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, utility functions in economics, 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 et al. (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, $\boldsymbol{X} \in \mathbb{R}^p$, $\boldsymbol{W} \in \mathbb{R}^p$, $\beta \in \mathbb{R}^p$, and each function $g_{hk}()$ is assumed monotone. The error ϵ is normally distributed with mean 0, and is independent of the covariates $(\boldsymbol{X}, \boldsymbol{Z})$. 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)); economics, where FILL (CITE); 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 et al. (Zhang and Zheng (2018)), and takes the following form:

$$Y = \left\{ \sum_{k=1}^{K} \pi_k (\sum_{h=1}^{p} g_{hk}(Z_h) \ + \ \sum_{j=1}^{q} \beta_{jk} X_j \ + \ \epsilon_k) \right. \tag{2}$$

We have identified three 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. approach 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. approach requires a tuning parameter for the fitting of each unconstrained mixture component function, which adds computational complexity.

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 (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

always non-parametric. monotone, concave, convex, discuss history, estimation methods (splines, inversion,), variance bounds. discuss monotone regressions in particular. different approaches – pava, active set, etc. The addition of a monotonicity shape constraint to any of the models mentioned thus far complicates their estimation, and indeed much has been written on the estimation of monotone nonparametric functions alone. As Zhang et al. do in their mixture model, most approaches to monotonic nonparametric estimation apply a two step algorithm, either estimating a smooth function and applying a monotonic constraint on the resulting function (e.g., Friedman & Tibshirani (Friedman and Tibshirani (1984))), or estimating a monotonic function and then smoothing the resulting estimate (e.g., Cheng & Lin (Cheng and Lin (1981))). Mammen compares the consistency of these approaches in a comprehensive treatment of smooth, monotonic nonparametric regression. Although these algorithms differ in their asymptotic behaviours, all of them require the selection of a tuning parameter (Mammen (1991)).

Monotone regressions alone have seen very diverse applications across research domains. They have been used to estimate disease risk as functions of spatial exposure (CITE), etc. etc.

2.2 partial linear models

discuss GAMs

2.3 Mixture Models

Model-based clustering models, or mixture models, are a common model for producing clusters with probabalistic or ”soft” cluster assignment. also talk about inclusion/exclusion constraints. discuss asymptotic MLE.

Finite mixture models date from the 19th century (CITE), while the more commonly known implementation via the EM algorithm was first introduced in 1977 (Dempster, Laird, Rubin). Ifinite mixture models: **?**

2.4 Mixtures of Regressions

From the larger set of mixture models, mixtures of regressions play a useful role in cases where we want to model a dependence structure amongst observed covariates without modelling the distributions of the covariates themselves. allows the analysis of such data where the researcher suspects latent categories among the observations.

2.5 Mixtures of Nonparametric Regressions

Both Xiang & Yao (CITE), and Huang et al. (CITE) 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 & Liu (CITE) propose a structure for estimating mixtures of PLMs with a univariate nonparametric effect and arbitrary linear effects per component. Most recently, Zhang & Pan (CITE) extended this model to accept arbitrary nonparametric effects.

2.6 Mixture Models with Monotone Shape Constraints

Within the smaller subset of mixtures of non-parametric regressions, we may frequently encounter situations in which we would like to place shape constraints on some, or each, of the components in our mixture model. A common such shape constraint is the monotonicity constraint, which ensures that in the regression model $E(Y|X) = f(X)$, the function $f(X)$ is either non-increasing or non-decreasing over the range of X.

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. There has been, however, much research in adjacent models.

3 AN OVERVIEW OF CONTRIBUTING MODELS AND ALGORITHMS

3.1 Mixture Models and the EM Algorithm

3.1.1 General Mixture Models

Given some number n of observed values $X_1, ..., X_n$ general mixture model has the following structure:

$$p(x) \ = \ \sum_{k=1}^{K} \pi_k \cdot p_k(x) \tag{3}$$

$$p(x) \ = \ \sum_{k=1}^{K} \pi_k \cdot p(x|Z_k) \tag{4}$$

3.1.2 Finite Mixtures of Regressions

At its most basic, the structure of a finite mixture of regressions model 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 $Y_i, ..., Y_n$ and associated $X_i, ..., X_n$. 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 vector of probabilities Z_i of length k, such that $\sum_{k=1}^{K} Z_{ik} = 1$.

