$, and $\sum_{j=1}^{k} \Lambda_{ij} = 1$
    $\vec{\pi}$ — a vector $\pi_1, ..., \pi_k$ of prior probabilities representing the mixing proportions of each component in the larger mixture model
    $\vec{\Theta}$ — a set of parameters $\Theta_k$ for each regression component $k$

 1: Set iteration index $d \leftarrow 1$
 2: for $i \in 1, ...n$ do
 3:     With uniform probability across $k$, assign one of the elements of $[\Lambda_{i1}, ..., \Lambda_{ik}]$ to 1 and all other to 0, such that $\Lambda_i = [0, ..., 1, ..., 0]$
 4: end for
 5: while algorithm is not converged do
 6:     for $j \in 1, ..., k$ do                                          ▷ Begin M-step
 7:         Set prior mixture proportion $\pi_j^{(d)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \Lambda_{ij}^{(d-1)}$
 8:         Set weighted partial linear model regression parameters such that $[\hat{\beta}_j, \hat{g}_j]^{(d)} \leftarrow \underset{\beta, g}{\operatorname{argmin}} \sum_{i=1}^{n} \Lambda_{ij}^{(d-1)} (y - x\beta_j - g_j(z))^2$ (See Algorithm 2, WPLR)
 9:     end for                                                           ▷ End M-step
10:     for $i \in 1, ..., n$ do                                          ▷ Begin E-step
11:         for $j \in 1, ..., k$ do
12:             Set $\Lambda_{ij}^{(d)} \leftarrow \pi_j^{(d)} p(y_i | x_i, \beta_j^{(d)}, g_j^{(d)}(z_i))$, where $p(y_i | x_i, \beta_j^{(d)}, g_j^{(d)}(z_i))$ is the density of a $Normal$ distribution with $\mu = x_i \beta_j^{(d)} + g_j^{(d)}(z_i)$ and $\sigma = \sqrt{\frac{\sum w_i r^2 / \bar{w}}{n - rk(X)}}$
13:         end for
14:         Normalize the posterior probabilities $[\Lambda_{i1}, ..., \Lambda_{ik}]^{(d)}$ such that $\sum_{j=1}^{k} \Lambda_{ij}^{(d)} = 1$
15:     end for                                                           ▷ End E-step
16:     $d = d + 1$
17: end while
```

**Algorithm 2** Weighted Partial Linear Regression

```
Require:
    $x$ — an $n \times p$ matrix (independent variables with no shape constraint)
    $z$ — an $n \times q$ matrix (independent variables with monotone shape constraint)
    $y$ — an $n \times 1$ matrix (dependent variable)
    $w$ — an $n \times 1$ matrix (observation weights)
Result:
    $\hat{\beta}, \hat{g_1}, ..., \hat{g_q}$ such that $[\hat{\beta}, \hat{g_1}, ..., \hat{g_q}] = \underset{\beta, g_1, ..., g_q}{\text{argmin}} \sum_{i=1}^{n} w_i (y_i - \sum_{h=1}^{q} g_h(z_{ih}) - x_i \beta)^2$
 1: Set iteration index $b \leftarrow 1$
 2: Set $\hat{\beta}^{(0)} \leftarrow \beta_x$, where $[\beta_x, \beta_z] = \underset{\beta_x, \beta_z}{\text{argmin}} \sum_{i=1}^{n} w_i (y_i - z_i \beta_z - x_i \beta_x)^2$
 3: while algorithm is not converged do
 4:     if $z$ is univariate then                                   ▷ See Algorithm 4, PAVA
 5:         Set $\hat{g}^{(b)} \leftarrow \underset{g}{\text{argmin}} \sum_{i=1}^{n} w_i ([y_i - x_i \beta^{(b-1)}] - g(z_i))^2$ holding $\beta^{(b-1)}$ fixed.
 6:     else                                                        ▷ See Algorithm 3, CPAV
 7:         Set $\sum_{h=1}^{q} \hat{g}_h^{(b)} \leftarrow \underset{g_1, ..., g_q}{\text{argmin}} \sum_{i=1}^{n} w_i ([y_i - x_i \beta^{(b-1)}] - \sum_{h=1}^{q} g_h(z_{ih}))^2$ holding $\beta^{(b-1)}$ fixed.
 8:     end if
 9:     Set $\hat{\beta}^{(b)} \leftarrow \underset{\beta}{\text{argmin}} \sum_{i=1}^{n} w_i ([y_i - g^{(b)}(z_i)] - x_i \beta)^2$ holding $g^{(b)}$ fixed.
10:     $b = b + 1$
11: end while
```

**Algorithm 3** Weighted Cyclic Pool Adjacent Violators Algorithm

```
Require:
    $x$ — an $n \times q$ matrix (independent variables)
    $z$ — an $n \times 1$ vector (observation weights)
    $y$ — an $n \times 1$ vector (dependent variable)
Result:
    A set of non-decreasing functions $\hat{f}_1, ...\hat{f}_q$ such that
    $[\hat{f}_1, ...\hat{f}_q] = \underset{f_1, ...f_q]}{\text{argmin}} \sum_{i=1}^{n} w_i (y_i - \sum_{h=1}^{q} f_h(x_{ih}))^2$
 1: Set iteration index $m \leftarrow 1$
 2: while algorithm is not converged do
 3:     for $h \in 1, ..., q$ do                                    ▷ See Algorithm 4, PAVA
 4:         Set $\hat{f}_h \leftarrow \underset{f_h}{\text{argmin}} \sum_{i=1}^{n} w_i ([y_i - \sum_{\substack{j=1 \\ j \neq h}}^{q} f_j(x_{ih})] - f_h(x_{ih}))^2$ holding all $f_j(), j \neq h$ fixed
 5:     end for
 6:     $m = m + 1$
 7: end while
```

**Algorithm 4** Weighted Pool Adjacent Violators Algorithm

