Mixtures of Partially Linear Models with Monotone Shape Constraints

Daniel Leibovitz[1] **and Matthias Loffler**[2]

[1]**daniel.leibovitz@uzh.ch**
[2]**matthias.loeffler@stat.math.ethz.ch**

ABSTRACT

Mixtures of non-parametric monotone regressions are readily applicable to clustering problems where there is prior knowledge about appropriate shape constraints within the resulting model. For example, option pricing functions in finance and risk-exposure relationships in epidemiology are both *a priori* monotonic. 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. 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.

Keywords: Mixture Models, Shape Constraints, Isotonic Regression

1 INTRODUCTION

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

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

where the model has K components, $\boldsymbol{X} \in \mathbb{R}^p$, $\boldsymbol{Z} \in \mathbb{R}^p$, $\beta \in \mathbb{R}^p$, and each function $g_{hk}()$ is assumed monotone. The error ϵ is assumed to be normally distributed with mean 0 and to be independent of the covariates $(\boldsymbol{X}, \boldsymbol{Z})$. The prior probabilities π_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 ANDOR or convexity (Ait-Sahalia and Duarte (2003)); 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:

$$Y = \begin{cases} g_1(Z) \ + \ \epsilon_1, \text{ with probability } \pi_1(Z); \\ \vdots \\ g_k(Z) \ + \ \epsilon_k, \text{ with probability } \pi_k(Z); \end{cases} \tag{2}$$

where $\boldsymbol{Z} \in \mathbb{R}^1$, and each function $g_k()$ is assumed monotone. The variables ϵ_k and π_k are not constant, as in model 1, but rather nonparametric functions $\epsilon_k(Z)$ and $\pi_k(Z)$. They nonetheless satisfy the same conditions as in model 1, namely, $\pi_k(Z) \in (0, 1)$ and $\sum_1^k \pi_k(Z) = 1$ for any Z, and $\epsilon_k(Z)$ is normally distributed with $E(\epsilon_k|Z) = 0$, $Var(\epsilon_k|Z) = \sigma_k(Z)$.

We have identified four drawbacks of in the model and estimator of Zhang et al.:

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 4.5). Including such variables as linear effects is a safe 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 adds computational complexity.
4. Zhang et al. state that their model can be extended to the multivariate case, i.e., where Z in model 2 is multivariate, but decline to explicitly define this model or its estimation.

We propose model 1 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; it fits the monotone functions within each component in a single step; and it does not require the calibration of any tuning parameters.

The main weakness of our generalized model is that the monotone component of each regression is not a smooth function, as is the case in the approach by Zhang et al., but rather a step-wise function. However, if the user does not require smooth monotone functions, the advantages of the algorithm we propose may outweigh this cost.

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 (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 20^{th} century. In the final section, we discuss implications and future work.

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.

2.1 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 $E(Y) = X\beta$ in $\mathbb{R}^1$, 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 intuitively, in the unidimensional case, 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 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 original Pool Adjacent Violators algorithm for the estimation of the isotonic regression MLE with normally 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 or step-wise. 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 (e.g., Friedman & Tibshirani (Friedman and Tibshirani (1984))), or estimate a monotonic function and then smooth the resulting estimate (e.g., Cheng & Lin (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)).

2.2 Partial Linear Models

The Generalized Additive Model, or GAM, 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.

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.

2.3 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 do specify a dependence structure amongst observed covariates without modelling the distributions of the covariates themselves.

2.4 Mixtures of Nonparametric Regressions

Both Xiang and Yao (2016), and Huang et al. (2013) have discussed 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.

3 AN OVERVIEW OF CONTRIBUTING MODELS AND ESTIMATORS

3.1 Mixture Models and the EM Algorithm

3.1.1 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 (3):

$$p(X) \;=\; \sum_{k=1}^{K} \pi_k \cdot p(X|\theta_k) \tag{3}$$

where the distribution of each component k is parametrized by some set of parameters θ_k, and the number of observations generated by component k is proportionate to π_k. 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 variable $\mathcal{L}$ such that $\mathcal{L}_i \in \{1, .., K\}$ for all $i \in \{1, ..., n\}$. Then, the distribution of $\mathcal{L}$ can be denoted by equation 4, and the additive distribution in 3 can be deconstructed to the conditional distribution in equation 5.

$$p(\mathcal{L} = k) \;=\; \pi_k \tag{4}$$

$$p(X|\mathcal{L} = k) \;=\; p_k(X) \;=\; p(X|\theta_k) \tag{5}$$

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

3.1.2 Finite Mixtures of Regressions

If one further specifies the distributions $p_k()$ in equation 3 as being 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 $Y = f(\cdot|\vec{X}) + \epsilon$ or the distribution of $E(Y|\vec{X})$. Suppose we observe, instead of univariate X, $Y_i, ..., Y_n$ 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 $\boldsymbol{\mathcal{L}}$.