Without loss of generality, we can assume that each specifying the exact form of the regression model $Y \sim f_k(\cdot|\vec{X})$ for component k, we can assume some vector of regression model parameters Θ and write the likelihood of the mixture model as such:

$$L(\pi, \Theta) \ = \ \prod_{i=1}^{n} \sum_{j=1}^{k} \pi_j p_j(y_i \mid \vec{x_i}, \ \Theta_j) \tag{5}$$

where π_j is the prior probability of component j, and p_j is the density of $f(Y)$ at y_i given observed x_i and Θ_j for component j. When the likelihood is maximized and parameters are estimated, the model provides the following:

1. An $n \times k$ matrix $\mathcal{L}$ 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 Θ_k for each regression component k

By extension, the fitted mixture model also allows us to compute the marginal density of Y given $X = x$:

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

We find the asymptotic global maximum [@emalgo] of the likelihood function via the EM algorithm, which is described in section 3.2 (Link).

3.1.3 The EM Algorithm

A finite mixture model can be estimated via the EM algorithm for any vector of distributions that have a calculable likelihood, which of course includes regression models.

3.2 Partially Linear Models and the Backfitting Algorithm

3.2.1 Partially Linear Models

a generalization of GAMs

3.2.2 The Backfitting Algorithm

describe backfitting mgcv as an alternative to backfitting

3.2.3 Partially Linear Models with Monotone Constraints

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

$$Y = \sum_{h=1}^{p} g_h(X_h) \ + \ \sum_{j=1}^{q} \beta_j X_j \ + \ \epsilon \tag{7}$$

where the model has h non-linear covariates and j linear covariates, where each $g_h()$ is some nonparametric function of X_h, 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:

$$\sum_{i=1}^{n} (y_i - \sum_{h=1}^{p} g_h(x_{ih}) \ - \ \sum_{j=1}^{q} \beta_j x_{ij})^2 \tag{8}$$

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

The MLE of the entire partial linear model is obtained via a backfitting algorithm suggested by Cheng [@cheng], and involves sequentially updating the linear and non-linear components of the partial linear model in a two-step process (9, 10) until convergence.

$$(I) \qquad \{\hat{g}_1, ..., \hat{g}_p\} = \underset{g_1:g_p}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{q} \beta_j x_{ij} - \sum_{h=1}^{p} g_h(x_{ih}) \right) \qquad \textit{holding} \;\; \vec{\beta} \;\; \textit{fixed} \tag{9}$$

$$(II) \qquad \hat{\vec{\beta}} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \left(y_i - \sum_{h=1}^{p} g_h(x_{ih}) - \sum_{j=1}^{q} \beta_j x_{ij} \right) \qquad \textit{holding} \;\; \{g_1, ..., g_p\} \;\; \textit{fixed} \tag{10}$$

3.3 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 $g(\cdot) = \operatorname{argmin}_g \sum_{i=1}^{n} (g(x_i) - x_i)^2$. If observations are weighted, the objective function becomes $g(\cdot) = \operatorname{argmin}_g \sum_{i=1}^{n} w_i (g(x_i) - x_i)^2$ 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.

The solution can be written out in the form of a min-max formula (Jordan et al. (2019)):

(MIN MAX EQUATION)

One well-known way to estimate $g()$ is through the Pool Adjacent Violators Algorithm (PAVA). The PAVA – for univariate monotone regression (11) – returns a step-function fit without either having to select a bandwidth or having to set a congergence tolerance parameter. For multivariable monotone regression (12), we must take a different approach suggested by Bacchetti and called the Cyclic Pool Adjacent Violators Algorithm (CPAV). Within the CPAV, we iterate through each univariate function sequentially and update univariate monotone functions until convergence, returning a sum of step-functions.