```
Require:
    $x$ — an $n \times 1$ vector (independent variable)
    $w$ — an $n \times 1$ vector (observation weights)
    $y$ — an $n \times 1$ vector (dependent variable)
Result:
    A non-decreasing function $\hat{f}(\cdot) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} w_i(y_i - f(x_i))^2$
 1: Set iteration index $l \leftarrow 0$
 2: Set blocks $r \leftarrow 1, ..., B$ where at $l = 0$, $B = n$
 3: Set $f^{(l=0)}(x_i) \leftarrow y_i$
 4: Set initial block membership $f^{(l=0)}(x_i) \in r_i$
 5: while any $f_r^l(x) \geq f_{r+1}^l(x)$ do
 6:     if $f_r^l(x) > f_{r+1}^l(x)$ then
 7:         Merge blocks $r$ and $r + 1$
 8:     end if
 9:     Solve $f_r^{(l)}()$ for block $r$ as the weighted mean, i.e, $f_r() = \frac{1}{\sum_{i=1}^{n} w_i} \sum_{i=1}^{n} w_i(y_i)$ for all $x \in r$
10:     $l = l + 1$
11: end while
```

### 4.0.3 Asymptotic Properties of Model and Estimator

Under various sets of regularity assumptions, other authors have been able to derive analytically the asymptotic variance of estimators for mixtures of various types of non-parametric and semi-parametric regressions (see, for example, the publications of Zhang and Zheng (2018), Huang et al. (2013), Zhang and Pan (2020)). This can be helpful if the analytic variance allows for the construction of simple, parametrized confidence intervals for model coefficients, or if the analytic representation gives some insight into the asymptotic shape and spread of the variance.

We know that the asymptotic variance of the model will be some function of the asymptotic variance of the model components. We therefore know that in this case, given the findings of Cheng (2009), the asymptotic variance of the model will be functions of a tensor power of a sum over the expectations of the linear covariates given the non-linear covariates (see equations [3.22] and [3.23] for the complete variance as proven by Guang). As Guang points out, these expectations (i.e., $\sum_{h=1}^{p} E(X|Z_h)$ can be estimated by a kernel method.

This means that calculating the analytic variance of the model would require the selection of a tuning parameter, whereas one of our model's primary strengths is its lack of tuning parameters. Moreover, the variance of both the linear coefficients and the nonparametric functions, as presented by Guang, are sufficiently complex as to offer little in the way of intuitive understanding. For these reasons, we opt to forego the analytic demonstration of the model variance and instead calculate the distributions of model estimates via the bootstrap.

### 4.0.4 Confidence Intervals via Bootstrapping

We opt to implement and demonstrate the proposed model with confidence intervals calculated via bootstrapping. Although the ordinary bootstrap applied to mixture models delivers all the typical advantages of bootstrapping for determining parameter confidence intervals (Efron (1979)), it runs into the well-known label-switching problem inherent in clustering algorithms (Grün and Leisch (2009)). Specifically, when the mixture model is run multiple times, as with the bootstrap, the labeling of the resulting clusters is random and there is no guarantee that clusters with the same labels in two different models will represent the same underlying mixture component.

One partial solution to this problem, which we apply in the current paper, is to select the estimated parameters and non-parametric functions of the complete model as the starting values when estimating each of the bootstrapped models. This results in both a decreased computational burden in calculating the bootstrapped estimates, and the bypassing of the label-switching problem, since bootstrap-estimated components will likely share the same label as the component from which their starting values were drawn. This is the solution proposed by Grün and Leisch (2009) and implemented in the Flexmix package (see section [A.2]).

The disadvantage of this approach is that it ignores the possibility of bootstrap-estimated parameters reaching local likelihood maxima. For this reason, this approach underestimates the parameter variance. Moreover, if the first model which one fit returned estimates from a local likelihood maximum instead of a global maximum, subsequent bootstrap estimates may likewise be caught in the same local maximum. Parameter distributions from such a bootstrap might then be heavily biased.

An alternative to the bootstrap paradigm is motivated by the selective inference approach for clustering, introduced by Gao et al. (2020). The approach of Gao et al. dispenses with the need for bootstrapping and instead provides an exact, finite-sample test for the difference in means between two components of a fitted cluster model, e.g., $C_1$ and $C_2 \in C(x)$. The test depends upon the computation of a set of "perturbed" datasets $S$ in which the observations belonging to clusters $C_1$ and $C_2$ are "pushed" or "pulled" apart (with respect to a predefined distance measure) but for which $C_1$ and $C_2$ still emerge as clusters. Although several difficulties remain in applying such an approach to our proposed model – namely, selecting an appropriate distance measure, computing the set $S$, and generalizing the approach to clusters of more than two components – the application of such an approach would yield exact inference *and* solve the label-switching problem, as well as possibly decreasing the computational burden. We leave this problem to future research.

### 4.0.5 Computational Complexity of Estimator

A common concern amongst users of this model will be the speed of the estimator, and by extension, the computational complexity of the estimator. The complexity of our estimator is $O(aknb(q^2 + cp))$, where $a$ is the number of iterations of the EM algorithm, $b$ is the number of iterations of the weighted partial linear regression algorithm, $c$ is the number of iterations of the CPAV algorithm, $n$ is the total number of observations, $k$ is the number of components, $p$ is the number of monotone terms in each component, and $q$ is the number of linear terms in each component. Since the complexity of the PAVA algorithm is $O(n)$ and the complexity of linear regression is $O(nq^2)$ (Best and Chakravarti (1990), Trip (2018)), the complexity of our estimator is quadratic only in $q$, the number of linear terms per component, and linear in all other terms.

However, the complexity of the estimator can easily come to be dominated by the iteration terms $a$, $b$, and $c$, making the practical complexity difficult to anticipate given that the number of iterations within each optimization step of the algorithm is problem-dependent. For a more intuitive understanding of the estimator complexity, we compare the complexity empirically for different types and numbers of features within the sub-component regression models via timed applications on pseudo-data. In the benchmarking table below, we compare the run-time of the estimation of 4 models – with one monotone covariate and no linear effects; with two monotone covariates and no linear effects; with one monotone covariate and four linear effects; with two monotone covariates and three linear effects – and all with the number of components, 4, known *a priori*.

| Test | Replications | Elapsed | Relative |
|---|---|---|---|
| Bivariate monotone with linear effects | 50 | 664.342 | 38.430 |
| Bivariate monotone without linear effects | 50 | 782.844 | 45.285 |
| Univariate monotone with linear effects | 50 | 17.287 | 1.000 |
| Univariate monotone without linear effects | 50 | 25.740 | 1.489 |

**Table 4.1:** Elapsed time for four different types of model constructions – with and without linear effects, and with and without multiple monotone terms.

One can see that – for a model with 4 latent components – adding a second monotone nonparametric effect within each component multiplies the computation time by approximately 50. This is an indication of the heavy cost of increasing even slightly the dimensionality of the non-parametric estimation within each component model.

By comparison, adding linear effects within the component models comes essentially for free. In fact, the estimation of models with univariate monotone effects and 4 linear effects is *faster* than the complementary model without linear effects.

As stated previously, the difference in speed between fitting models with monovariate and multivariate monotone terms is a function of the maximum number of iterations and the tolerance threshold set on the EM algorithm, the backfitting sequence in the partial linear model estimator, and the backfitting sequence in the CPAV sub-estimator (see algorithms [2] and [3]). All of these parameters are set by the user, and there is as of yet no automated way of selecting parameters for the optimal convergence rate of the algorithm. We leave this problem to future research.

# Chapter 5

# Model Applications

### 5.0.1 Simulated Data

In this section, we demonstrate the application of the proposed model by fitting it to randomly generated pseudo-data. We begin by modeling 1000 observations generated from 2 latent categories with the following underlying structure:

$$\begin{aligned} Y_1 &= 50 + X^3 + \epsilon_1 \\ Y_2 &= -50 + 0.04 \cdot X^5 + 30 \cdot X + \epsilon_2 \end{aligned} \tag{5.1}$$

where

$$\begin{aligned} \epsilon_1 &\sim N(0, 200) \\ \epsilon_2 &\sim N(0, 300) \end{aligned} \tag{5.2}$$

and

$$\begin{aligned} \pi_1 &= 0.65 \\ \pi_2 &= 0.35 \end{aligned} \tag{5.3}$$

and

$$X \sim Uniform(-10, 10) \tag{5.4}$$

We proceed to estimate a mixture of univariate regressions (model [5.5]), with the number of components known *a priori* as 2. The fitted model includes only monotone non-decreasing function of covariate $X$, and no intercept. The regression models are identical for each of the 2 components.

$$Y = \sum_{k=1}^{2} \pi_k (g_k(X) \ + \ \epsilon_k) \tag{5.5}$$

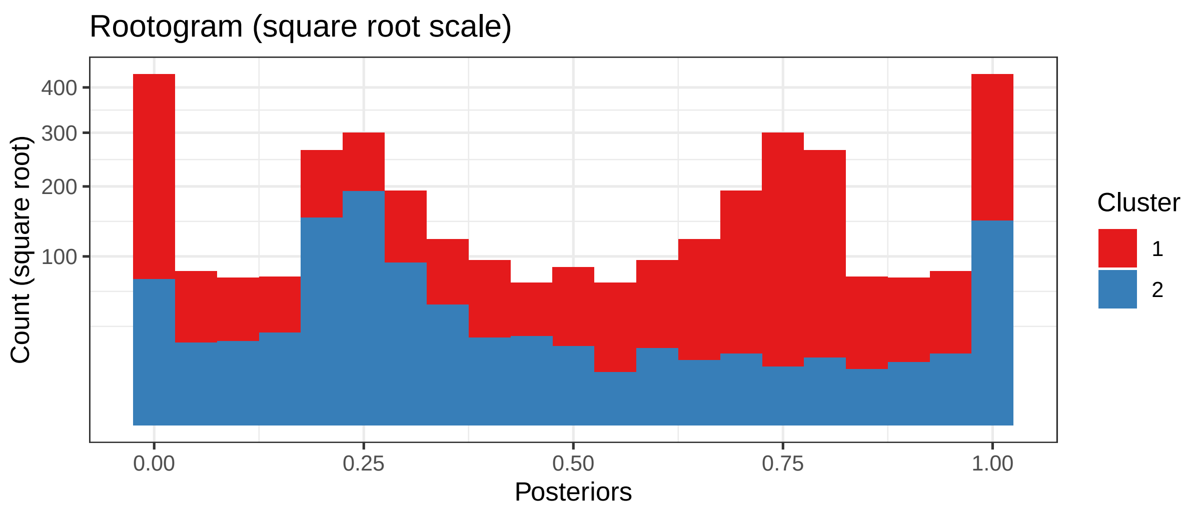

**Figure 5.1:** The rootogram of the two-component mixture model shows the distribution of posterior probabilities with reference to the binary latent variable, for all observations used to fit the model. The model indicates higher confidence in the identification of clusters and the classification of individual observations when the observations accumulate near the limits of the rootogram, at 0 and 1. Conversely, greater mass at the center of the rootogram represents observations that are less confidently classified.

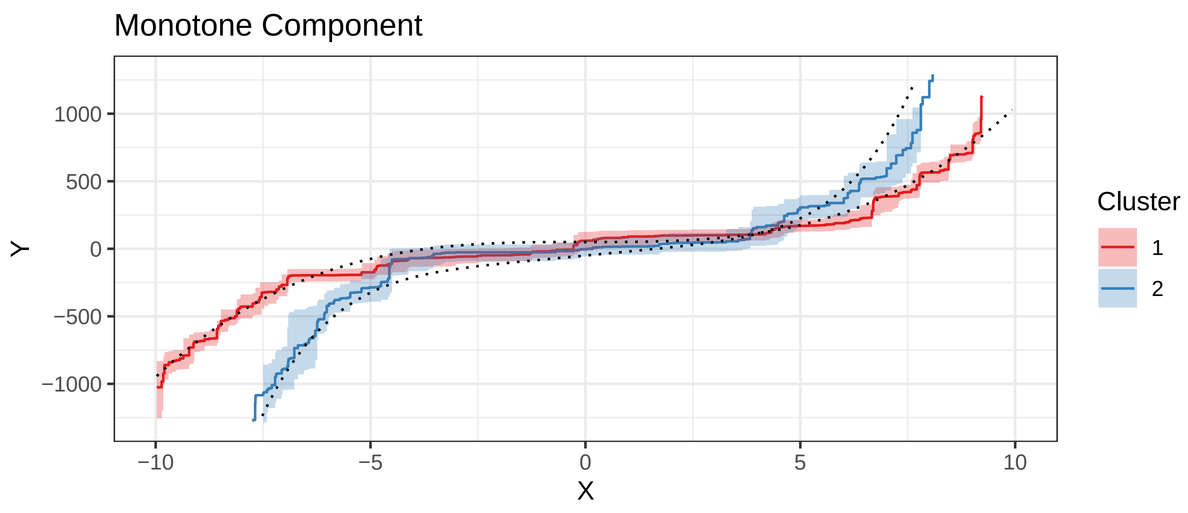

**Figure 5.2:** The estimated monotone functions of the two-component mixture model, with overlaid, dotted black lines representing the true functions. The confidence intervals are generated by 500 iterations of an ordinary (non-parametric) bootstrap.

As can be seen from figure [5.2], the algorithm is more biased in its estimate where the true data generating functions run in parallel.

We continue the demonstration with the inclusion of linear effects. For the next model, we generate pseudo-data from 4 latent categories with the following underlying structure:

$$\begin{aligned} Y_1 &= 10 + X_1^3 + 1.5 \cdot X_2 - 1.5 \cdot X_3 - X_4 + X_5 + \epsilon_1 \\ Y_2 &= -10 + 40 \cdot X_1 + 3 \cdot X_2 + 2 \cdot X_3 - 2 \cdot X_4 + 2 \cdot X_5 + \epsilon_2 \\ Y_3 &= -4 + 2 \cdot (X_1^3) - 2 \cdot X_2 - X_3 + 2 \cdot X_4 + 4 \cdot X_5 + \epsilon_3 \\ Y_4 &= 4 + 0.1 \cdot (X_1^5) - 3 \cdot X_2 - 3 \cdot X_3 - 3 \cdot X_4 + 3 \cdot X_5 + \epsilon_4 \end{aligned} \tag{5.6}$$

where

$$\begin{aligned} \epsilon_1 &\sim N(0,50) \\ \epsilon_2 &\sim N(0,60) \\ \epsilon_3 &\sim N(0,60) \\ \epsilon_4 &\sim N(0,40) \end{aligned} \qquad \begin{aligned} X_1 &\sim Uniform(-5,5) \\ X_2 &\sim Uniform(-10,10) \\ X_3 &\sim Uniform(-100,100) \\ X_4 &\sim Uniform(-100,100) \\ X_5 &\sim Uniform(-100,100) \end{aligned} \tag{5.7}$$

and

$$\pi_1 = \pi_2 = \pi_3 = \pi_4 \tag{5.8}$$

We proceed to estimate a mixture of partial linear regressions (model [5.9]), with the number of components known *a priori* as 4. The fitted model includes one monotone non-decreasing function of covariate $X_1$, an intercept, and a linear effect for each of $X_2, ..., X_5$. The regression models are identical for each of the 4 components.

$$Y = \sum_{k=1}^{4} \pi_k (g_k(X_1) \;+\; \beta_{0,k} \;+\; \beta_{1,k} \cdot X_2 \;+\; \beta_{2,k} \cdot X_3 \;+\; \beta_{3,k} \cdot X_4 \;+\; \beta_{4,k} \cdot X_5 \;+\; \epsilon_k) \tag{5.9}$$

Tables [5.1] through [5.5] show 89 % confidence intervals for the estimates of the linear effects and priors of model [5.9]. It should be noted that, since these estimates are derived from the nonparametric bootstrap, there are no constraints on the shape of the estimate distributions, meaning they may not be either symmetric or unimodal, and information is therefore lost by summarizing their complete distributions by means and confidence intervals. Moreover, since each bootstrap repetition returns an associated set of estimates, there is known covariance between the estimates. One may represent both full distributions and covariance of up to three linear effects at a time with three-dimensional plots of the covariate space, though with more than three linear effects the interpretation of such plots becomes more difficult. We choose no to report the covariance structure here, though users may find it useful in other applications.

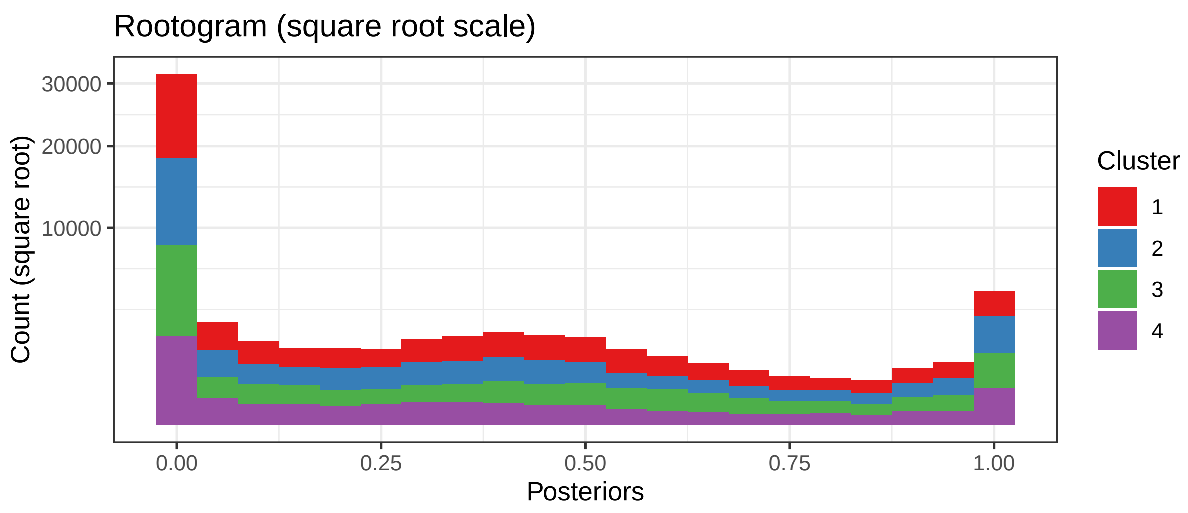

**Figure 5.3:** The rootogram of the four-component mixture model represented by equation [5.9].

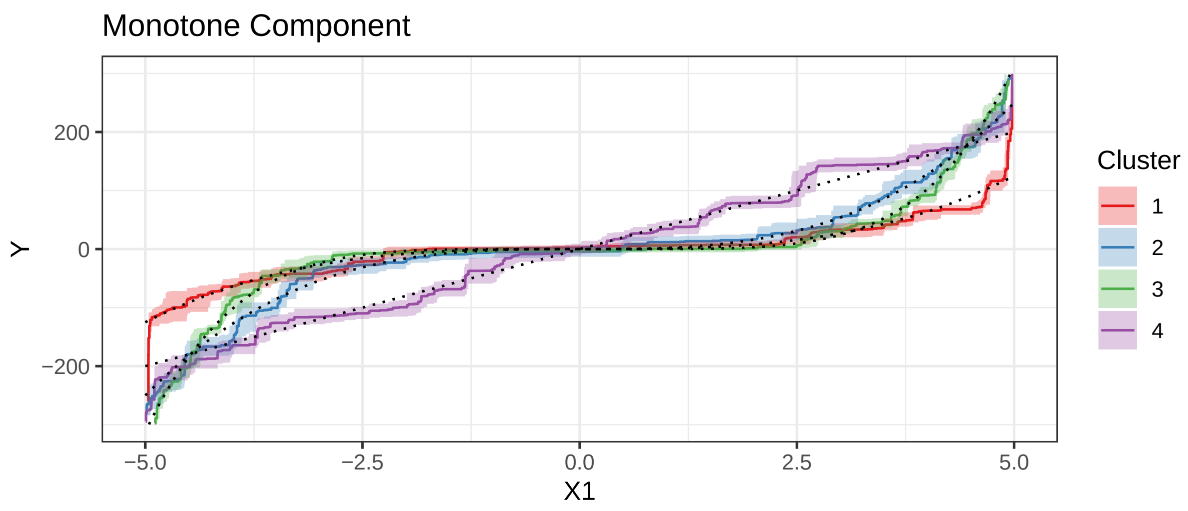

**Figure 5.4:** The estimated monotone functions of the four-component mixture model, with overlaid, dotted black lines representing the true functions. The confidence intervals are generated by 500 iterations of an ordinary (non-parametric) bootstrap.

**Table 5.1:** Component 1

| | lower | mean | upper |
|---|---|---|---|
| (Intercept) | -11.91 | -7.37 | -3.21 |
| X2 | -2.63 | -1.43 | -0.21 |
| X3 | -1.11 | -1.01 | -0.90 |
| X4 | 1.95 | 2.04 | 2.13 |
| X5 | 3.91 | 4.02 | 4.12 |
| sigma | 61.00 | 65.48 | 68.74 |

**Table 5.2:** Component 2

| | lower | mean | upper |
|---|---|---|---|
| (Intercept) | -0.71 | 9.91 | 19.79 |
| X2 | 0.20 | 1.93 | 3.45 |
| X3 | -1.67 | -1.44 | -1.10 |
| X4 | -1.17 | -0.88 | -0.59 |
| X5 | 0.53 | 0.72 | 0.94 |
| sigma | 123.67 | 131.94 | 141.23 |

**Table 5.3:** Component 3

| | lower | mean | upper |
|---|---|---|---|
| (Intercept) | -24.97 | -17.84 | -8.91 |
| X2 | 0.32 | 1.73 | 2.92 |
| X3 | 1.79 | 1.89 | 2.03 |
| X4 | -2.14 | -2.04 | -1.93 |
| X5 | 1.95 | 2.10 | 2.24 |
| sigma | 91.67 | 97.09 | 102.44 |

**Table 5.4:** Component 4

| | lower | mean | upper |
|---|---|---|---|
| (Intercept) | -6.48 | -1.66 | 5.21 |
| X2 | -4.64 | -3.22 | -2.18 |
| X3 | -3.18 | -3.09 | -3.02 |
| X4 | -3.20 | -3.12 | -3.05 |
| X5 | 2.80 | 2.94 | 3.05 |
| sigma | 67.87 | 74.42 | 79.18 |

**Table 5.5:** Priors

| lower | mean | upper |
|---|---|---|
| 0.22 | 0.24 | 0.26 |
| 0.21 | 0.23 | 0.26 |
| 0.25 | 0.27 | 0.28 |
| 0.25 | 0.26 | 0.28 |

### 5.0.2 Algorithm Comparisons with Simulated Data

Here, we empirically compare the behaviour of our proposed model with that of the Zhang et al. model. We fit both models on four separate datasets, each with two components. The first dataset is generated from a model where the true priors of the mixture components $\pi_k$, as well as the true variance of those components $\sigma_k$, are functions of observed covariates, i.e., $\pi_k(X)$ and $\sigma_k(X)$. The second dataset is generated from a model where the true priors and true variances, $\pi_k$ and $\sigma_k$, are constant. The third dataset is generated from a model with constant true priors and true variances, as well as much "jagged" true underlying functions. The fourth dataset, again, is generated from a model with constant true priors and true variances, as well as true underlying functions that twice intersect. Because of the label-switching problem, we choose to plot and compare the resulting fitted models without reference to the component labels. Instead, we overlay component mean estimates sequentially in order to try to compare the general bias and variance behaviours of the two respective model classes. In each pair of comparison plots, the true underlying functions are represented by dotted black lines.

The models resulting from the first dataset, seen in figure [5.5], show that the Zhang et al. model produces tight fits around the true functions, with very low variance. Our proposed model also fits tightly around the true functions when its component estimates are well-separated, but it frequently encounters a component-switching problem where the function estimate is in fact a hybrid of the underlying true functions. When the underlying model has constant true priors and variances, as in figure [5.6], our model is slightly better behaved, although the component-switching problem remains.

In the third dataset, shown in figure [5.7], our model follows the jagged breaks and changes in the true underlying functions with no visible increase in variance. The Zhang et al. estimator, on the other hand, become severely biased and confidently estimates an incorrect pair of functions. When the Zhang et al. estimator does estimate the correct function neighbourhood, it smooths over the sharp edges of the underlying function. In the fourth dataset, shown in figure [5.8], where the true functions intersect, the Zhang et al. model confidently estimates smooth, non-intersecting functions. Our model, on the other hand, correctly identifies the function intersections.

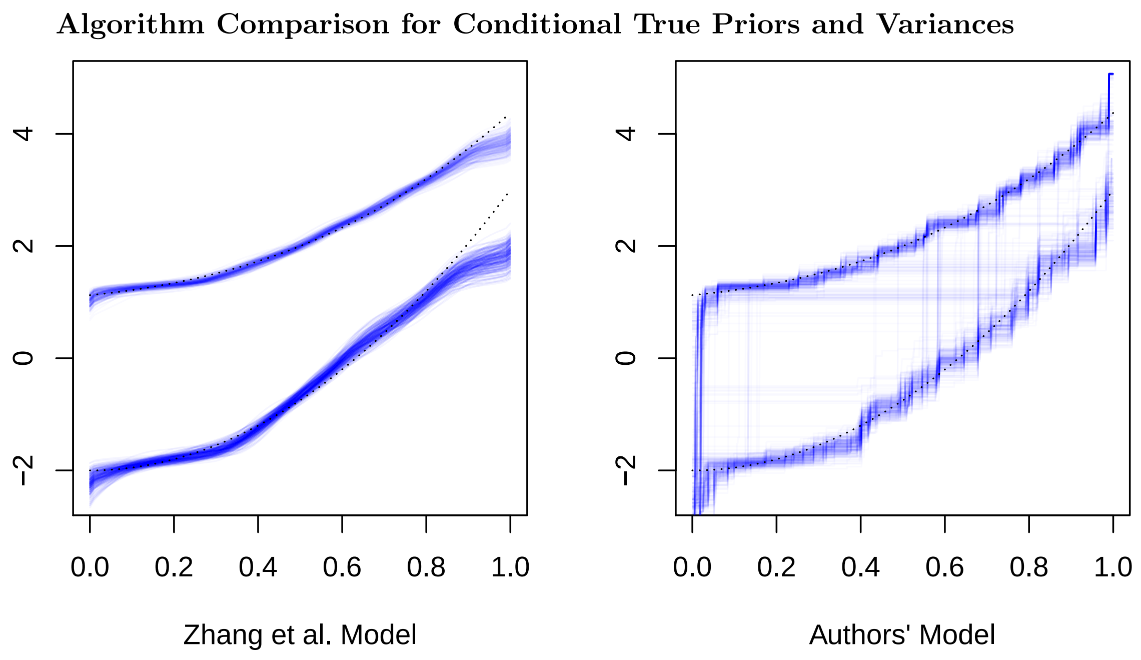

**Figure 5.5:** The Zhang et al. model and the authors' model, fit to data generated from an underlying model with conditional true priors and variances.

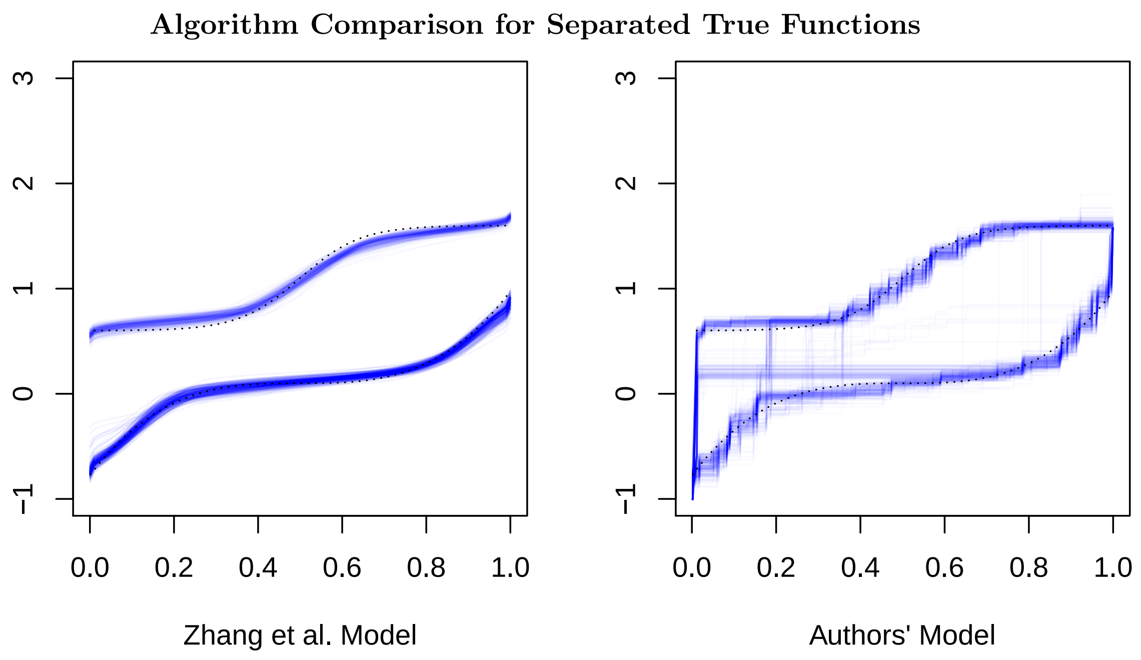

**Figure 5.6:** The Zhang et al. model and the authors' model, fit to data generated from an underlying model with constant true priors and variances.

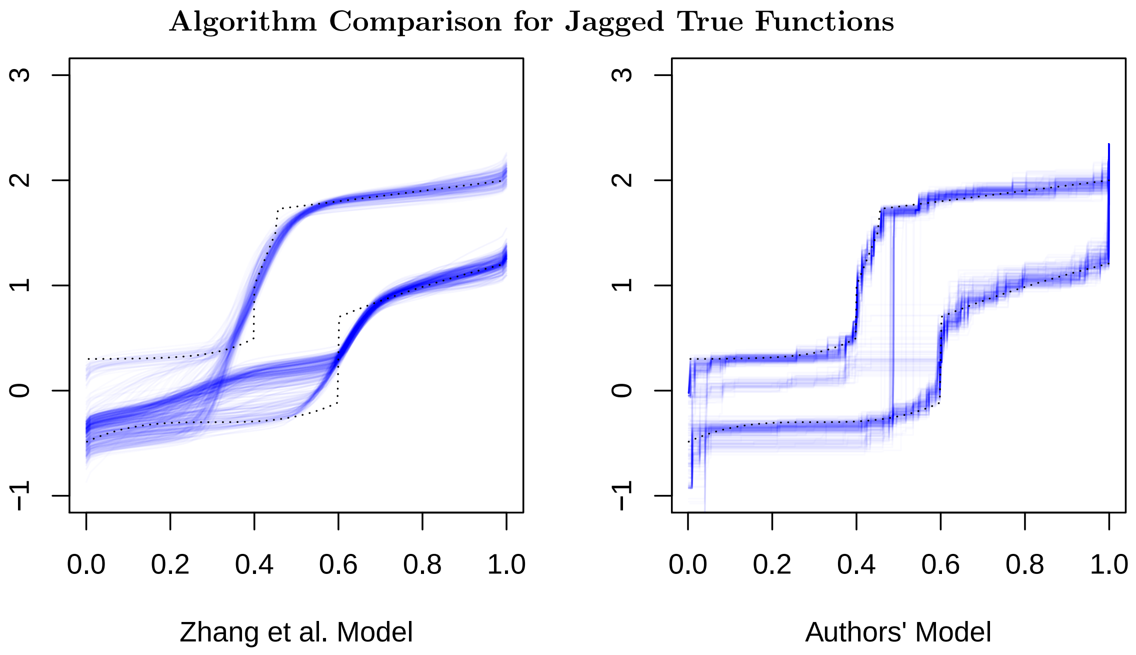

**Figure 5.7:** The Zhang et al. model and the authors' model, fit to data generated from an underlying model with constant true priors and variances, as well as "jagged" underlying true functions.

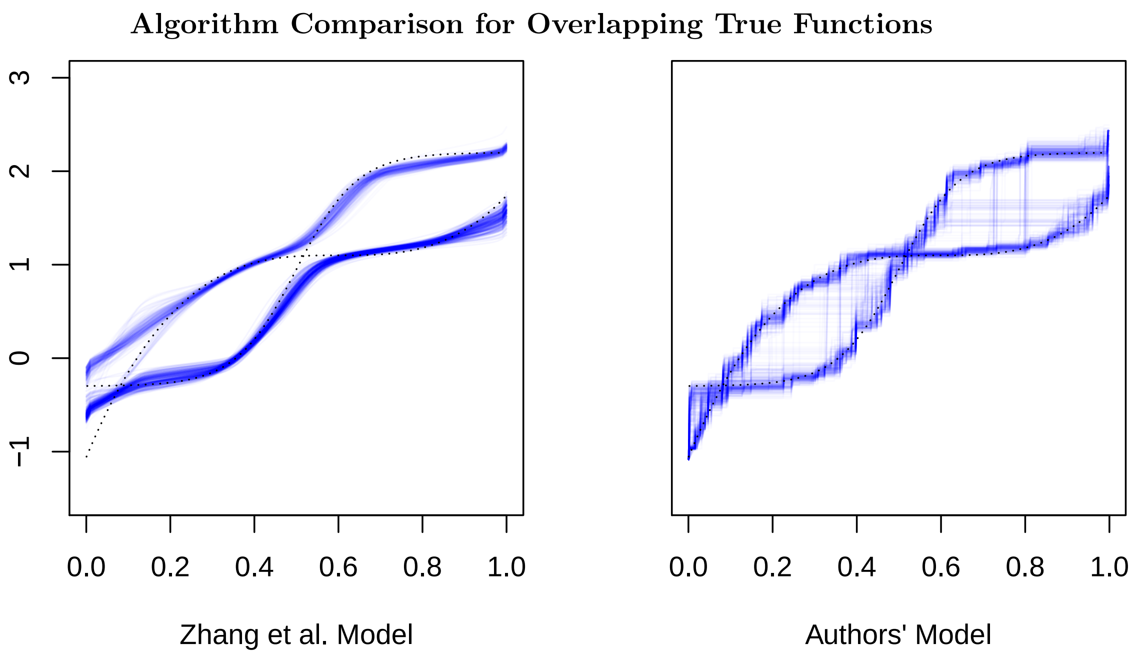

**Figure 5.8:** The Zhang et al. model and the authors' model, fit to data generated from an underlying model with constant true priors and variances, as well as intersecting underlying true functions.

### 5.0.3 Real Data: Global Life Expectancy

In this section, we apply the mixture of monotone regressions to global life expectancy data. Consider data on 'GDP per person' and 'Life expectancy' for all countries between the years 1960 and 2018, drawn from the free online resources of the World Bank (Bank (2018)). Specifically, the data consists of $n$ observations $(y_1, \vec{x_1}), ..., (y_n, \vec{x_n})$, where $Y$ represents Life Expectancy and the vector $\vec{X}$ represents both GDP and Year.

Moreover, this data has two properties that are very common in real world data:

1. Missing Data: Not all countries have data for all years, and several have gaps due to years of conflict in which data was not collected.

2. *A priori* Groupings: This data contains multiple observations per country, but we expect that our model will constrain countries to be clustered together.

On a first pass visualization of this data, we find a mostly linear relationship between Life Expectancy & Year (figure [5.9]), and a mostly logarithmic relationship between Life Expectancy & GDP (figure [5.10]).

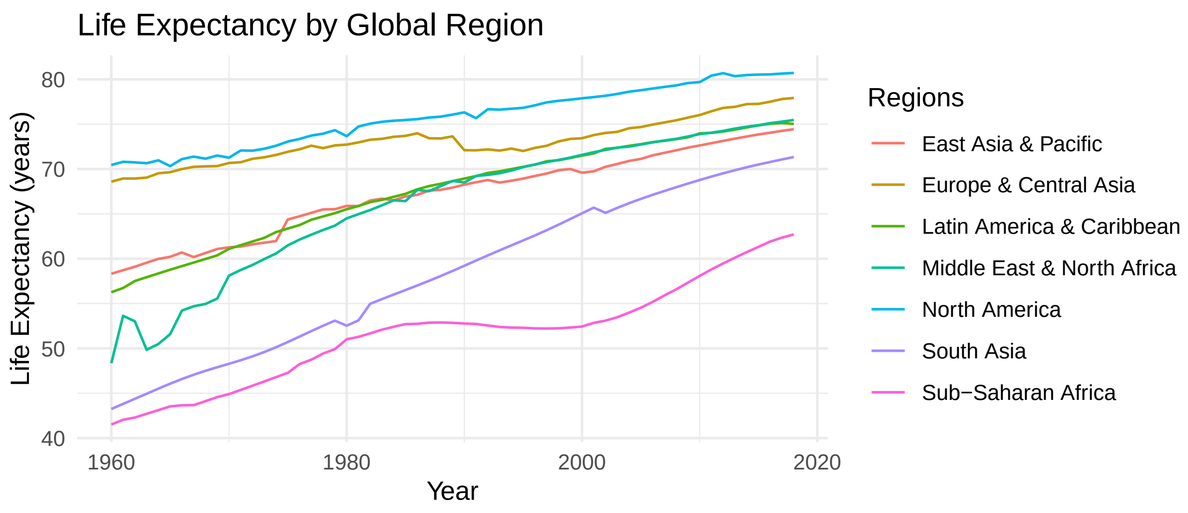

**Figure 5.9:** The relationships between Life Expectancy and time are largely linear across all global regions. Sub-saharan Africa uniquely demonstrates what appears to be a cubic relationship over time.

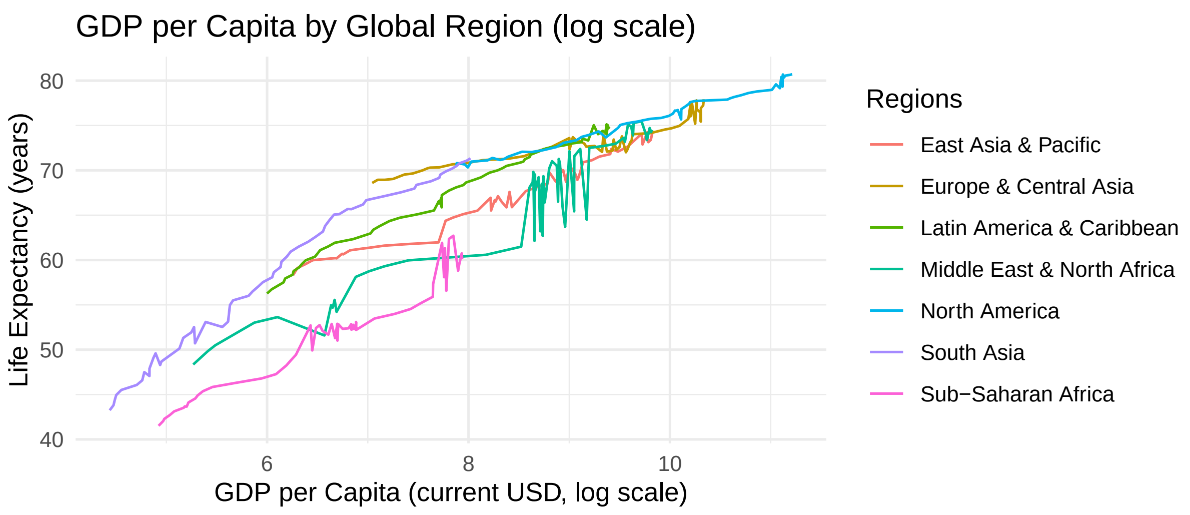

**Figure 5.10:** The relationships between Life Expectancy and GDP per capita are largely log-linear across global regions. Nearly all regions feature brief noisy sections surrounded by notably larger smooth sections.

For the purposes of demonstration, we choose to model this data without log-transforming the GDP data in order to preserve its highly non-linear relationship with Life Expectancy. We proceed by building two step models: with and without an intercept, and each with 'GDP' as the monotone covariate. Each model is in fact a series of 27 mixture models, 3 for each of $k = 1, ..., 9$. These models have the form of equations [5.10] and [5.11] respectively.

$$Y = \sum_{k=1}^{K} \pi_k(g_k(GDP) \ + \ \beta_k \cdot Year \ + \ \epsilon_k) \tag{5.10}$$

$$Y = \sum_{k=1}^{K} \pi_k(g_k(GDP) \ + \ \beta_{0,k} \ + \ \beta_{1,k} \cdot Year \ + \ \epsilon_k) \tag{5.11}$$

For each series, we plot the AIC and BIC per $k$, the rootogram of the model with the lowest AIC, and the fitted monotone functions for the model with the lowest AIC. In the plots of fitted monotone functions, since the $Y$-value only represents the additive contribution of the monotone term, the range of the $Y$ axis spans across 0 and into negative real numbers.

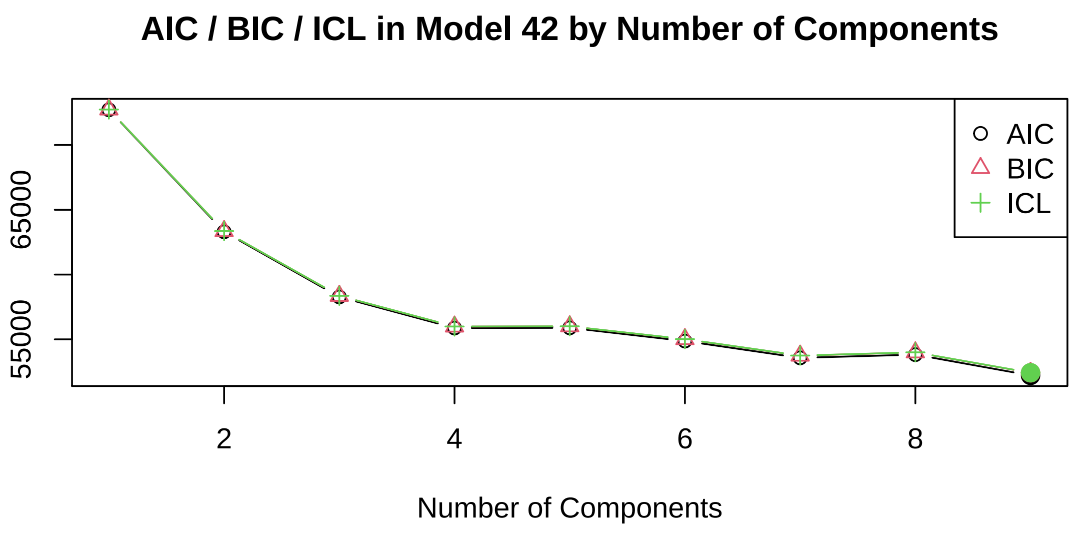

**Figure 5.11:** Here we observe the various model selection metrics – AIC, BIC and ICL – for models represented by equation [5.10], constructed with each of $k$ components.

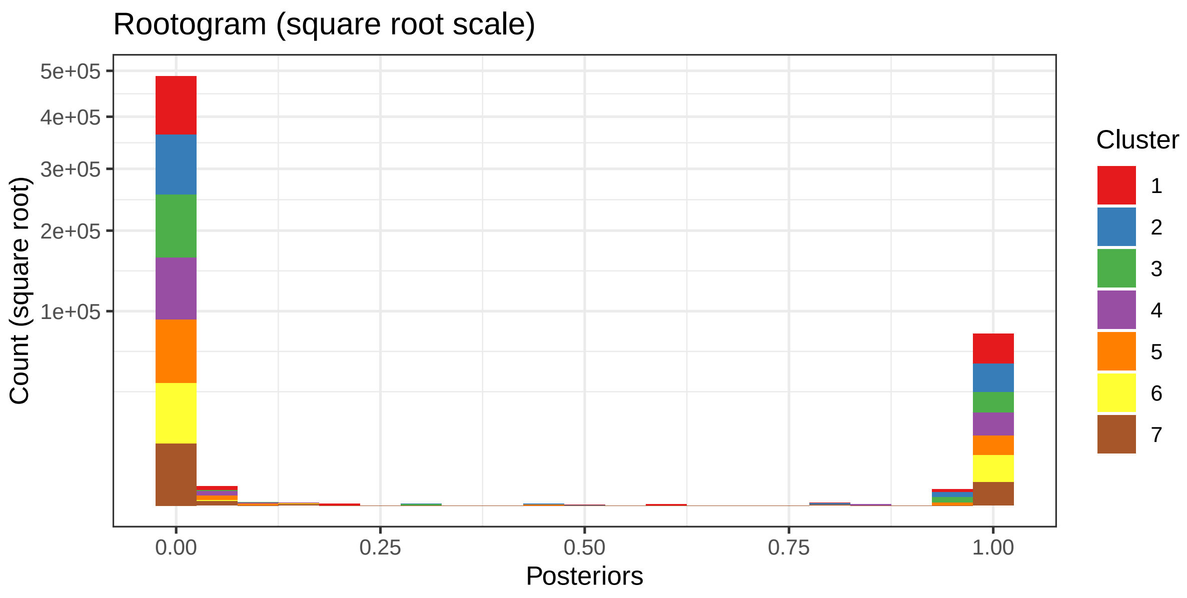

**Figure 5.12:** Here we observe the rootogram of model [5.10] with the lowest AIC. The high proportion of observations classed near the outer limits of 0 and 1 indicate a well-separated model.

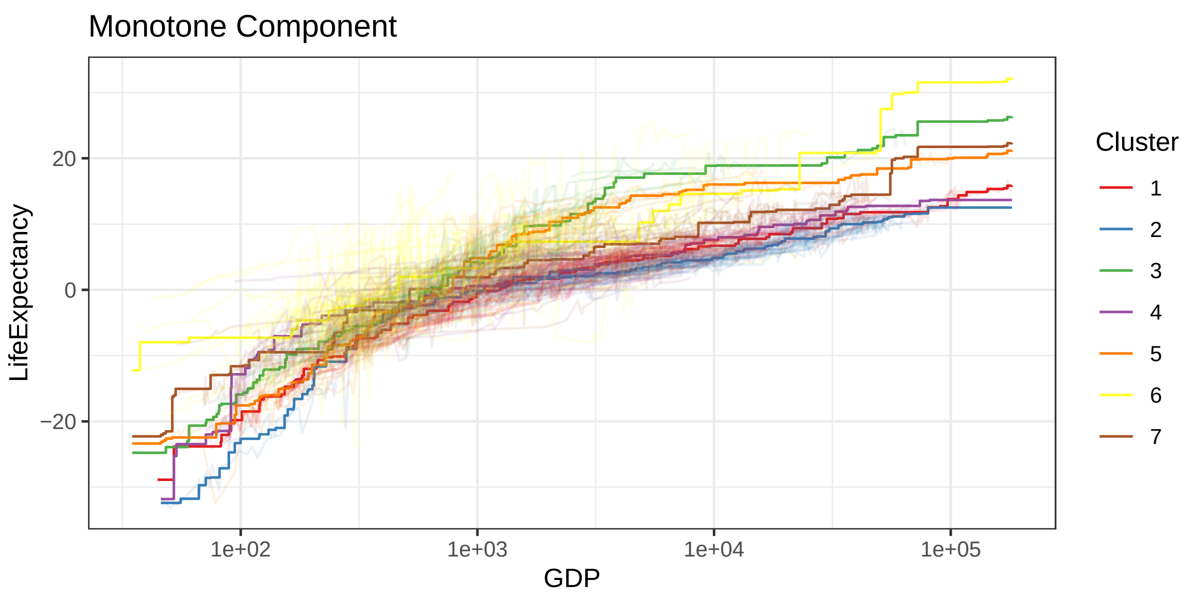

**Figure 5.13:** Here we observe the monotone functions for all components in the fitted model [5.10] with lowest AIC. Observed values are overlaid in the colour of the component to which each country is assigned.

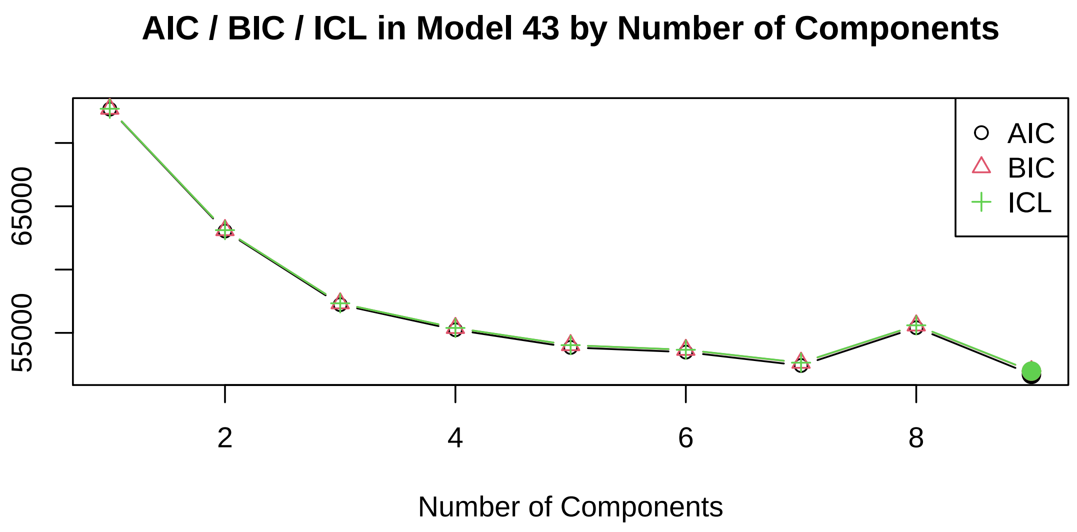

**Figure 5.14:** Here we observe the various model selection metrics – AIC, BIC and ICL – for models represented by equation [5.11], constructed with each of $k$ components.

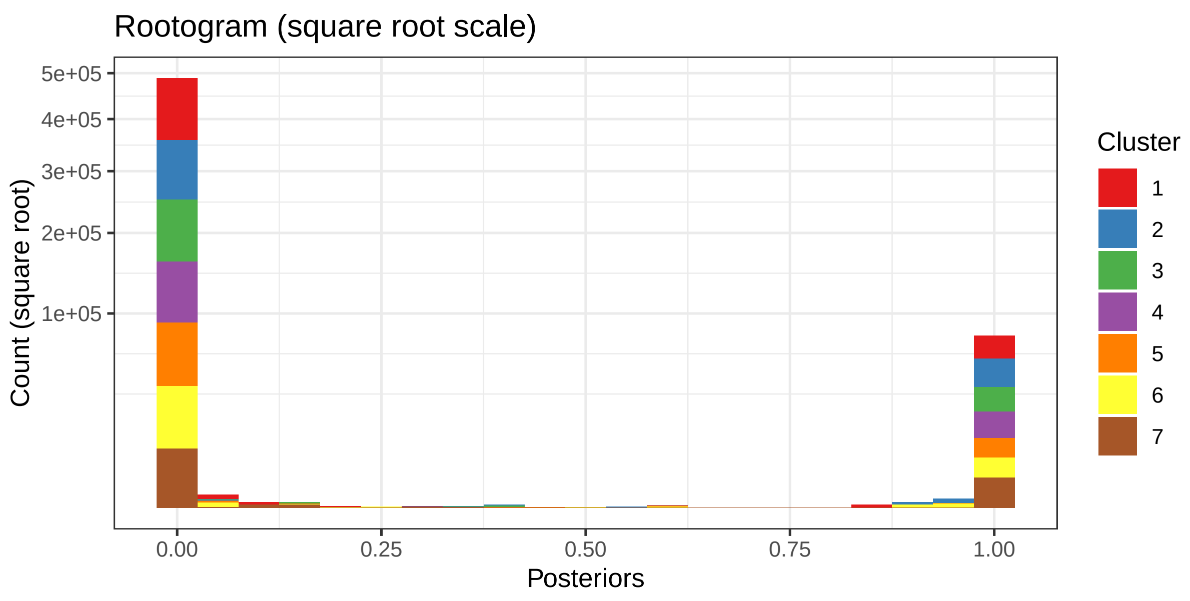

**Figure 5.15:** Here we observe the rootogram of model [5.11] with the lowest AIC. Again, the distribution of the rootogram indicates a well-separated model.

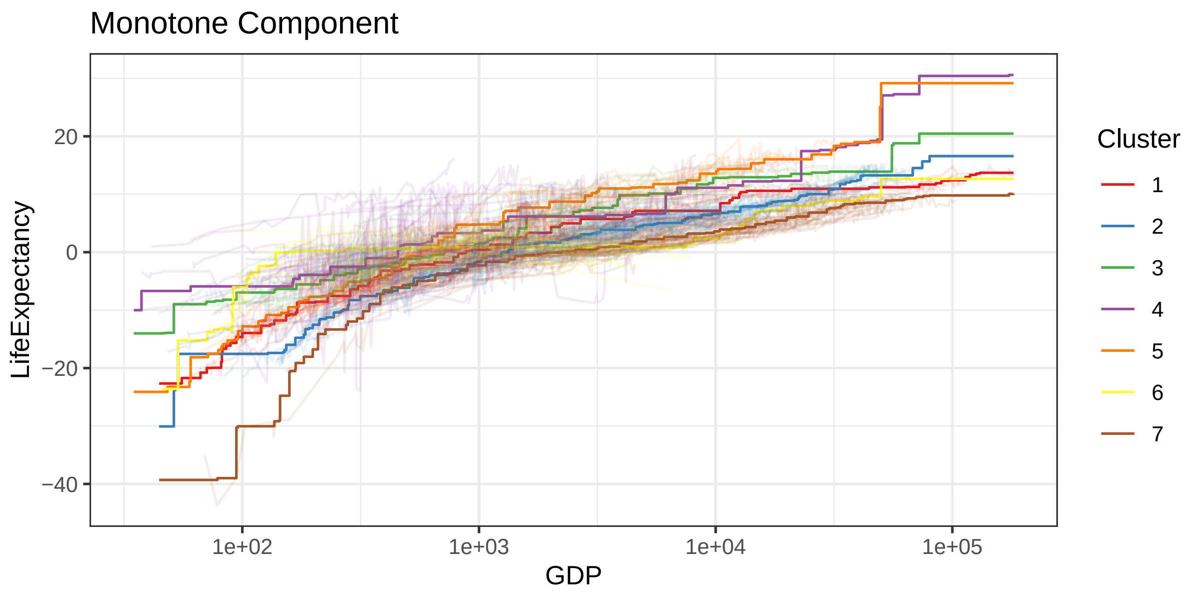