Thus equation 3 becomes equation 6, in which the distribution of Y is conditioned on the associated covariates $\vec{X}$. The likelihood of this model is written out in equation 7.

$$p(Y) \;=\; \sum_{k=1}^{K} \pi_k p(Y = y \mid X = x, \theta_k) \tag{6}$$

$$L(\vec{\pi}, \vec{\theta}) \;=\; \prod_{i=1}^{n} \sum_{k=1}^{K} \pi_k p_k(y_i \mid \vec{x_i},\ \theta_k) \tag{7}$$

3.1.3 The EM Algorithm

The EM algorithm is a general 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 8, 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 π_k is not required, as here we have assumed that each x_i has a *true* cluster assignment, also known as "hard" assignment.

$$\ell(\theta) \;=\; log \sum_{k=1}^{K} p_k(X = x, Z = z|\theta) \tag{8}$$

The EM algorithm allows us to avoid determining the maximum of equation 8 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:

$$\ell(\theta) \;=\; log \sum_{k=1}^{K} q(Z) \frac{p_k(X = x, Z = z|\theta)}{q(Z)} \tag{9}$$

Since the log-likelihood function is concave, Jensen's inequality (equation 10, 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 11.

$$E(f(X)) \geq f(E(X)), \text{ for convex function } f \tag{10}$$

$$log \sum_{k=1}^{K} q(Z) \frac{p_k(X = x, Z = z|\theta)}{q(Z)} \;\geq\; \sum_{k=1}^{K} q(Z) log \frac{p_k(X = x, Z = z|\theta)}{q(Z)} \tag{11}$$

The function $J() = \sum_{k=1}^{K} q(Z) log \frac{p_k(X=x, Z=z|\theta)}{q(Z)}$ therefore serves

the EM algo reaches a local maximum. that's why we run several times.

The EM algorithm for a mixture model

3.1.4 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 by summing over the product of the mixture prior and the density at the new observed data (12), a mixture of regressions model must classify with a bayesian approach of summing over the likelihood given the observed data (13).

[THIS IS ALL WRONG]

$$p(Y = y) \;=\; \sum_{k=1}^{K} \pi_k p_k(\vec{Y} = \vec{y} \,|\theta_k) \tag{12}$$

$$p(Z = j|X = x, \theta) \;=\; \frac{\pi_j p_j(y \mid X = x, \theta_j)}{\sum_{k=1}^{K} \pi_k p_k(y \mid X = x, \theta_k)} \tag{13}$$

This is simply the application of Bayes' theorem (14), where the distributions of the components k are multiplied by the posterior given $X = x$ rather than the uninformed prior π_k. Practically, it is not necessary to calculate the integral in the denominator of the posterior analytically; rather, the sum of the posteriors are normalized to sum to 1.

$$p(\theta|X) \;=\; \frac{p(X|\theta)p(\theta)}{\int_\theta p(X|\theta)p(\theta)} \tag{14}$$

3.2 Partially Linear Models

3.2.1 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:

$$Y = \sum_{h=1}^{p} g_h(Z_h) \;+\; \sum_{j=1}^{q} \beta_j X_j \;+\; \epsilon \tag{15}$$

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 ϵ 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, as pointed out by Hastie and Tibshirani (1986),

For example, Cheng (2009) show that, for the model estimated in section 3.2.2, the difference between the multivariate and additive estimate: where it is shown that estimate 17 is a consistent estimator of $\{\vec{\beta}, g_1, ..., g_p\}$

3.2.2 Partially Linear Models with Monotone Constraints

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

$$Y = \sum_{h=1}^{p} g_h(Z_h) \;+\; \sum_{j=1}^{q} \beta_j X_j \;+\; \epsilon \tag{16}$$

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 17. The estimation of the functions $g_1(\cdot), ...g_p(\cdot)$ is a problem of additive isotonic regression, discussed in the next subsection.

$$\{\hat{\vec{\beta}}, \hat{g}_1, ..., \hat{g}_p\} = \underset{\vec{\beta}, g_1:g_p}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \sum_{h=1}^{p} g_h(z_{ih}) \ - \ \sum_{j=1}^{q} \beta_j x_{ij})^2 \tag{17}$$

The MLE of the entire partial linear model is obtained via the backfitting algorithm, iterating through the two-step process (18, 19) until convergence.

$$(I) \qquad \{\hat{g}_1, ..., \hat{g}_p\} = \underset{g_1:g_p}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{q} \beta_j x_{ij} - \sum_{h=1}^{p} g_h(z_{ih}) \right) \qquad \textit{holding} \quad \vec{\beta} \quad \textit{fixed} \tag{18}$$

$$(II) \qquad \hat{\vec{\beta}} = \underset{\vec{\beta}}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \sum_{h=1}^{p} g_h(z_{ih}) - \sum_{j=1}^{q} \beta_j x_{ij} \right) \qquad \textit{holding} \quad \{g_1, ..., g_p\} \quad \textit{fixed} \tag{19}$$