$$Y = g(X) \; + \; \epsilon \tag{11}$$

$$Y = \sum_{h=1}^{p} g_h(X_h) \; + \; \epsilon \tag{12}$$

4 PROPOSED MODEL

4.1 Model Definition

The model proposed in this article has the following structure:

$$Y = \begin{cases} \displaystyle\sum_{h=1}^{p} g_{h1}(Z_h) \; + \; \sum_{j=1}^{q} \beta_{j1} X_j \; + \; \epsilon_1 \text{ with probability } \pi_1; \\ \vdots \\ \displaystyle\sum_{h=1}^{p} g_{hk}(Z_h) \; + \; \sum_{j=1}^{q} \beta_{jk} X_j \; + \; \epsilon_k \text{ with probability } \pi_k; \end{cases} \tag{13}$$

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 13 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 linear effects β_j, including the intercept β_0, need not be same across different components k.

4.1.1 Model-Based Prediction

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.

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 (14), a mixture of regressions model must classify with a bayesian approach of summing over the likelihood given the observed data (15).

$$p(Y = y) \;=\; \sum_{k=1}^{K} \pi_k p_k(\vec{Y} = \vec{y} \;|\Theta_k) \tag{14}$$

$$p(Y = y|X = x) \;=\; \sum_{k=1}^{K} \frac{\pi_k p_k(y \mid X = x, \Theta_k)}{\int (f(x|theta)f(theta)d\theta)} p_k(Y = y \mid X = x, \Theta_k) \tag{15}$$

This is simply the application of Bayes' theorem (16), 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.

$$p(\theta|X) \;=\; \frac{p(X|\theta)p(\theta)}{\int_\theta p(X|\theta)p(\theta)} \tag{16}$$

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 $\Theta_k$ 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 $[\mathcal{L}_{i1}, ..., \mathcal{L}_{ik}]$ to 1 and all
         other to 0, such that $\mathcal{L}_i = [0, ..., 1, ..., 0]$ ;
 4  end
 5  while algorithm is not converged do
 6      for $j \in 1, ..., k$ do
 7          Set prior mixture proportion $\pi_j^{(d)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{ij}^{(d-1)}$ ;
 8          Set weighted partial linear model regression parameters such that
            $[\hat{\beta}_j, \hat{g}_j]^{(d)} \leftarrow \underset{\beta,g}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}_{ij}^{(d-1)} (y - x\beta_j - g_j(z))^2$ (See Algorithm 2, WPLR);
 9      end
10      for $i \in 1, ..., n$ do
11          for $j \in 1, ..., k$ do
12              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 $\sigma = \sqrt{\frac{\sum w_i r^2 / \bar{w}}{n - rk(X)}}$
13          end
14          Normalize the posterior probabilities $[\mathcal{L}_{i1}, ..., \mathcal{L}_{ik}]^{(d)}$ such that $\sum_{j=1}^{k} \mathcal{L}_{ij}^{(d)} = 1$;
15      end
16      $d = d + 1$;
17  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 $[\hat{\beta}, \hat{g_1}, ..., \hat{g_q}] = \operatorname*{argmin}_{\beta, g_1, ..., g_q} \sum_{i=1}^{n} w_i(y_i - \sum_{h=1}^{q} g_h(z_{ih}) - x_i\beta)^2$
1  Set iteration index $b \leftarrow 1$;
2  Set $\hat{\beta}^{(0)} \leftarrow \beta_x$, where $[\beta_x, \beta_z] = \operatorname*{argmin}_{\beta_x, \beta_z} \sum_{i=1}^{n} w_i(y_i - z_i\beta_z - x_i\beta_x)^2$;
3  while algorithm is not converged do
4      if z is univariate then
5          Set $\hat{g}^{(b)} \leftarrow \operatorname*{argmin}_{g} \sum_{i=1}^{n} w_i([y_i - x_i\beta^{(b-1)}] - g(z_i))^2$ holding $\beta^{(b-1)}$ fixed. (See Algorithm
           4, PAVA)
6      else
7          Set $\sum_{h=1}^{q} \hat{g}_h^{(b)} \leftarrow \operatorname*{argmin}_{g_1, ..., g_q} \sum_{i=1}^{n} w_i([y_i - x_i\beta^{(b-1)}] - \sum_{h=1}^{q} g_h(z_{ih}))^2$ holding $\beta^{(b-1)}$ fixed. (See
           Algorithm 3, CPAV)
8      end
9      Set $\hat{\beta}^{(b)} = \operatorname*{argmin}_{\beta} \sum_{i=1}^{n} w_i([y_i - g^{(b)}(z_i)] - x_i\beta)^2$ holding $g^{(b)}$ fixed.;
10     $b = b + 1$;
11 end
```

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
        $[\hat{f_1}, ...\hat{f_q}] = \operatorname*{argmin}_{f_1, ...f_q]} \sum_{i=1}^{n} w_i(y_i - \sum_{h=1}^{q} f_h(x_{ih}))^2$
1  Set iteration index $m \leftarrow 1$ ;
2  while algorithm is not converged do
3      for $h \in 1, ..., q$ do
4          Set $\hat{f_h} \leftarrow \operatorname*{argmin}_{f_h} \sum_{i=1}^{n} w_i([y_i - \sum_{\substack{j=1 \\ j \neq h}}^{q} f_j(x_{ih})] - f_h(x_{ih}))^2$ holding all $f_j(), j \neq h$ fixed (See
           Algorithm 4, PAVA) ;
5      end
6      $m = m + 1$;
7  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 $\hat{f}(\cdot) = \underset{f}{\text{argmin}} \sum_{i=1}^{n} w_i(y_i - f(x_i))^2$
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
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
```