**Figure 5.16:** Here we observe the monotone functions for all components in the fitted model [5.11] with lowest AIC. Observed values are overlaid in the colour of the component to which each country is assigned.

Next, we would typically plot the distribution of clusters within the lowest-AIC model of each step-model series, projected onto a world map. For demonstration purposes, we instead plot the 4-component model from each series. In these world map plots, the colors of each cluster span a spectrum from white to full-color, representing the strength of the posterior and the confidence of the model in placing a given country within a given cluster. The world-maps indicate what the rootograms had previously indicated, namely that the resulting models are extremely confident about the clustering of nearly all countries.

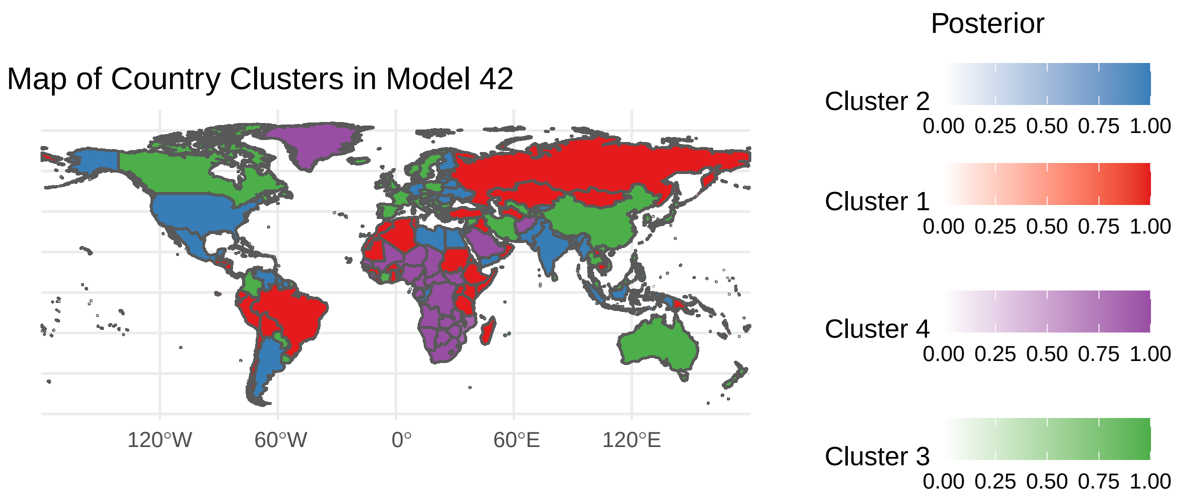

**Figure 5.17:** Here we observe the world map with colour code corresponding to the results of model [5.10].

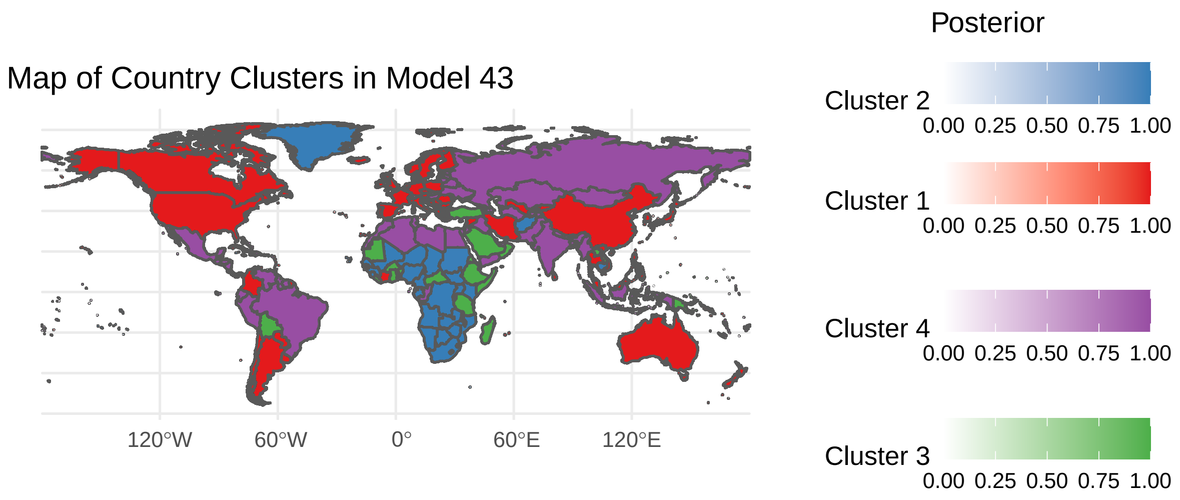

**Figure 5.18:** Here we observe the world map with colour code corresponding to the results of model [5.11].

Finally, we iteratively refit model [5.10] while excluding all observations from a series of countries – Argentina, Canada, Iran, Senegal, Switzerland, Thailand, chosen to represent a variety of countries from different regions – in order to demonstrate the predictive capacities of the mixture model. As stated previously, in section [3.0.1], there are two possible contexts in which one might use the mixture model predictively. One may have a complete observation or set of observations, e.g., the `Year`, `GDP` and `Life Expectancy` data for a given country over several years, in which case one could use the mixture model to generate a set of posterior probabilities representing the probability of that country belonging to each of the model clusters. Alternately, one may have an incomplete observation or set of observations, e.g., the `Year` and `GDP` data (but not `Life Expectancy`) for a country over several years, in which case one could use the mixture model to generate a conditional marginal distribution of `Life Expectancy` for the given country. In figure [5.19], we demonstrate the latter.

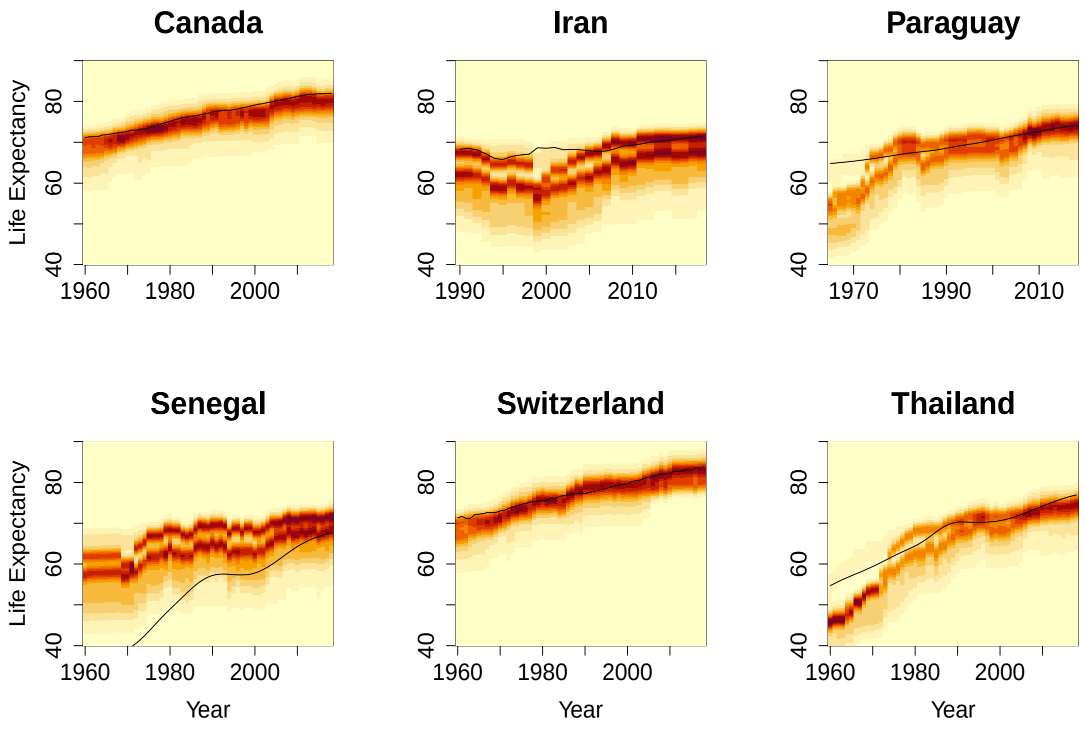

**Figure 5.19:** Marginal distributions of Life Expectancy over time for each country, corresponding to the results of model [5.10]. Observed Life Expectancy values are superimposed in black.

The marginal distributions produced by model [5.10] illustrate the flexibility of the model, and at the same time highlight some of its crucial assumptions. First, the *must-link* constraints, which form the groupings of the model and ensure that all observations from a single country are clustered together, assume rather minimally that observations from a single country are *i.i.d.* and drawn from some common underlying distribution. In this case, that is an untenable assumption since observations were in fact drawn sequentially, and one should therefore assume autocorrelation among the observations, as with any time-series data.

Second, as discussed in section [3.0.1], the model is constructed such that the posterior probabilities of an observation ($\boldsymbol{\Lambda}_i$ for some $i$, to keep with earlier notation) depend on covariates $\boldsymbol{X}_i$ only through dependent variable $Y_i$. The simplicity of this dependence structure is advantageous, but for the predictive task of determining a marginal distribution of $Y$ for an observed $\boldsymbol{X} = \boldsymbol{x}$, the resulting distribution is a mixture proportionate only to the uninformed priors. Thus, although in model [5.10] countries are clearly being clustered with respect to economic development and therefore, in some way, with respect to GDP per capita, the marginal distributions of Life Expectancy for a wealthy country such as Switzerland and a poorer country such as Senegal will have the same mixing proportions regardless of observed GDP per capita. A clear solution to this would be to model the priors $\boldsymbol{\pi}$ of the mixture model as additionally dependent on covariates $\boldsymbol{X}$, as is typically done in mixture-of-expert models (c.f., section [3.0.1]).

# Chapter 6

# Discussion

The model presented in this paper provides a general and robust option for representing mixtures of regressions where some, or all, of the regression terms are non-parametric, monotone functions. There are several advantages to this implementation which make it simpler and more versatile. First, this model generalizes univariate mixtures of monotone regressions to the multivariate case, where any finite number of both linear and monotone terms can be specified within each mixture component. Second, this estimator is free of any tuning parameters in the component regression functions, which allows the overall model to be computationally simpler and leaves fewer modelling decision to the user. Finally, this estimator fits each monotone term in its mixture components in a single step, thereby avoiding the bias inherent in a two-step process, and giving the model flexibility in fitting data for which the true underlying functions are jagged and/or intersect. This flexibility is especially important for modelling in contexts where assuming that the model components are well separated or divisible by a hyperplane is unrealistic.

### 6.0.1 Weaknesses

Although the model presented in this paper makes some substantial improvements on its predecessor, the Zhang et al. model, these improvements are conditional upon the data being modelled. The Zhang et al. model still has lower variance in contexts where the true underlying components have sufficiently smooth monotone terms and are sufficiently well-separated. When these are reasonable assumptions to make, the Zhang et al. model is clearly preferable. Moreover, given that users may have little reason to believe that these assumptions are or are not reasonable, deciding which model to use may be difficult.

Another weakness of our model is that the monotone component of each regression is not a smooth function but rather a step-wise function. For cases in which a smooth function is required, this is prohibitive. However, where smooth monotone functions are not required, the advantages of our model may outweigh this cost.

### 6.0.2 Future Developments

There are three clear directions for future development with respect to the model presented in this paper. First, the model could be generalized to accept alternate types of shape constraints – for example, convex or $k$-monotone constraints. This would greatly extend the model's applicability, especially in cases where model components might combine nonparametric terms with different types of shape constraint.

Second, the model could be generalized to fit regressions with non-normal conditional errors. Since mixtures of generalized linear models have previously been implemented (see, for example, Grün and Leisch (2008)) and the MLE isotonic estimator has already been generalized to non-normal errors (see De Leeuw et al. (2009)), we expect that this would be a simple practical extension. The clearest advantage to such an extension would be the ability to model mixtures of partially monotone logistic regressions. Such models would have ready applications in, for example, epidemiological studies like Morton-Jones et al. (2000) where the outcomes are binary disease states instead of continuous disease severities.

Third, the model could be generalized to fit mixtures with coefficient-dependent priors, as do the model of Zhang et al. and other variations of the mixture-of-experts model. Although such models do not necessarily have interpretable prior parameters, they are much better suited to applications that place high importance on prediction.

Finally, an exact, finite-sample approach in the style of Gao et al. (2020) could be developed for the inference of confidence intervals. The successful application of such an approach would solve the label-switching problem explored in section [4.0.4] without adding bias to the parameter estimates.

# Appendix A

# Appendix

## A.1 Asymptotic Behaviour of Partially Linear Models with Monotone Constraints

As discussed in section [3.0.2], the asymptotic behaviour of partially linear models with monotone constraints described by Cheng (2009) requires that the assumptions below hold.

Let the estimators $\hat{\beta}_n, \hat{h}_1, ..., \hat{h}_J$ be defined as the minimizer of

$$S_n(\beta, h_1, ..., h_j) \;=\; n^{-1}\sum_{i=1}^{n}(Y_i - X_i'\beta - \sum_{j=1}^{J} h_j(W_{ij}))^2 \tag{A.1}$$

where each $h_j$ is a monotone function with bounded derivative and $\int h_j(w_j)dw_j = 0$. Then, the following are regularity conditions for $1 \leq j \leq J$:

1. The function $h_j$ satisfies the condition that

$$\inf_{|w_j - w_j'| \geq \delta} |h_j(w_j) - h_j(w_j')| \;\geq\; C_1\delta^\gamma \tag{A.2}$$

   for any $\delta > 0$ and some constants $C_1, \gamma > 0$.

2. The density for $W_j$, denoted as $p_{w_j}$, is assumed to be bounded away from zero and infinity, and fulfills the below Lipschitz condition

$$\sup_{-1 \leq w_j, w_j' \leq 1} |p_{w_j}(w_j) - p_{w_j}(w_j')| \;\leq\; M|w_j - w_j'|^\rho \tag{A.3}$$

   for some constants $M, \rho > 0$.

3. The function $\zeta_j(w) \equiv E(X|W_j = w)$ satisfies the condition

$$||\zeta_j(w_j) - \zeta_j(w_j')|| \leq \; C_2|w_j - w_j'| \tag{A.4}$$

   for some constant $C_2 > 0$.

4. $E(X - \sum_{j=1}^{J} E(X|W_j))^{\otimes 2}$ is strictly positive.

## A.2 Flexmix Extension

All results in this paper were produced by an extension of Flexmix, a flexible implementation of generalized finite mixture models created by Bettina Grun and Friedrich Leisch (Leisch (2004)). The package provides a framework for implementing specific types of mixture models based on a central, universal framework. It is thus intended to allow users to elaborate a specific estimator or "driver" for the modeling of mixture components, while the more abstract behaviours are managed by the Flexmix backend. More specifically, Flexmix implements:

1. The universal functions of the EM algorithm for assigning prior values $\pi_k$ to each of the mixture components and posterior values $\Lambda_i$ to each of the observations upon each iteration of the EM algorithm;
2. Threshold constants for the convergence of the EM algorithm;
3. Restrictions on the model estimate, e.g., restricting components to have an estimated prior above a certain threshhold;
4. Ordinary- and parametric-bootstrap methods for the estimation of confidence intervals with respect to each component.

Conversely, the user must provide a model estimator and a model object with a log-likelihood method, `logLik()`, which returns the density of an observation given a component's parameters, and a prediction function, `predict()`, which gives an expected value of an observation's dependent variable given the observed independent variables.

To implement the Flexmix extension, one must call `flexmix()` from the Flexmix package with `mono_reg()` as the model argument. E.g., the following call to `flexmix()` produces a model with 6 components, each of which is a partial linear model regressing `Y` on all other variables of `df`, excluding an intercept. The `mon_inc_index` argument to `mono_reg()` instructs the function to estimate an isotonic function on the second independent variable in `df`.