This model and estimator is thoroughly explored by Cheng (2009), where it is shown that, under certain conditions (see section 6.3), $\hat{\beta}_n$ is $\sqrt{n}$-consistent (21), while the estimates $\{\hat{g}_1, ..., \hat{g}_p\}$ converge in distribution as to a two-sided brownian motion plus a parabola (20).

$$\begin{aligned} n^{1/3} \frac{(2p_{Z_h}(z_h))^{1/3}}{\sigma^{2/3} g_h(z_h)^{1/3}} [\hat{g}_h(z_h) - g_h(z_h)] &\xrightarrow{d} GCM(Z(t) + t^2) \\ \text{where } p_{Z_h}(z_h) &\text{ is the density of } Z_h \text{ evaluated at } z_h, \\ GCM(Z(t)) &\text{ is the greatest convex minorant of } Z(t) \\ \text{and } Z(t) &\text{ is a two-sided Brownian motion} \end{aligned} \tag{20}$$

$$\begin{aligned} \sqrt{n}(\hat{\beta}_n - \beta_0) &\xrightarrow{d} N(0, \Sigma) \\ \text{where } \Sigma &= \sigma^2 [E(X - \sum_{h=1}^{p} E(X|Z_h))^{\otimes 2}]^{-1} \\ \text{and } \sigma^2 &= Var(\epsilon) \end{aligned} \tag{21}$$

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.

3.3 Isotonic Regression

3.3.1 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 $\hat{g}(\cdot) = \operatorname{argmin}_g \sum_{i=1}^{n} ||g(x_i) - x_i||_L$ 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 discuss only the non-decreasing case.

If we consider specifically the squared error loss, our risk function and weighted risk function become equations 22 and 23 respectively. Equation 23 can alternately be written explicitly in the form of a min-max formula, as in equation 24 (Jordan et al. (2019)), giving it a characterization which facilitates the study of the properties of the estimator $\hat{g}(\cdot)$. Equation 24 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.

$$\hat{g}(\cdot) = \underset{g}{\text{argmin}} \sum_{i=1}^{n} (g(x_i) - x_i)^2 \tag{22}$$

$$\hat{g}(\cdot) = \underset{g}{\text{argmin}} \sum_{i=1}^{n} w_i (g(x_i) - x_i)^2 \tag{23}$$

$$\hat{g}(x_i) = \min_{j \geq i} \max_{k \leq j} \frac{\sum_{k=k}^{j} w_k x_k}{\sum_{k=k}^{j} w_k}, \; i = 1, ..., n \tag{24}$$

One can see from equation 24 that determining the estimate $\hat{g}(\cdot)$ for the least square isotonic regression returns a step function, and requires no tuning parameter.

3.3.2 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 (25) – returns a step-function fit without either having to select a bandwidth or having to set a congergence tolerance parameter. For multivariable monotone regression (26), 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 26.

$$Y = g(X) \; + \; \epsilon \tag{25}$$

$$Y = \sum_{h=1}^{p} g_h(X_h) \; + \; \epsilon \tag{26}$$

4 PROPOSED MODEL

4.1 Model Definition

The model proposed in this article has the following structure:

$$Y = \begin{cases} \sum_{h=1}^{p} g_{h1}(Z_h) \ + \ \sum_{j=1}^{q} \beta_{j1} X_j \ + \ \epsilon_1 \text{ with probability } \pi_1; \\ \vdots \\ \sum_{h=1}^{p} g_{hk}(Z_h) \ + \ \sum_{j=1}^{q} \beta_{jk} X_j \ + \ \epsilon_k \text{ with probability } \pi_k; \end{cases} \tag{27}$$

where π_k represents the prior probability of mixture component k; $g_{hk}(\cdot)$ represents the monotone function of variable h within component k; β_{jk} represents the linear effect of variable j within mixture component k; and ϵ_k represents the error associated with component k. All ϵ_k are assumed to be normally distributed with mean 0, and are assumed to be independent of the covariates $(\boldsymbol{X}, \boldsymbol{Z})$. All π_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 27 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 β_j, including the intercept β_0, need not be same across different components k.

4.2 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

Data: ;
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: ;
$\mathcal{L}$ — 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 $j = 1, ..., k$, $\mathcal{L}_{ij}$ is a real number in the range $[0, 1]$, and $\sum_{j=1}^{k} \mathcal{L}_{ij} = 1$;
$\vec{\pi}$ — 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 Θ_k for each regression component k ;