4.3 Asymptotic Properties of Model and Estimator

Discuss the error estimates of parameters.

4.4 Confidence Intervals via Bootstrapping

4.4.1 Ordinary Bootstrap

4.4.2 Conditional Bootstrap

The obvious thing to do would be to take the bootstrapped parameters and functions as observations, and run yet another mixture model on these parameters to determine a probabalistic and bayesian interpretation of the distributions of parameters. The interpetation of such a hypothetical model would be difficult, and we leave this task for future research.

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 for different types and numbers of features within the component regression models, and we can support these comparisons with timed applications on pseudo-data.

TIMED COMPLEXITY: here.

In the benchmarking table above (LINK), 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

$$\epsilon_1 \sim N(0, 30)$$
$$\epsilon_2 \sim N(0, 20)$$

and

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

and

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

We proceed to estimate a mixture of univariate regressions (17), 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.

$$Y = \sum_{k=1}^{2} \pi_k (g_k(X) \ + \ \epsilon_k) \tag{17}$$

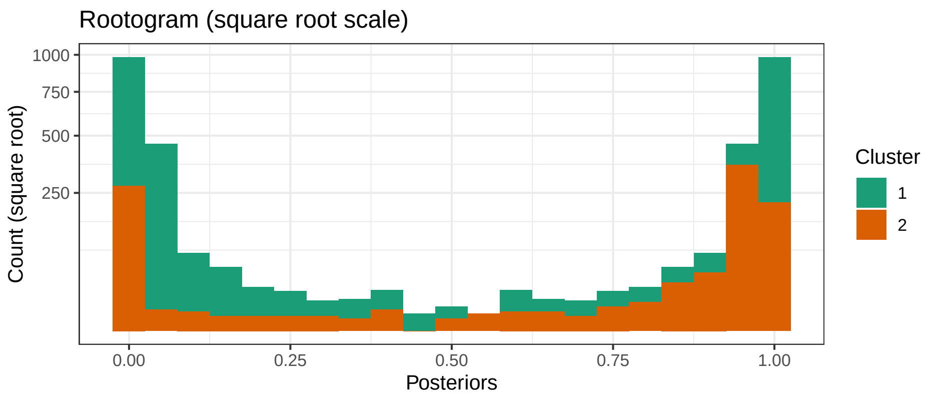


Figure 1. The rootogram of the two-component mixture model shows the distribution of posterior probabilities with reference to the binary latent variable, for all observations used to fit the model. The model indicates higher confidence in the identification of clusters and the classification of individual 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.