```r
mod <- flexmix(Y ~ .-1, data = df, k = 6, model =
                       mono_reg(mon_inc_index = 2))
```

It should be noted that the indexing arguments passed to `mono_reg()` are indices of the design matrix constructed by the formula passed to `flexmix()`, so they change based on the design matrix. For example, without an intercept, `mono_inc_index = 2` refers to the 2nd independent variable of the data frame; when an intercept *is* included, `mono_inc_index = 2` refers to the 1st independent variable of the data frame `df`. It is therefore safer and recommended that the user specify the monotone terms by name using the `mono_inc_names` or `mono_dec_names` arguments, as below.

```
mod <- flexmix(Y ~ ., data = df, k = 6, model =
                        mono_reg(mon_inc_names = "Var_2"))
```

The extension allows the user to include any number of isotonic and/or antitonic (monotone-non-decreasing, monotone-non-increasing) variables in the call to `mono_reg`, as demonstrated below.

```
mod <- flexmix(Y ~ ., data = df, k = 6, model =
                        mono_reg(mon_inc_names = c("Var_2", "Var_4"),
                                 mon_dec_names = c("Var_3", "Var_6")))
```

Flexmix also allows the construction of multiple mixture models within a single object. The following call to `stepFlexmix()` builds 25 mixture models. For each of $k$ equal to 1 through 5 components, `nrep` specifies the construction of 5 models. Each model contains 2 monotone components – a monotone-non-decreasing, or isotonic, relationship between `Y` and `Var_2`, and a monotone-non-increasing, or antitonic, relationship between `Y` and `Var_3`.