Set iteration index $d \leftarrow 1$;
for $i \in 1, ...n$ **do**
  With uniform probability across k, assign one of the elements of $[\mathcal{L}_{i1}, ..., \mathcal{L}_{ik}]$ to 1 and all other to 0, such that $\mathcal{L}_i = [0, ..., 1, ..., 0]$;
end
while *algorithm is not converged* **do**
  for $j \in 1, ..., k$ **do**
    Set prior mixture proportion $\pi_j^{(d)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{ij}^{(d-1)}$;
    Set weighted partial linear model regression parameters such that $[\hat{\beta}_j, \hat{g}_j]^{(d)} \leftarrow \underset{\beta, g}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}_{ij}^{(d-1)} (y - x\beta_j - g_j(z))^2$ (See Algorithm 2, WPLR);
  end
  for $i \in 1, ..., n$ **do**
    for $j \in 1, ..., k$ **do**
      Set $\mathcal{L}_{ij}^{(d)} \leftarrow \pi_j^{(d)} p(y_i | x_i, \beta_j^{(d)}, g_j^{(d)}(z_i))$, 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 $\sigma = \sqrt{\frac{\sum w_i r^2 / \bar{w}}{n - rk(X)}}$
    end
    Normalize the posterior probabilities $[\mathcal{L}_{i1}, ..., \mathcal{L}_{ik}]^{(d)}$ such that $\sum_{j=1}^{k} \mathcal{L}_{ij}^{(d)} = 1$;
  end
  $d = d + 1$;
end

Algorithm 2: Weighted Partial Linear Regression

Data: ;
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 $[\hat{\beta}, \hat{g_1}, ..., \hat{g_q}] = \underset{\beta, g_1, ..., g_q}{\operatorname{argmin}} \sum_{i=1}^{n} w_i (y_i - \sum_{h=1}^{q} g_h(z_{ih}) - x_i\beta)^2$

Set iteration index $b \leftarrow 1$;
Set $\hat{\beta}^{(0)} \leftarrow \beta_x$, where $[\beta_x, \beta_z] = \underset{\beta_x, \beta_z}{\operatorname{argmin}} \sum_{i=1}^{n} w_i (y_i - z_i\beta_z - x_i\beta_x)^2$;
while *algorithm is not converged* **do**
  if z *is univariate* **then**
    Set $\hat{g}^{(b)} \leftarrow \underset{g}{\operatorname{argmin}} \sum_{i=1}^{n} w_i([y_i - x_i\beta^{(b-1)}] - g(z_i))^2$ holding $\beta^{(b-1)}$ fixed. (See Algorithm 4, PAVA)
  else
    Set $\sum_{h=1}^{q} \hat{g}_h^{(b)} \leftarrow \underset{g_1, ..., g_q}{\operatorname{argmin}} \sum_{i=1}^{n} w_i([y_i - x_i\beta^{(b-1)}] - \sum_{h=1}^{q} g_h(z_{ih}))^2$ holding $\beta^{(b-1)}$ fixed. (See Algorithm 3, CPAV)
  end
  Set $\hat{\beta}^{(b)} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} w_i([y_i - g^{(b)}(z_i)] - x_i\beta)^2$ holding $g^{(b)}$ fixed.;
  $b = b + 1$;
end

Algorithm 3: Weighted Cyclic Pool Adjacent Violators Algorithm

Data: ;
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

$$[\hat{f}_1, ...\hat{f}_q] = \underset{f_1, ...f_q]}{\operatorname{argmin}} \sum_{i=1}^{n} w_i (y_i - \sum_{h=1}^{q} f_h(x_{ih}))^2$$

Set iteration index $m \leftarrow 1$;
while *algorithm is not converged* **do**
  for $h \in 1, ..., q$ **do**
    Set $\hat{f}_h \leftarrow \underset{f_h}{\operatorname{argmin}} \sum_{i=1}^{n} w_i([y_i - \sum_{\substack{j=1 \\ j \neq h}}^{q} f_j(x_{ih})] - f_h(x_{ih}))^2$ holding all $f_j(), j \neq h$ fixed (See Algorithm 4, PAVA) ;
  end
  $m = m + 1$;
end

Algorithm 4: Weighted Pool Adjacent Violators Algorithm

Data: ;
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 $\hat{f}(\cdot) = \underset{f}{\operatorname{argmin}} \sum_{i=1}^{n} w_i(y_i - f(x_i))^2$

Set iteration index $l \leftarrow 0$;
Set blocks $r \leftarrow 1, ..., B$ where at $l = 0$, $B = n$;
Set $f^{(l=0)}(x_i) \leftarrow y_i$;
Set initial block membership $f^{(l=0)}(x_i) \in r_i$;
while *any* $f_r^l(x) \geq f_{r+1}^l(x)$ **do**
    if $f_r^l(x) > f_{r+1}^l(x)$ **then**
        Merge blocks r and $r+1$
    end
    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 $x \in r$;
    $l = l + 1$;
end

4.3 Asymptotic Properties of Model and Estimator

[Discuss the asymptotic standard error estimates of parameters and functions.]

4.4 Confidence Intervals via Bootstrapping

We opt to implement and demonstrate the proposed model with confidence intervals calculated by bootstrapping. In this regard, we use and compare two approaches in the remainder of this paper – the ordinary, or traditional bootstrap, and a component-wise bootstrap conditioned on the posterior matrix of the first model. Each approach has advantages and disadvantages, which we discuss in the following sections.