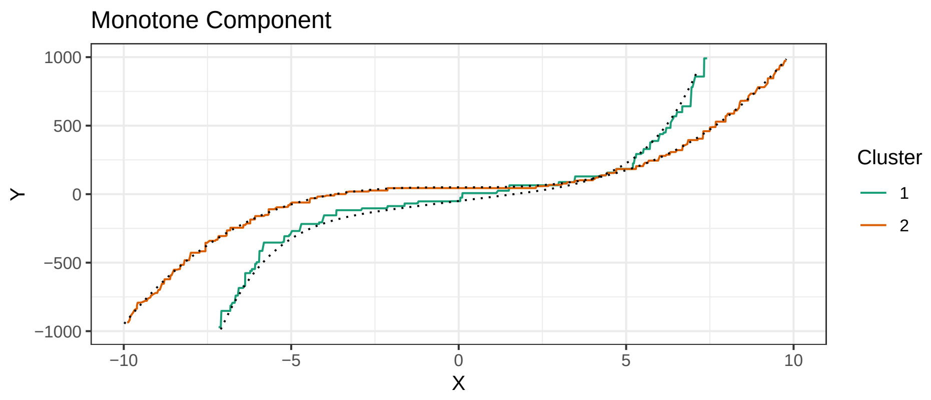


Figure 2. The estimated monotone functions of the two-component mixture model, with overlaid, dotted black lines representing the true functions. The confidence intervals are generated by an ordinary (non-parametric) bootstrap.

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

Alternately, we can fit mixtures with alternate specifications for each component. In the next demonstration, we model data generated from 2 latent categories with contrary monotonic effects, i.e., one with a monotone non-increasing true function and one with a monotone non-decreasing true function. The data has the following underlying structure:

$$Y_1 = X^3 + \epsilon_1$$
$$Y_2 = 100 + 0.02 \cdot X^5 + \epsilon_2$$

where

$$\epsilon_1 \sim N(0, 30)$$
$$\epsilon_2 \sim N(0, 20)$$

and

$$\pi_1 = 0.7$$
$$\pi_2 = 0.3$$

and

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

We proceed to estimate a mixture of univariate regressions (18), 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:

$$Y = \big\{\pi_1(g_1(X) \ + \ \epsilon_1)\pi_2(g_2(X) \ + \ \epsilon_2) \tag{18}$$

where g_1 is non-decreasing, and g_2 is non-increasing.

Figure 3. 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 4. 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 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 (19), 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.

$$Y = \sum_{k=1}^{4} \pi_k (g_k(X_1) \ + \ \beta_{0,k} \ + \ \beta_{1,k} \cdot X_2 \ + \ \beta_{2,k} \cdot X_3 \ + \ \beta_{3,k} \cdot X_4 \ + \ \beta_{4,k} \cdot X_5 \ + \ \epsilon_k) \tag{19}$$

Figure 5. Here we see the rootogram and mo

Figure 6. Estimated monotone functions from within each component of the

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.

(INCLUDE COMPARISON PLOTS HERE).