```
m2 <- stepFlexmix(Y ~ ., data = df, model =
                       mono_reg(mon_inc_names = "Var_2",
                                mon_dec_names = "Var_3"), k = 1:5, nrep = 5)
```

A notable weakness in the Flexmix architecture is that the structure of all components must be specified identically within a given model. Thus, whereas in theory one could fit a mixture of monotone regressions where different components have different monotone directions – e.g., the first component is monotone non-decreasing and the second component is monotone non-increasing – this is not possible with the current instantiation of Flexmix.

For further discussion of the use of the Flexmix package, see the guides at https://ro.uow.edu.au/cgi/viewcontent.cgi?article=3410&context=commpapers and https://cran.rapporter.net/web/packages/flexmix/vignettes/mixture-regressions.pdf.

The complete code for the extension of Flexmix for modelling mixtures of partially-linear monotone regressions is included below. The first code block implements the partial linear model with monotone shape constraints; the second code block integrates the partial linear model with the Flexmix framework; the third code block provides additional functions for the visualization of results.

### A.2.1 Code for the Estimation of Partial Linear Models

```
# define estimator for partial linear model with arbitrary monoto-
# ne-constrained component
```

```
cpav <- function(x_mat, y, weights, inc_index=NULL, dec_index=NULL
                 , max_iters_cpav=NULL, max_delta_cpav=NULL){

  joint_ind <- c(inc_index, dec_index)

  if(!is.matrix(x_mat)) stop("x_mat is not of class matrix, and
                               will be rejected by lm.wfit")
  if(any(weights == 0)) stop("monoreg(), and therefore cpav(),
                               cannot take weights of 0!")
  if(length(y) != length(weights) | length(y) !=
    dim(x_mat)[1]) stop("The dimension of the inputs is not
                                           equal to the dimension
                                           of the weights")

  # if there is only 1 monotone component, apply ordinary
  # monotone regression
  if(length(joint_ind) == 1){
    if(length(inc_index) == 1){ # the component is monotone
      # increasing
      return( # cast the monoreg object as a matrix, with all
        #attributes as rows in the first column
        matrix(suppressWarnings(monoreg(x = x_mat[,inc_index],
                                        y = y, w = weights)),
               dimnames = list(c("x", "y", "w", "yf", "type",
                                 "call")))
      )}
    else{ # the component is monotone decreasing
      return( # cast the monoreg object as a matrix, with all
        #attributes as rows in the first column

        matrix(suppressWarnings(monoreg(x = x_mat[,dec_index],
                                        y = y, w = weights, type =
                                          "antitonic")),
               dimnames = list(c("x", "y", "w", "yf", "type",
                                 "call")))
      ) } }

  else{ # the monotone components are multiple, so continue with
    # cyclic algorithm
```

```r
# fit ordinary lm on x_mat and y
start_betas <- coef(lm.wfit(x=x_mat[,joint_ind], y=y,
                            w=weights))

# set initial monotone reg estimates by calling each monoreg()
#against y - lm.predict(all other vars)

mr_fits <- sapply(1:length(joint_ind), function(i)
  if(joint_ind[i] %in% inc_index){
    # i apologize to anyone trying to read this line,
    # but think: the columns of the x_matrix
    # indicated by join_ind, except the value of joint_ind at
    # the ith place in joint_ind
    suppressWarnings(monoreg(x = x_mat[,joint_ind[i]],
            y = (y - (as.matrix(x_mat[,joint_ind[-i]]) %*%
                        start_betas[-i]) ), w = weights,
            type = "isotonic"))
    }
  else if(joint_ind[i] %in% dec_index){
    suppressWarnings(monoreg(x = x_mat[,joint_ind[i]],
            y = (y - (as.matrix(x_mat[,joint_ind[-i]]) %*%
                        start_betas[-i]) ), w = weights,
            type = "antitonic"))
  })

# iterate through mr_fits. each column of mr_fits (e.g.,
# mr_fits[,1]) is a monoreg fitted object,
# and its attributes can be called (e.g., mr_fits[,1]$yf)
iters <- 0
delta <- 0.5
if(is.null(max_iters_cpav)){
  max_iters_cpav <- 100
}
if(is.null(max_delta_cpav)){
  max_delta_cpav <- 0.00001}

while(abs(delta) > max_delta_cpav & iters < max_iters_cpav){
  old_SS <- mean((y - get_pred(mr_fits, x_mat[,joint_ind]))^2)
```

```
      for(i in 1:length(joint_ind)){
        if(joint_ind[i] %in% inc_index){
          # i apologize to anyone trying to read this line, but
          # think: the columns of the x_matrix
          # indicated by join_ind, except the value of joint_ind
          # at the ith place in joint_ind
          mr_fits[,i] <- suppressWarnings(monoreg(x =
                                             x_mat[,joint_ind[i]],
                                      y = (y - get_pred(mr_fits[,-i],
                                             x_mat[,joint_ind[-i]])),
                                      w = weights, type = "isotonic"))}
        else if(joint_ind[i] %in% dec_index){
          mr_fits[,i] <- suppressWarnings(monoreg(x =
                                             x_mat[,joint_ind[i]],
                                      y = (y - get_pred(mr_fits[,-i],
                                             x_mat[,joint_ind[-i]])),
                                      w = weights, type = "antitonic"))}}

      new_SS <- mean((y - get_pred(mr_fits, x_mat[,joint_ind]))^2)
      delta <- (old_SS - new_SS)/old_SS

      iters <- iters + 1 }

    return(mr_fits)
  }}

# first, define function for obtaining f(x_new) for monotone
# regression f()
# get_pred returns a vector of length = nrows(xvals), ie, a value
# for each observation of xvals
get_pred <- function(mr_obj, xvals){
  xvals <- as.matrix(xvals)
  mr_obj <- as.matrix(mr_obj)

  if(dim(mr_obj)[2] != dim(xvals)[2]) stop("get_pred() must take
  an X-matrix with as many columns
                                             as monoreg() objects")
  if(dim(xvals)[1] == 1){ # if xvals is a single observation
    sapply(1:ncol(xvals), function(j)
      mr_obj[,j]$yf[sapply(xvals[,j], function(z)
        ifelse( z < mr_obj[,j]$x[1], 1,
                ifelse(z >= tail(mr_obj[,j]$x, n=1),
                       length(mr_obj[,j]$x),
                       which.min(mr_obj[,j]$x <= z)-1 )))])}
  else{
    apply(sapply(1:ncol(xvals), function(j)
         mr_obj[,j]$yf[sapply(xvals[,j], function(z)
           ifelse( z < mr_obj[,j]$x[1], 1,
            ifelse(z >= tail(mr_obj[,j]$x, n=1),
                   length(mr_obj[,j]$x),
                   which.min(mr_obj[,j]$x <= z)-1 )))]
         ), 1, function(h) sum(h))}}

# define partial linear regression of y on x with weights w
# inputs are: x, y, wates, mon_inc_index, mon_dec_index, max_iter
part_fit <- function(x, y, wates = NULL, mon_inc_index=NULL,
                     mon_dec_index=NULL, max_iter=NULL,
                     component = NULL, na.rm=T,
                     mon_inc_names = NULL,
                     mon_dec_names = NULL, start_fit = NULL, ...){

  # cast x to matrix
  x <- as.matrix(x)

  # set default weights
  if(is.null(wates)) wates <- rep(1, length(y))

  # remove incomplete cases
  if(T){ # for now, there is no alternative to na.rm=T.  All
    # incomplete cases are removed.
    cc <- complete.cases(y) & complete.cases(x) &
      complete.cases(wates)
    y <- y[cc]
    x <- x[cc,, drop=FALSE]
    wates <- wates[cc]
    cc <- NULL}

  x <- as.matrix(x) # cast again. hacky but apparently necessary
```

```r
# make sure y and wates is not multivariate
if(length(y) != dim(x)[1] | length(y) != length(wates)) stop(
    "Inputs are not of the same dimension!")

# take monotone indices of previous component
if(!is.null(component)){
  inc_ind <- component$mon_inc_index
  dec_ind <- component$mon_dec_index}
else{
  if(is.null(mon_inc_index) & is.null(mon_dec_index) &
      is.null(mon_inc_names) & is.null(mon_dec_names)){
    stop("Some monotone index or name must be specified")
  }
  if(!is.null(mon_inc_names)){ # add names to index list
    mon_inc_index <- unique(c(mon_inc_index, which(colnames(x)
                                     %in% mon_inc_names)))
  }
  if(!is.null(mon_dec_names)){
    mon_dec_index <- unique(c(mon_dec_index, which(colnames(x)
                                     %in% mon_dec_names)))
  }
  # transfer inc_names to inc_index
    inc_ind <- mon_inc_index
    dec_ind <- mon_dec_index}

# throw warning if there are duplicates in inc_ind or dec_ind,
# and then remove
if(anyDuplicated(inc_ind) | anyDuplicated(dec_ind)){
  warning("There are duplicate index instructions; Duplicates
          are being removed.")
  inc_ind <- unique(inc_ind)
  dec_ind <- unique(dec_ind)}

# throw error if indices overlap
if(length(intersect(inc_ind, dec_ind)) > 0) stop("At least one
variable was marked as BOTH
                                              monotone increasing
and monotone decreasing.")
```

```
# throw error if indices are not integers
if(!is.null(inc_ind)){
  if(any(inc_ind != as.integer(inc_ind))) stop("Monotone
                    increasing indices are not integers.")
}
if(!is.null(dec_ind)){
  if(any(dec_ind != as.integer(dec_ind))) stop("Monotone
                    decreasing indices are not integers.")}

# throw error if indices are not positive
if(any(c(inc_ind, dec_ind) < 1)) stop("all monotone component
                                      indices must be positive")

# throw error if the number of indices exceeds columns of x
if(length(c(inc_ind, dec_ind)) > ncol(x)) stop("Number of
    proposed monotonic relationships exceeds columns of x.")