4.4.1 Ordinary Bootstrap

Although the ordinary bootstrap applied to mixture models delivers all the typical advantages of bootstrapping for determining parameter confidence intervals, it runs into the well-known label-switching problem inherent in clustering algorithms. 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, nor even a higher probability, 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.

The disadvantage of this approach is that it ignores the possibility of bootstrap-estimated parameters reaching different likelihood maxima as a result of having had random as opposed to fixed starting values. For this reason, this approach underestimates the parameter variance. Moreover, if the first model which one fit returned estimates from a local likelihood maximum, subsequent bootstrap estimates

may likewise be caught in the same local likelihood maximum. Parameter distributions from such a bootstrap might then be heavily biased.

One obvious approach to the problem of label-switching would be to run the ordinary bootstrap with random starting values and to treat the bootstrap-estimated parameters and non-parametric functions as "observations". We would then have a set of observations with latent class membership, and moreover, a known number of classes k, which is in fact precisely the problem structure which typically motivates a mixture model. Thus, we would run yet another mixture model on these parameters to determine a probabalistic interpretation of the distributions of parameters per component, as well as an estimate of the posterior probability of each parameter set belonging to each component. The implementation of such a hypothetical model may be difficult given that each observed parameter set includes both point-estimates and step functions, and we therefore leave this task for future research.

4.4.2 Conditional Bootstrap

An alternative to the ordinary bootstrap is what we shall call in this section the *conditional bootstrap*. Instead of resampling with replacement from the data and rebuilding the entire mixture model from the beginning for each iteration of the bootstrap, the conditional bootstrap takes the posterior estimates for each observation from the initial model as fixed weights in the calculation of the mixture subcomponents. The data for the conditional bootstrap are resampled from the complete dataset as usual.

The conditional bootstrap can thus be thought of as the ordinary bootstrap for the mixture components, conditioned on the weights being equal to the point-estimate posteriors from the initial model. There are two clear advantages to this approach: First, the label-switching problem is entirely avoided, as the component labels are *a priori* known. Second, the bootstrap model refitting is extremely computationally efficient, as the original mixture model fitting process is run only once.

The disadvantage to this approach is, of course, that the uncertainty in the posterior estimates is entirely ignored in the construction of the final parameter distributions. Moreover, depending on the data, the posterior estimates may themselves be highly unstable. Thus, this approach underestimates the variance of the parameter estimates, and the degree to which it underestimates the variance is unknown. Moreover, if the initial model from which the posterior point-estimates are drawn terminated in a local maximum, the results of the conditional bootstrap may be highly biased and misleading.

4.5 Computational Complexity of Estimator

A common concern amongst users of this algorithm will be the speed of the estimator, and by extension, the computational complexity of the estimator. It is not possible to specify exact $O(\cdot)$ notation given that the number of iterations within each optimization step of the algorithm is problem-dependent. However, given the generalized nature of the algorithm, we can compare its complexity empirically for different types and numbers of features within the sub-component regression models by running 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*.

Figure 1. In the table above, we see the average 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 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.

5 MODEL APPLICATIONS

5.1 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:

$$Y_1 = 50 + X^3 + \epsilon_1$$
$$Y_2 = -50 + 0.04 \cdot X^5 + 30 \cdot X + \epsilon_2$$

where

$$\epsilon_1 \sim N(0, 200)$$
$$\epsilon_2 \sim N(0, 300)$$

and

$$\pi_1 = 0.65$$
$$\pi_2 = 0.35$$

and

$$X \sim Uniform(-10, 10)$$

We proceed to estimate a mixture of univariate regressions (28), 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.

$$Y = \sum_{k=1}^{2} \pi_k (g_k(X) \ + \ \epsilon_k) \tag{28}$$

As can be seen from figure (3), the algorithm is more uncertain of the monotone regression shapes where the true data generating functions overlap.

Figure 2. 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 3. 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 1000 iterations of an ordinary (non-parametric) bootstrap.

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:

$$
\begin{aligned}
Y_1 &= 10 + X_1^3 + 1.5 \cdot X_2 - 1.5 \cdot X_3 - X_4 + X_5 + \epsilon_1 \\
Y_2 &= -10 + 25 \cdot X_1 + 3 \cdot X_2 + 2 \cdot X_3 - 2 \cdot X_4 + 2 \cdot X_5 + \epsilon_2 \\
Y_3 &= -4 + 1.5 \cdot (X_1^3) - 2 \cdot X_2 - X_3 + 2 \cdot X_4 + 4 \cdot X_5 + \epsilon_3 \\
Y_4 &= 4 + 0.1 \cdot (X_1^5) - 3 \cdot X_2 - 3 \cdot X_3 - 3 \cdot X_4 + 3 \cdot X_5 + \epsilon_4
\end{aligned}
$$

where

$$
\begin{array}{ll}
\epsilon_1 \sim N(0,3) & X_1 \sim Uniform(-5,5) \\
\epsilon_2 \sim N(0,10) & X_2 \sim Uniform(-10,10) \\
\epsilon_3 \sim N(0,3) & X_3 \sim Uniform(-100,100) \\
\epsilon_4 \sim N(0,7) & X_4 \sim Uniform(-100,100) \\
 & X_5 \sim Uniform(-100,100)
\end{array}
$$