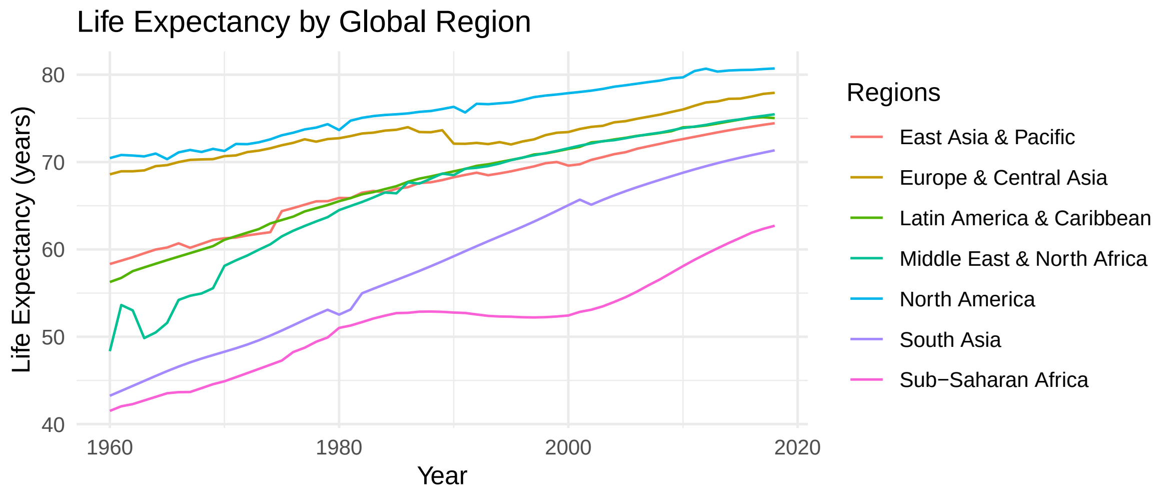
5.3 Real Data: Global Life Expectancy

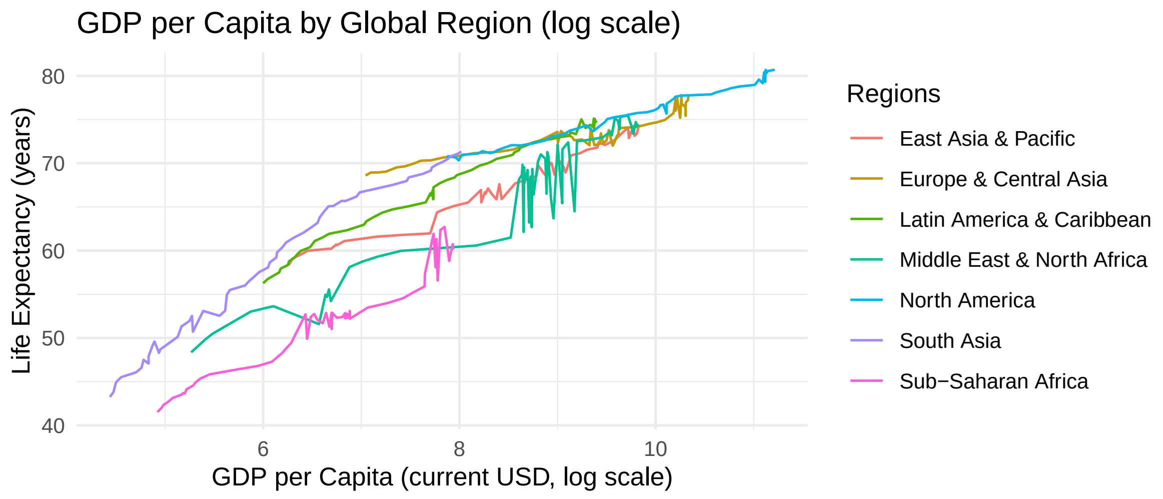
In this section, we apply the mixture of monotone regressions to global life expectancy data. Consider data on GDP per Person and Life Expectancy of all countries between the years 1960 and 2018, drawn from the free online resources of the World Bank [@worldbank]. Specifically, the data consists of n 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:
2. *A priori* Groupings: This data contains multiple observations per country, but we expect that our model will constrain countries to be clustered together (LINK constraints section).

On a first pass visualization of this data, we find a mostly linear relationship between Life Expectancy & Year (1), and a mostly logarithmic relationship between Life Expectancy & GDP (2).





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 20 and 21 respectively.

$$Y = \sum_{k=1}^{K} \pi_k (g_k(GDP) \ + \ \beta_k \cdot Year \ + \ \epsilon_k) \tag{20}$$

$$Y = \sum_{k=1}^{K} \pi_k (g_k(GDP) \ + \ \beta_{0,k} \ + \ \beta_{1,k} \cdot Year \ + \ \epsilon_k) \tag{21}$$

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.

[PLOT MODELS AIC/BIC AND MONOTONE COMPS HERE]

Figure 7. The

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

6 DISCUSSION

6.1 Applications

6.2 Weaknesses

6.3 Future Developments

ACKNOWLEDGMENTS

Here I acknowledge all my homies.