# If there is an intercept but no other linear effects, stop
if((length(c(inc_ind, dec_ind))+1) == ncol(x) & "(Intercept)"
   %in% colnames(x)){
  stop("For identifiability purposes, you cannot build a
    part_fit with only an intercept as a linear component.")}

# If start_fit is specified, make sure it has only part_fit
# elements with the appropriate dimensions
if(!is.null(start_fit)){

  if(!is(start_fit, "list")) stop("start_fit must be a list of
                                  part_fit attributes")
  if("coef" %in% names(start_fit)){
    if(length(start_fit$coef) != (dim(x)[2] - length(c(inc_ind,
                                              dec_ind))) ){
      stop("Not the right number of coefficients in
           starting values")
    }
    if(!all(names(start_fit$coef) %in% colnames(x)[-c(inc_ind,
                                              dec_ind)]) ){
      stop("Some coefficient(s) in starting values have
           incorrect names")
    }
    if(!all(colnames(x)[-c(inc_ind, dec_ind)] %in%
            names(start_fit$coef)) ){
      stop("Some coefficient(s) in starting values are missing")
    }
    if(!all(names(start_fit$coef) == colnames(x)[-c(inc_ind,
                                                dec_ind)])){
      start_fit$coef <- start_fit$coef[
        match(colnames(x)[-c(inc_ind,
                dec_ind)],start_fit$coef)] }}}

# option for fit with no linear independent components and one
# or multiple monotone components:
if(length(c(inc_ind, dec_ind)) == ncol(x)){

  yhat <- cpav(x_mat = as.matrix(x[wates != 0,]), y =
                 y[wates != 0],
               weights = wates[wates != 0],
               inc_index = inc_ind, dec_index = dec_ind)

  # get residuals of model
  resids <- y - get_pred(yhat, x[,c(inc_ind, dec_ind)])

  # mod must have: coef attribute, sigma attribute, cov
  # attribute, df attribute, ..., and
  # may have mon_inc_index and mon_dec_index attributes
  mod <- list(coef = NULL, fitted_pava = NULL, sigma = NULL,
              df = NULL, mon_inc_index = NULL,
              mon_dec_index = NULL,
              iterations = NULL,
              mon_inc_names = NULL, mon_dec_names = NULL)

  mod$coef <- NULL
  mod$fitted_pava <- yhat
  mod$mon_inc_index <- inc_ind
  mod$mon_dec_index <- dec_ind
  mod$sigma <- sqrt(sum(wates * (resids)^2 /
                            mean(wates))/ (nrow(x)-qr(x)$rank))
  mod$df <- ncol(x)+1
  class(mod) <- "part_fit"

  return(mod)
}
else{
  # for starting values, fit a regular lm or use starting values
  if(!is.null(start_fit) && "coef" %in% names(start_fit)){
    betas <- start_fit$coef
  }
  else{
    fit <- lm.wfit(x=x, y=y, w=wates)
    betas <- coef(fit)[-c(inc_ind, dec_ind)]}

  # set maximum iterations for convergence
  if(!is.null(max_iter) & !is.list(max_iter)){
    if(max_iter < 1) stop("max_iter must be positive")
    maxiter <- max_iter
  }
  else{
    maxiter <- 200 }

  # set while loop initial values
  iter <- 0
  delta <- 10
  # iterate between pava and linear model
  # set while loop condition(s).
  # delta works well enough with delta > 1e-12.
  while(delta > 1e-6 & iter < maxiter){

    yhat <- cpav(x_mat = as.matrix(x[wates != 0,]), y =
                   (y[wates != 0] - (as.matrix(x[wates != 0,
                            -c(inc_ind, dec_ind)]) %*% betas)),
                 weights = wates[wates != 0], inc_index =
                   inc_ind, dec_index = dec_ind)

    # save old betas for distance calculation
    old_betas <- betas
    # to retrieve old ordering of y for fitted values, we use
    # y[match(x, sorted_x)]
    betas <- coef(lm.wfit(x=as.matrix(x[,-c(inc_ind, dec_ind)]),
                      y= (y - get_pred(yhat,
                         x[,c(inc_ind, dec_ind)]) ), w=wates))

    # quantify change in yhat vals and beta vals
    # get euclidian distance between betas
    # transformed into unit vectors
    delta <- dist(rbind( as.vector(betas)/norm(as.vector(betas),
                                          type ="2"),
                as.vector(old_betas)/norm(as.vector(old_betas),
                                          type="2")
                         ))

    iter <- iter + 1     # iterate maxiter
  }
}

# get residuals of model
resids <- y - (get_pred(yhat, x[,c(inc_ind, dec_ind)]) +
              (as.matrix(x[,-c(inc_ind, dec_ind)]) %*% betas))

# mod must have: coef attribute, sigma attribute, cov attribute,
# df attribute, ..., and may have mon_inc_index and
# mon_dec_index attributes
mod <- list(coef = NULL, fitted_pava = NULL, sigma = NULL,
            df = NULL, mon_inc_index = NULL, mon_dec_index =
              NULL, iterations = NULL, mon_inc_names = NULL,
            mon_dec_names = NULL)

mod$coef <- betas
mod$fitted_pava <- yhat
mod$iterations <- iter
mod$mon_inc_index <- inc_ind
mod$mon_dec_index <- dec_ind
mod$sigma <- sqrt(sum(wates * (resids)^2 /
                        mean(wates))/ (nrow(x)-qr(x)$rank))
mod$df <- ncol(x)+1

class(mod) <- "part_fit"
```

```r
  return(mod)
}


# write plot method for objects returned from part_fit()
append_suffix <- function(num){
  suff <- case_when(num %in% c(11,12,13) ~ "th",
                    num %% 10 == 1 ~ 'st',
                    num %% 10 == 2 ~ 'nd',
                    num %% 10 == 3 ~'rd',
                    TRUE ~ "th")
  paste0(num, suff)
}

plot.part_fit <- function(z){
  if(dim(as.matrix(z$fitted_pava))[2] > 1){
    temp <- list()
    for(i in 1:dim(as.matrix(z$fitted_pava))[2]){
      temp[[i]] <- ggplotGrob(ggplot() +
        geom_line(aes(x = z$fitted_pava[,i]$x, y =
                        z$fitted_pava[,i]$yf)) +
        theme_bw() +
        labs(title = paste(append_suffix(i),
                           " Monotone Regression"),
             x = "X",
             y = "Y"))
    }
    return(grid.arrange(grobs=temp, ncol=1))
  }
  else{
    temp <- ggplot() +
      geom_line(aes(x = z$fitted_pava[,1]$x,
                    y = z$fitted_pava[,1]$yf)) +
      theme_bw() +
      labs(title = "Monotone Regression",
           x = "X",
           y = "Y")
    return(temp)
  }
```

```
}
```

### A.2.2 Code for the M-step Driver

```
# allow slots defined for numeric to accept NULL
setClassUnion("numericOrNULL",members=c("numeric", "NULL"))
setClassUnion("characterOrNULL",
              members = c("character", "NULL"))
setOldClass("monoreg")
setClassUnion("matrixOrMonoreg",
              members = c("matrix", "monoreg"))

# Define new classes
setClass(
  "FLX_monoreg_component",
  contains="FLXcomponent",
  # allow mon_index to take either numeric or NULL
  slots=c(mon_inc_index="numericOrNULL",
          mon_dec_index="numericOrNULL",
          mon_obj="matrix",
          mon_inc_names="characterOrNULL",
          mon_dec_names="characterOrNULL"
          ))

# Define FLXM_monoreg
setClass("FLXM_monoreg",
         contains = "FLXM",
         slots = c(mon_inc_index="numericOrNULL",
                   mon_dec_index="numericOrNULL",
                   mon_inc_names="characterOrNULL",
                   mon_dec_names="characterOrNULL"))

# definition of monotone regression model.
mono_reg <- function (formula = .~., mon_inc_names = NULL,
                      mon_dec_names = NULL, mon_inc_index=NULL,
                      mon_dec_index=NULL, ...) {

  # only names or indices can be indicated, not both
  if((!is.null(mon_inc_names)|!is.null(mon_dec_names)) &
    (!is.null(mon_inc_index)|!is.null(mon_dec_index))) stop(
    "mono_reg() can accept either monotone
    names or indices can be chosen, but not both.")

  retval <- new("FLXM_monoreg", weighted = TRUE,
                formula = formula,
                name = "partially linear monotonic regression",
                mon_inc_index= sort(mon_inc_index),
                mon_dec_index= sort(mon_dec_index),
                mon_inc_names= mon_inc_names,
                mon_dec_names= mon_dec_names)

  # @defineComponent: Expression or function constructing the
  # object of class FLXcomponent fit must have attributes:
  # coef, sigma, cov, df, ..., and
  # may have mon_inc_index and mon_dec_index attributes
  # ... all must be defined by fit() function
  retval@defineComponent <- function(fit, ...) {
    # @logLik: A function(x,y) returning the
    # log-likelihood for observations in matrices x and y
                  logLik <- function(x, y) {
                    dnorm(y, mean=predict(x, ...), sd=fit$sigma,
                          log=TRUE)}
                  # @predict: A function(x) predicting y given x.
                  predict <- function(x) {
                    x <- as.matrix(x)
                    inc_ind <- fit$mon_inc_index
                    dec_ind <- fit$mon_dec_index

                    p <-  get_pred(fit$fitted_pava, x[,c(inc_ind,
                                                       dec_ind)])
                    if(!is.null(fit$coef)){
                      p <- p + (as.matrix(x[,-c(inc_ind,
                                          dec_ind)]) %*% fit$coef)
                    }
                    p
                  }
                  # return new FLX_monoreg_component object
                  new("FLX_monoreg_component", parameters =
                        list(coef = fit$coef, sigma = fit$sigma,
                             mon_obj = fit$fitted_pava),
                      df = fit$df, logLik = logLik, predict =
                        predict, mon_inc_index =
                        fit$mon_inc_index, mon_dec_index =
                        fit$mon_dec_index,
                      # mon_obj = fit$fitted_pava,
                      mon_inc_names = fit$mon_inc_names,
                      mon_dec_names = fit$mon_dec_names)
  }

  # @fit: A function(x,y,w) returning an object of
  # class "FLXcomponent"
  retval@fit <- function(x, y, w, component, mon_inc_index =
                           retval@mon_inc_index,
                         mon_dec_index = retval@mon_dec_index,
                         mon_inc_names = retval@mon_inc_names,
                         mon_dec_names = retval@mon_dec_names,
                         ...) {

                  if(is.null(mon_inc_index) &
                     is.null(mon_dec_index)){

                # if not all monotone names are in the design
                # matrix, stop & print the name that is missing
                    if(!all(c(mon_inc_names, mon_dec_names) %in%
                            colnames(x))){
                      stop(paste(setdiff(c(mon_inc_names,
                                    mon_dec_names), colnames(x)),
                                 "could not be found in the model
                                 matrix. Check your spelling."))
                    }
                    # Discover correct monotone indices
                    if(any(colnames(x) %in% mon_inc_names)){
                      mon_inc_index <- which(colnames(x) %in%
                                               mon_inc_names)
                    }
                    if(any(colnames(x) %in% mon_dec_names)){
                      mon_dec_index <- which(colnames(x)
                                            %in% mon_dec_names)
                    }
                  }
                  if(is.null(mon_inc_names) &
                     is.null(mon_dec_names)){
                    # Discover correct monotone names
                    mon_inc_names <- colnames(x)[
                      sort(mon_inc_index)]
                    mon_dec_names <- colnames(x)[
                      sort(mon_dec_index)]
                  }

                  fit <- part_fit(x, y, w, component,
                              mon_inc_index=mon_inc_index,
                              mon_dec_index=mon_dec_index, ...)

                  retval@defineComponent(fit, ...)
                  }
  retval
  }
```

### A.2.3 Code for Wrapper Functions

```
# Wrapper functions for Flexmix objects with monoreg components

# import libraries
library(ggplot2)
library(grid)
library(gridExtra)
library(RColorBrewer)



# Multiple plot function
#
# ggplot objects can be passed in ..., or to plotlist (as a list
# of ggplot objects)
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
  library(grid)
  # Make a list from the ... arguments and plotlist
  plots <- c(list(...), plotlist)

  numPlots = length(plots)

  # If layout is NULL, then use 'cols' to determine layout
  if (is.null(layout)) {
    # Make the panel
    # ncol: Number of columns of plots
    # nrow: Number of rows needed, calculated from # of cols
    layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                     ncol = cols, nrow = ceiling(numPlots/cols))
  }

  if (numPlots==1) {
    print(plots[[1]])

  } else {
    # Set up the page
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(nrow(layout),
                                               ncol(layout))))

    # Make each plot, in the correct location
    for (i in 1:numPlots) {
      # Get the i,j matrix positions of the regions
      # that contain this subplot
      matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

      print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                      layout.pos.col = matchidx$col))
    }
  }
}



####

# overwrite method for plot.flexmix
```

```r
setMethod('plot',  signature(x="flexmix", y="missing"),
          function(x, mark=NULL, markcol=NULL, col=NULL,
                   eps=1e-4, root=TRUE, ylim=NULL, xlim=NULL,
                   main=NULL, xlab=NULL, ylab=NULL,
                   as.table = TRUE, endpoints = c(-0.04, 1.04),
                   rootogram=F, palet = NULL,
                   root_scale = "unscaled", subplot=NULL, ...) {

            if(is.null(palet)){
              palet <- "Accent"
            }

    # check that this is a mixture of part_fits
    # assign appropriate names for graph labelling
  if(is(x@components[[1]][[1]], "FLX_monoreg_component")){
    if(is.null( c(x@model[[1]]@mon_inc_names,
                  x@model[[1]]@mon_dec_names) )){
      xnames <- sapply(1:dim(x@components[[1]][[1]]@mon_obj)[2],
                       function(x) paste0("X", x))
      mono_names <- c("Y", xnames)
    }
    else{
      mono_names <- c(x@formula[[2]], c(x@model[[1]]@mon_inc_names,
                                        x@model[[1]]@mon_dec_names))
    }

            # get dimension of monotone components by reading
            # columns of fitted_pava object
            if(dim( x@components[[1]][[1]]@mon_obj )[2] > 1){
              np <- list()
              for(i in 1:dim( x@components[[1]][[1]]@mon_obj )[2]){
                holder <- ggplot()

                if(length(x@components) == 1){
                    holder <- holder +
                      geom_line(aes(x =
                        x@components[[1]][[1]]@mon_obj[,i]$x,
                              y =
                        x@components[[1]][[1]]@mon_obj[,i]$yf))
```