We proceed to estimate a mixture of partial linear regressions (29), 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.

$$
Y = \sum_{k=1}^{4} \pi_k (g_k(X_1) \ + \ \beta_{0,k} \ + \ \beta_{1,k} \cdot X_2 \ + \ \beta_{2,k} \cdot X_3 \ + \ \beta_{3,k} \cdot X_4 \ + \ \beta_{4,k} \cdot X_5 \ + \ \epsilon_k) \tag{29}
$$

5.2 Algorithm Comparisons with Simulated Data

In such instances, the proposed algorithm is demonstrably tighter around the true functions than the algorithm of Zhang et al.

Figure 4. The estimated monotone functions of the two-component mixture model, now with confidence intervals generated via 1000 iterations of the conditional bootstrap.

Figure 5. The rootogram of the four-component mixture model represented by equation 29.

Figure 6. 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 1000 iterations of an ordinary (non-parametric) bootstrap.

(INCLUDE COMPARISON PLOTS HERE).

5.3 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 [@worldbank]. 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 (8), and a mostly logarithmic relationship between Life Expectancy & GDP (9).

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 21 mixture models, 3 for each of $k = 1, ..., 7$. These models have the form of 30 and 31 respectively.

$$Y = \sum_{k=1}^{K} \pi_k (g_k(GDP) \ + \ \beta_k \cdot Year \ + \ \epsilon_k) \tag{30}$$

$$Y = \sum_{k=1}^{K} \pi_k (g_k(GDP) \ + \ \beta_{0,k} \ + \ \beta_{1,k} \cdot Year \ + \ \epsilon_k) \tag{31}$$

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.

Next, we plot the distribution of clusters within the lowest-AIC model of each step-model series, projected onto a world map. 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

Figure 7. The estimated monotone functions of the two-component mixture model, now with confidence intervals generated via 1000 iterations of the conditional bootstrap.

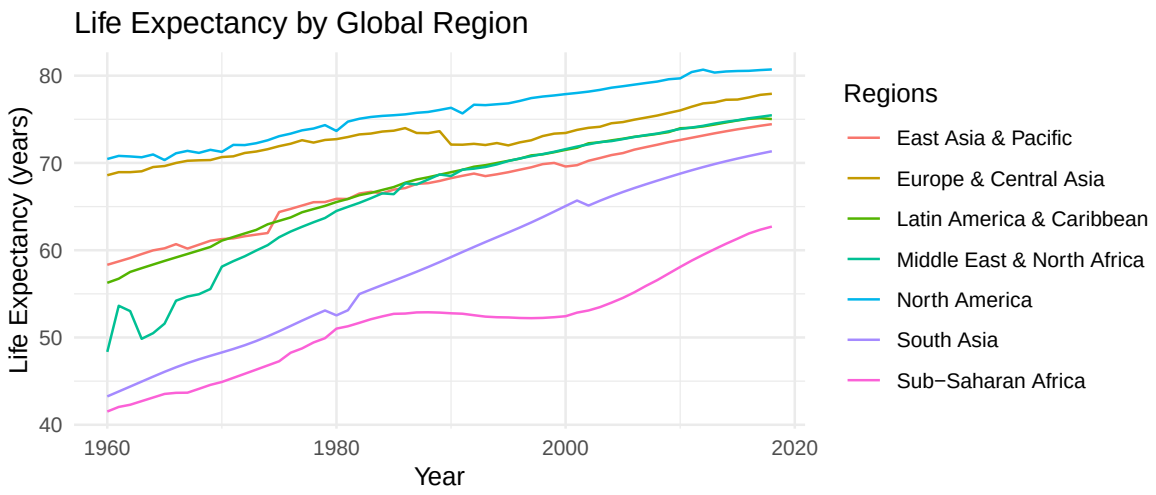

Figure 8. The relationships between Life Expectancy and time are largely linear across all global regions. Sub-saharan Africa uniquely demonstrates what appears to be a cubic relationship over time.