REFERENCES

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

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

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

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

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.

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.

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

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).

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

# import libraries
library(gridExtra)
library(dplyr)


cpav <- function(x_mat, y, weights, inc_index=NULL, dec_index=NULL, max_iters_cpav=NULL, max_delta_cp

  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 in-
puts 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 col-
umn
        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 col-
umn

        matrix(suppressWarnings(monoreg(x = x_mat[,dec_index], y = y, w = weights, type = "an-
titonic")),
               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.predict(all other v

    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
tonic"))
        }
      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
titonic"))
      })

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

    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 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 = weight
tonic"))
        }
        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 = weight
titonic"))
        }

      }

      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")

  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, mono_inc_names = NULL, mon_dec_names = 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 re-
moved.
    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 same di-
mension!")

  # 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 spec-
ified otherwise
    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 BOTH
                                                   monotone increasing and monotone decreas-
ing.")

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

  # 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 relation-
ships 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 inter-
cept as a linear component.")
  }
  # option for fit with no linear independent components and one or multiple monotone com-
ponents:
  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,
                mono_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. Try-
ing 1e-6

      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)

      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, x[,c(inc_in

      # 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)]) %*% b
tas))

  # 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,
              mono_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"),
```

```r
             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)
  }
}
```

```r
# 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 ei-
ther monotone
                                                            names or indices can be cho-
sen, 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 FLXcom-
ponent
  # 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 observa-
tions in matrices x and y
                 logLik <- function(x, y) {
                   dnorm(y, mean=predict(x, ...), sd=fit$sigma, log=TRUE)
                 }
                 # @predict: A function(x) predicting y given x.
                 # TODO x must be partitioned into linear and monotone covars
                 predict <- function(x) {
                   inc_ind <- fit$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 the name that
ing
                      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)]
                    }

                    # if(any(apply(x, 2, function(x) is.factor(x)))) stop("x cannot have fac-
tor columns as monotone components")

                    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=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"
            }


  if(is(x@components[[1]][[1]], "FLX_monoreg_component")){ # check that this is a mix-
ture of part_fits
    # 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", x))
```

```r
      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 ob-
ject
            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, mon-
```

```
list))
                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 num-
bers
            post <- melt(setDT(post), measure.vars = c(1:dim(post)[2]), variable.name = "Clus-
ter")
            rg <- ggplot(post, aes(x=value, fill=Cluster)) + # plot rootogram, with color in-
dicating 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)
            }
          }
)




internal_boot <- function(fm, R=100){

  # Check that fm is a flexmix object
  if(!is(fm, "flexmix")) stop("This method is for FlexMix objects of type: partially lin-
ear monotonic regression.")

  # Check that fm has at least one monotone obj
  if(!"mon_obj" %in% names(attributes(fm@components[[1]][[1]]))) stop("This method is for FlexMix ob-
jects of type: partially linear monotonic regression.")

  Y <- fm@model[[1]]@y # store independent data
  dat <- fm@model[[1]]@x # store dependent data
  if("(Intercept)" %in% colnames(fm@model[[1]]@x)){
    x <- x[, colnames(m3@model[[1]]@x) %in% "(Intercept)"] # remove intercept
  }

  k <- length(m2@components) # Store the number of components
  coefs <- rep(list(),k)
  sigmas <- matrix(ncol = k, nrow = R)
  mon_fits <- rep(list(rep(NA, R)),k)

  for(i in 1:k){ # populate each parameter
    iter <- 1
    while(iter < R ){
      # TODO give part_fit an optional formula argument
      # TODO give part_fit a starting coef and g() argument
      current <- part_fit(formula = fm@model[[1]]@fullformula, start = fm@components[[i]][[1]])

      coefs[[i]][[iter]] <- current$coef
      sigmas[iter, i] <- current$sigma
      mon_fits[[i]][[iter]] <-current$fitted_pava

      # assign
      iter <- iter + 1
    }

  }
```

```
}
```