```
           + theme_bw() +
          labs(title = paste(append_suffix(i),
                          " Monotone Regression"),
           x = mono_names[i+1],
           y = mono_names[1])
         }

      if(length(x@components) > 1){

        monlist <- list()
        for(b in 1:length(x@components)){
          monlist[[b]] <- data.frame(x =
                   x@components[[b]][[1]]@mon_obj[,i]$x,
                                   yf =
                   x@components[[b]][[1]]@mon_obj[,i]$yf)
        }

        mondf   <- cbind(Cluster=
                          rep(1:length(x@components),
                           sapply(monlist,nrow)),
                        do.call(rbind,monlist))
        mondf$Cluster <- as.factor(mondf$Cluster)

        holder <- holder + geom_line(mondf, mapping =
                              aes(x,yf, color=Cluster))
          + scale_color_brewer(palette=palet) +
          theme_bw() +
          labs(title = paste(append_suffix(i),
                           " Monotone Regression"),
              x = mono_names[i+1],
              y = mono_names[1])
      }

      if(!is.null(ylim)){
        if(length(ylim) != dim(
          x@components[[1]][[1]]@mon_obj )[2] |
           length(ylim[[1]]) != 2 ){
          stop("If you pass a ylim argument, it must have
          as many element pairs as the model has monotone
          components. Try formulating the argument
               as: ylim = list(c(i,j), c(i,j), ...)")}
        holder <- holder + ylim(ylim[[i]])
      }
      if(!is.null(xlim)){
        if(length(xlim) != dim(
          x@components[[1]][[1]]@mon_obj )[2] |
           length(xlim[[1]]) != 2 ){
          stop("If you pass a xlim argument, it must have
          as many element pairs as the model has monotone
          components. Try formulating the argument
               as: xlim = list(c(i,j), c(i,j), ...)")}
        holder <- holder + xlim(xlim[[i]])
      }
      if(!is.null(ylab)){
        if(length(ylab) != dim(
          x@components[[1]][[1]]@mon_obj )[2]){
          stop("If you pass a ylab argument, it must have
          as many elements as the model has monotone
          components. Try formulating the argument
               as: ylab = c(\"first\",\"second\",...)")}
        holder <- holder + ylab(ylab[[i]])
      }
      if(!is.null(xlab)){
        if(length(xlab) != dim(
          x@components[[1]][[1]]@mon_obj )[2]){
          stop("If you pass a xlab argument, it must have
          as many elements as the model has monotone
          components. Try formulating the argument
               as: xlab = c(\"first\",\"second\",...)")}
        holder <- holder + xlab(xlab[[i]])
      }
      if(!is.null(main)){
        if(length(main) != dim(
          x@components[[1]][[1]]@mon_obj )[2]){
          stop("If you pass a main argument, it must have
          as many elements as the model has monotone
          components. Try formulating the argument
               as: main = c(\"first\",\"second\",...)")}
```

```
                  holder <- holder + ggtitle(main[[i]])
                }




              np[[i]] <- holder
            }
            # return(grid.arrange(grobs=np, ncol=1))
            # return(grid.arrange(grobs=np, ncol=1))
          }
          else{
            np <- ggplot()

            if(length(x@components) == 1){
              np <- np + geom_line(aes(x =
                      x@components[[1]][[1]]@mon_obj[,1]$x, y =
                      x@components[[1]][[1]]@mon_obj[,1]$yf)) +
                theme_bw() +
                labs(title = "Monotone Component",
                  x = mono_names[2],
                  y = mono_names[1])
              }

            if(length(x@components) > 1){

              monlist <- list()
              for(b in 1:length(x@components)){
                monlist[[b]] <- data.frame(x =
                          x@components[[b]][[1]]@mon_obj[,1]$x,
                                                yf =
                          x@components[[b]][[1]]@mon_obj[,1]$yf)
              }

              mondf   <- cbind(Cluster=rep(1:length(x@components),
                                          sapply(monlist,nrow)),
                                do.call(rbind, monlist))
              mondf$Cluster <- as.factor(mondf$Cluster)

              np <- np + geom_line(mondf, mapping = aes(x,yf,
                                                color=Cluster)) +
                     scale_color_brewer(palette=palet) +
                     theme_bw() +
                     labs(title = "Monotone Component",
                          x = mono_names[2],
                          y = mono_names[1])
                 }



                 if(!is.null(ylim)){
                   np <- np + ylim(ylim)
                 }
                 if(!is.null(xlim)){
                   np <- np + xlim(xlim)
                 }
                 if(!is.null(ylab)){
                   np <- np + ylab(ylab)
                 }
                 if(!is.null(xlab)){
                   np <- np + xlab(xlab)
                 }
                 if(!is.null(main)){
                   np <- np + ggtitle(main)
                 }





                 # return(np)
               }
  }
               # plot and append rootogram
               # collect posteriors
               post <- data.frame(x@posterior$scaled)
               # change columns of posteriors to cluster numbers
               names(post) <- 1:dim(post)[2]
               post <- melt(setDT(post), measure.vars =
                         c(1:dim(post)[2]), variable.name = "Cluster")
               # plot rootogram, with color indicating cluster
```

```
            rg <- ggplot(post, aes(x=value, fill=Cluster)) +
              geom_histogram(binwidth = 0.05) +
              scale_fill_brewer(palette=palet) +
              theme_bw() +
              labs(title = "Rootogram",
                   x = "Posteriors",
                   y = "Count")

            if(root_scale == "sqrt"){rg <- rg +
              scale_y_sqrt() +
              labs(title = "Rootogram (square root scale)",
                   x = "Posteriors",
                   y = "Count (square root)")}
            if(root_scale == "log"){rg <- rg +
              scale_y_log10() +
              labs(title = "Rootogram (log scale)",
                   x = "Posteriors",
                   y = "Count (log)")}

            if(!is.null(subplot)){
              return(list(rg, np)[[subplot[1]]])
            }
            else{
              multiplot(rg, np)
            }
          }
)
```

# Bibliography

Ait-Sahalia, Y. and Duarte, J. (2003). Nonparametric option pricing under shape restrictions. *Journal of Econometrics*, 116(1):9–47. Frontiers of financial econometrics and financial engineering. 1

Ayer, M., Brunk, H. D., Ewing, G. M., Reid, W. T., and Silverman, E. (1955). An Empirical Distribution Function for Sampling with Incomplete Information. *The Annals of Mathematical Statistics*, 26(4):641 – 647. 4

Bank, W. (2018). World bank open data. `https://data.worldbank.org/indicator/NY.GDP.PCAP.CD`. 27

Best, M. and Chakravarti, N. (1990). Active set algorithms for isotonic regression; a unifying framework. *Math. Program.*, 47:425–439. 20

Breiman, L. and Friedman, J. H. (1985). Estimating optimal transformations for multiple regression and correlation. *Journal of the American Statistical Association*, 80(391):580–598. 4, 11

Brunk, H. D. (1958). On the Estimation of Parameters Restricted by Inequalities. *The Annals of Mathematical Statistics*, 29(2):437 – 454. 4

Cai, B. and Dunson, D. B. (2007). Bayesian multivariate isotonic regression splines: Applications to carcinogenicity studies. *Journal of the American Statistical Association*, 102:1158–1171. 1

Cheng, G. (2009). Semiparametric additive isotonic regression. *Journal of Statistical Planning and Inference*, 139:1980–1991. 12, 18, 39

Cheng, K.-F. and Lin, P.-E. (1981). Nonparametric estimation of a regression function: Limiting distribution2. *Australian Journal of Statistics*, 23(2):186–195. 4

De Leeuw, J., Kurt, H., and Mair, P. (2009). Isotone optimization in r: Pool-adjacent-violators algorithm (pava) and active set methods. *Journal of Statistical Software*, 32. 38

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38. 5

Diggle, P., Morris, S., Elliott, P., and Shaddick, G. (1997). Regression modelling of disease risk in relation to point sources. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 160(3):491–505. 3

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26. 19

Engle, R. F., Granger, C. W. J., Rice, J., and Weiss, A. (1986). Semiparametric estimates of the relation between weather and electricity sales. *Journal of the American Statistical Association*, 81(394):310–320. 4

Fraley, C. and Rafter, A. (2012). Mclust version 3 for r: Normal mixture modeling and model-based clustering. *Technical Report, Department of Statistics, University of Washington*, 504. 5

Friedman, J. and Tibshirani, R. (1984). The monotone smoothing of scatterplots. *Technometrics*, 26(3):243–250. 4

Frisen, M. (1986). Unimodal regression. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 35(4):479–485. 3

Gao, L. L., Bien, J., and Witten, D. (2020). Selective inference for hierarchical clustering. 19, 38

Geenens, G. (2011). Curse of dimensionality and related issues in nonparametric functional regression. *Statistics Surveys*, 5(none):30 – 43. 11

Gormley, I. C. and Frühwirth-Schnatter, S. (2018). Mixtures of experts models. 9

Grenander, U. (1956). On the theory of mortality measurement. *Scandinavian Actuarial Journal*, 1956(1):70–96. 4

Groeneboom, P., Jongbloed, G., and Wellner, J. A. (2001). Estimation of a Convex Function: Characterizations and Asymptotic Theory. *The Annals of Statistics*, 29(6):1653 – 1698. 3

Grün, B. and Leisch, F. (2008). *Finite Mixtures of Generalized Linear Regression Models*, pages 205–230. Physica-Verlag HD, Heidelberg. 38

Grün, B. and Leisch, F. (2009). Dealing with label switching in mixture models under genuine multimodality. *Journal of Multivariate Analysis*, 100(5):851–861. 19

Grün, B., Leisch, F., Shalabh, S., and Heumann, C. (2008). *Finite Mixtures of Generalized Linear Regression Models*, pages 205–230. 5

Guntuboyina, A. and Sen, B. (2018). Nonparametric shape-restricted regression. 3

Hamilton, S. A. and Truong, Y. K. (1997). Local linear estimation in partly linear models. *Journal of Multivariate Analysis*, 60(1):1–19. 4

Hanson, D. L. and Pledger, G. (1976). Consistency in Concave Regression. *The Annals of Statistics*, 4(6):1038 – 1050. 3

Hastie, T. and Tibshirani, R. (1986). Generalized Additive Models. *Statistical Science*, 1(3):297 – 310. 4

Hildreth, C. (1954). Point estimates of ordinates of concave functions. *Journal of the American Statistical Association*, 49(267):598–619. 3

Hu, J., Kapoor, M., Zhang, W., Hamilton, S. R., and Coombes, K. R. (2005). Analysis of dose–response effects on gene expression data with comparison of two microarray platforms. *Bioinformatics*, 21(17):3524–3529. 3

Huang, M., Li, R., and Wang, S. (2013). Nonparametric mixture of regression models. *Journal of the American Statistical Association*, 108(503):929–941. 5, 18

Hurn, M., Justel, A., and Robert, C. P. (2003). Estimating mixtures of regressions. *Journal of Computational and Graphical Statistics*, 12(1):55–79. 5

Jensen, J. L. W. V. (1906). Sur les fonctions convexes et les inégualités entre les valeurs Moyennes. 9

Jordan, A., Mühlemann, A., and Ziegel, J. (2019). Optimal solutions to the isotonic regression problem. 13

Kiri Wagstaff, C. C. (2000). Clustering with instance-level constraints. *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1103–1110. 5

Kuosmanen, T. (2008). Representation theorem for convex nonparametric least squares. *The Econometrics Journal*, 11(2):308–325. 3

Leisch, F. (2004). FlexMix: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software*, 11(8):1–18. 40

Liang, H. (2006). Estimation in partially linear models and numerical comparisons. *Computational Statistics & Data Analysis*, 50(3):675–687. 4

Luss, R., Rosset, S., and Shahar, M. (2012). Efficient regularized isotonic regression with application to gene–gene interaction search. *The Annals of Applied Statistics*, 6(1). 3

Mammen, E. (1991a). Estimating a Smooth Monotone Regression Function. *The Annals of Statistics*, 19(2):724 – 740. 4

Mammen, E. (1991b). Nonparametric Regression Under Qualitative Smoothness Assumptions. *The Annals of Statistics*, 19(2):741 – 759. 3

Marin, J.-M., Mengersen, K., and Robert, C. (2005). Bayesian modelling and inference on mixtures of distributions. *Handbook of Statistics*, 25. 5

Mazumder, R., Choudhury, A., Iyengar, G., and Sen, B. (2015). A computational framework for multivariate convex regression and its variants. 3

McLachlan, G. and Peel, D. (1999). The emmix algorithm for the fitting of normal and t-components. *Journal of Statistical Software, Articles*, 4(2):1–14. 5

Morton-Jones, T., Diggle, P., Parker, L., Dickinson, H. O., and Binks, K. (2000). Additive isotonic regression models in epidemiology. *Statistics in Medicine*, 19(6):849–859. 1, 38

Newcomb, S. (1886). A generalized theory of the combination of observations so as to obtain the best result. *American Journal of Mathematics*, 8(4):343–366. 5

Oussalah, A., Gleye, S., clerc urmès, I., Laugel, E., Barbé, F., Orlowski, S., Malaplate, C., Aimone-Gastin, I., Caillierez, B., Merten, M., Jeannesson, E., Kormann, R., Olivier, J.-L., Rodriguez-Guéant, R.-M., Namour, F., Bevilacqua, S., Thilly, N., Losser, M.-R., Kimmoun, A., and Guéant, J.-L. (2020). The spectrum of biochemical alterations associated with organ dysfunction and inflammatory status and their association with disease outcomes in severe covid-19: A longitudinal cohort and time-series design study. *EClinicalMedicine*, 27:100554. 1

Pearson, K. (1894). Contributions to the mathematical theory of evolution. ii. skew variation in homogeneous material. *Philosophical Transactions of the Royal Society of London*, 186:343–414. 5

Rasmussen, C. (2000). The infinite gaussian mixture model. *Advances in Neural Information Processing Systems 12*, pages 554–560. 5

Seijo, E. and Sen, B. (2011). Nonparametric least squares estimation of a multivariate convex regression function. *The Annals of Statistics*, 39(3):1633 – 1657. 3

Speckman, P. (1988). Kernel smoothing in partial linear models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(3):413–436. 4

Trip, T. K. (2018). Computational complexity of machine learning algorithms. 20

Viele, K. and Tong, B. (2002). Modeling with mixtures of linear regressions. *Statistics and Computing*, 12:315–330. 5

Wright, F. T. (1981). The Asymptotic Behavior of Monotone Regression Estimates. *The Annals of Statistics*, 9(2):443 – 448. 4

Wu, X. and Liu, T. (2017). Estimation and testing for semiparametric mixtures of partially linear models. *Communications in Statistics - Theory and Methods*, 46(17):8690–8705. 5

Xiang, S. and Yao, W. (2016). Mixture of regression models with single-index. 5

Zhang, Y. and Pan, W. (2020). Estimation and inference for mixture of partially linear additive models. *Communications in Statistics - Theory and Methods*, 0(0):1–15. 5, 18

Zhang, Y. and Zheng, Q. (2018). Non parametric mixture of strictly monotone regression models. *Communications in Statistics - Theory and Methods*, 47(2):415–426. iii, 1, 18