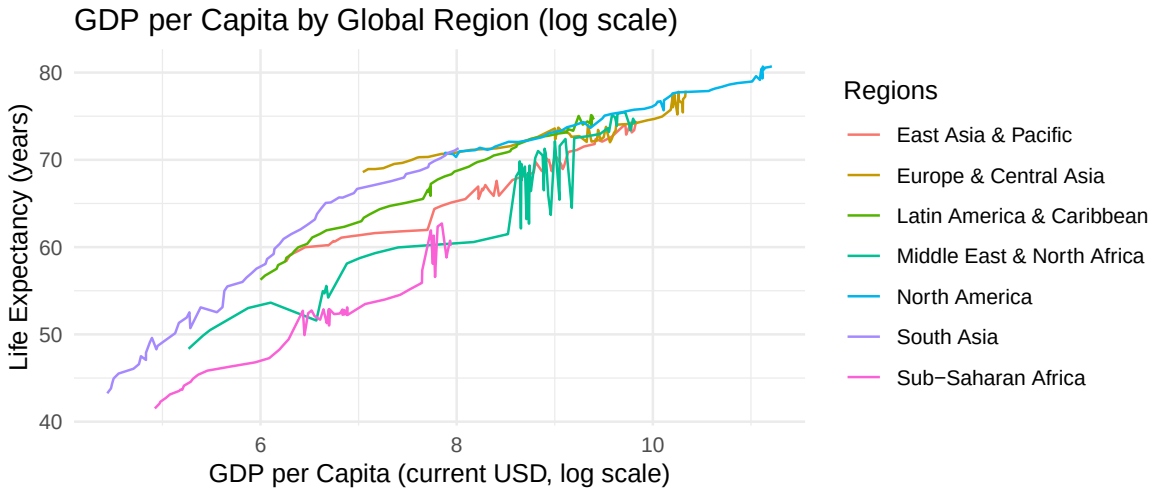

Figure 9. The relationships between Life Expectancy and GDP per capita are largely log-linear across global regions. Nearly all regions feature brief noisy sections surrounded by notably larger smooth sections.

Figure 10. Here we observe the various model selection metrics – AIC, BIC and ICL – for models represented by equation 30, constructed with each of k components. Below, we have the estimated monotone components of the model with lowest AIC.

Figure 11. Here we observe the various model selection metrics – AIC, BIC and ICL – for models represented by equation 31, constructed with each of k components. Below, we have the estimated monotone components of the model with lowest AIC.

Figure 12. Here we observe the world map with colour code corresponding to the results of model 30.

Figure 13. Here we observe the world map with colour code corresponding to the results of model 31.

indicated, namely that the resulting models are extremely confident about the clustering of nearly all countries.

Finally, we refit model 30 while excluding all observations from Australia in order to demonstrate the predictive capacities of the mixture model. As stated previously, in section 3.1.4, 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 Australia over several years, in which case one could use the mixture model to generate a set of posterior probabilities representing the probability of Australia 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 Australia over several years, in which case one could use the mixture model to generate a conditional marginal distribution of `Life Expectancy` for Australia.

6 DISCUSSION

6.1 Applications

6.2 Weaknesses

6.3 Future Developments

ACKNOWLEDGMENTS

Here I acknowledge all my homies.

REFERENCES

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.

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.

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.

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

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.

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

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

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.

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.

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.

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.

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

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

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

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.

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

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

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

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

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

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.

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

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

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

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

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

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

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).

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

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

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

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

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.

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.

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.

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.
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.
Rasmussen, C. (2000). The infinite gaussian mixture model. *Advances in Neural Information Processing Systems 12*, pages 554–560.
Seijo, E. and Sen, B. (2011). Nonparametric least squares estimation of a multivariate convex regression function. *The Annals of Statistics*, 39(3):1633 – 1657.
Viele, K. and Tong, B. (2002). Modeling with mixtures of linear regressions. *Statistics and Computing*, 12:315–330.
Wright, F. T. (1981). The Asymptotic Behavior of Monotone Regression Estimates. *The Annals of Statistics*, 9(2):443 – 448.
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.
Xiang, S. and Yao, W. (2016). Mixture of regression models with single-index.
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.
Zhang, Y. and Zheng, Q. (2018). Non parametric mixture of strictly monotone regression models. *Communications in Statistics - Theory and Methods*, 47(2):415–426.

Appendices

Asymptotic Behaviour of Partially Linear Models with Monotone Constraints

As discussed in (REF SECTION), the asymptotic behaviour of partially linear models with monotone constraints described by Cheng (2009) requires that the following assumptions hold:

page 1983 guang cheng

R Code

All results in this paper were produced by an extension of Flexmix (CITE), a flexible implementation of generalized mixture models written by FLEISCH and GRUN. 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 code. More specifically, Flexmix implements:

1. The universal functions of the EM algorithm for assigning prior values π_k to each of the mixture components and posterior values $\mathcal{L}_i$ 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() (FONT), which returns the density of an observation given a component's parameters. In the case of mixtures of regressions, this additionally implies a prediction function, predict() (FONT),

which gives an expected value of an observation's dependent variable given the observed independent variables.

WHAT ELSE DOES FLEXMIX DO FOR US?

The 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; the fourth code block implements the "conditional bootstrap" described in section (LINK).

initialize solution (CITE Finite Mixture Model Diagnostics Using Resampling Methods)

```
# Define function for fitting a partial linear model with arbitrary monotone-constrained

# import libraries
library(gridExtra)
library(dplyr)


cpav <- function(x_mat, y, weights, inc_index=NULL, dec_index=NULL, max_iters_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.w
  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 t
                                          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
        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

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

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

    # 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.predic

    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
```

```
        # indicated by join_ind, except the value of joint_ind at the ith place in joint_
        suppressWarnings(monoreg(x = x_mat[,joint_ind[i]],
                y = (y - (as.matrix(x_mat[,joint_ind[-i]]) %*% start_betas[-i]) ), w = w
        }
      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 = w
      })

    # iterate through mr_fits. each column of mr_fits (e.g., mr_fits[,1]) is a monoreg f
    # 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 <- 0.0000001
    }

    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 th
          # indicated by join_ind, except the value of joint_ind at the ith place in joi
          mr_fits[,i] <- suppressWarnings(monoreg(x = x_mat[,joint_ind[i]],
                                  y = (y - get_pred(mr_fits[,-i], x_mat[,joint_ind[-i]])
        }
        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]])
        }

      }

      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 o
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 man
                                           as monoreg() objects")
```

```
  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_ite
                     component = NULL, na.rm=T, mon_inc_names = NULL, mon_dec_names = NU

  # 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 remove
    cc <- complete.cases(y) & complete.cases(x) & complete.cases(wates)
    y <- y[cc]
    x <- x[cc,]
    wates <- wates[cc]
    cc <- NULL
  }

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


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

  # take monotone indices of previous component
  if(!is.null(component)){
    inc_ind <- component$mon_inc_index
    dec_ind <- component$mon_dec_index
  }
  else{
    # assume that monotone variable is first column in x and increasing, unless specifie
    if(!is.null(mon_inc_index)){
      inc_ind <- mon_inc_index
    }
    else{
      inc_ind <- 1
    }
    if(!is.null(mon_dec_index)){
      dec_ind <- mon_dec_index
    }
    else{
      dec_ind <- NULL
```

```
  }
}

# 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
                                                 monotone increasing and monotone decreasi

# 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 in
}
if(!is.null(dec_ind)){
  if(any(dec_ind != as.integer(dec_ind))) stop("Monotone decreasing indices are not in
}

# 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 relations

# 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 interce
}

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

  yhat <- cpav(x_mat = as.matrix(x[wates != 0,]), y = y[wates != 0], weights = wates[w
               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, ...,
  # 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
  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 <- 10000
  }

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

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

    old_betas <- betas    # save old betas for distance calculation
    # 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

    # 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_

# mod must have: coef attribute, sigma attribute, cov attribute, df attribute, ..., an
# 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"

  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)
  }
}
```

```
# The M-step of the EM Algorithm. Meshes with Flexmix Package.

# 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",
         # TODO what does FLXM_monoreg need to inherit?
         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 eith
                                                              names or indices can be cho

  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 FLXcompone
  # 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 observatio
                  logLik <- function(x, y) {
                    dnorm(y, mean=predict(x, ...), sd=fit$sigma, log=TRUE)
                  }
                  # @predict: A function(x) predicting y given x.
                  # TODO x must be partitioned into linear and monotone covars
                  predict <- function(x) {
                    inc_ind <- fit$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$sigma),
                      df = fit$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 t
                    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 spe
                    }
                    # 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)]
                  }

                  # if(any(apply(x, 2, function(x) is.factor(x)))) stop("x cannot have f

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

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

```
  }
```

```
# 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)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
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
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=
                   as.table = TRUE, endpoints = c(-0.04, 1.04), rootogram=F, palet = NULL
                   root_scale = "unscaled", subplot=NULL, ...) {

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


  if(is(x@components[[1]][[1]], "FLX_monoreg_component")){ # check that this is a mixture
    # assign appropriate names for graph labelling
    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",
      mono_names <- c("Y", xnames)
    }
    else{
      mono_names <- c(x@formula[[2]], c(x@model[[1]]@mon_inc_names, x@model[[1]]@mon_dec_

            # get dimension of monotone components by reading columns of fitted_pava obj
            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]$x,
                                               yf = x@components[[b]][[1]]@mon_obj[,i]$y
                  }

                  mondf    <- cbind(Cluster=rep(1:length(x@components),
                                                sapply(monlist,nrow)),do.call(rbind,monli
                  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 pair
             as the model has monotone components. Try formulating the argume
             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 pair
             as the model has monotone components. Try formulating the argume
             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 argume
             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 argume
             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 argume
             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[,
      }

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

```
            post <- data.frame(x@posterior$scaled) # collect posteriors
            names(post) <- 1:dim(post)[2] # change columns of posteriors to cluster numb
            post <- melt(setDT(post), measure.vars = c(1:dim(post)[2]), variable.name =
            rg <- ggplot(post, aes(x=value, fill=Cluster)) + # plot rootogram, with colo
              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)
            }
          }
)
```