The New England Journal of Statistics in Data Science Volume 2, 339–356 (2024)
DOI: https://doi.org/10.51387/23-NEJSDS49

Sparse Estimation in Finite Mixture of Accelerated Failure Time and Mixture of Regression Models with R Package fmrs

Farhad SHOKOOHI

Abstract

Variable selection in large-dimensional data has been extensively studied in different settings over the past decades. In a recent article, Shokoohi et. al. [29, DOI:10.1214/18-AOAS1198] proposed a method for variable selection in finite mixture of accelerated failure time regression models for studies on time-to-event data to capture heterogeneity within the population and account for censoring. In this paper, we introduce the **fmrs** package, which implements the variable selection methodology for such models. Furthermore, as a byproduct, the **fmrs** package facilitates variable selection in finite mixture regression models. The package also incorporates a tuning parameter selection mechanism based on component-wise BIC. Commonly used penalties, such as Least Absolute Shrinkage and Selection Operator, and Smoothly Clipped Absolute Deviation, are integrated into **fmrs**. Additionally, the package offers an option for non-mixture regression models. The C language is chosen to boost the optimization speed. We provide an overview of the **fmrs** principles and the strategies employed for optimization. Hands-on illustrations are presented to help users get acquainted with **fmrs**. Finally, we apply **fmrs** to a lung cancer dataset and observe that a two-component mixture model reveals a subgroup with a more aggressive form of the disease, displaying a lower survival time.

KEYWORDS AND PHRASES: Feature selection, Penalized likelihood, The EM algorithm, Survival model, Lung Cancer.

1. INTRODUCTION

Over the last two decades, the variable selection problem has been the center of attention in many research areas due to the surge of high-dimensional data resulting from advances in modern technologies. A recent study in epigenetics by The Cancer Genome Atlas network on the relationship between the survival time of ovarian cancer patients and their DNA methylation profile of genomic features, for instance, has a complex structure with 396 observations and at least 9,000 covariates. Approximately 28% of the data are right censored and there are some signs of heterogeneity. These data motivated the development of new methodologies for capturing possible heterogeneity while accounting for right censoring in the finite mixture of accelerated failure time regression (FMAFTR) models [29]. Because of the novelty of this research, there was no software package to be used in such situations. Thus, the **fmrs** package aims to provide a tool for variable selection and estimation in FMAFTR. As a byproduct, variable selection in the finite mixture of regression (FMR) models [24] can be carried out using this package. This is because the likelihood of FMR is proportional to that of FMAFTR when all observations are actually failure times. For ease of reference, we use the term "FMRs" to refer to both FMAFTR and FMR hereafter, hence the name of the **fmrs** package.

There are several packages that focus on estimation and inference in finite mixture models. The **mixtools** package [2] provides a collection of R functions for fitting univariate and multivariate finite mixture models, primarily focusing on two-component mixture models without covariates. It covers parametric, nonparametric, and Bayesian approaches in mixture models. The **EMMIXuskew** package [21] estimates the parameters of finite mixtures of unrestricted multivariate skew-t distributions. The **mixsmsn** package [26] estimates the parameters of finite mixture models with components belonging to the class of scale mixtures of the skew-Normal distribution. The **FlexMix** package [13] performs parametric inference in finite mixture models, including concomitant variable models and varying and constant parameters for the component-specific generalized linear regression models. The **CAMAN** package [28] focuses on the analysis of finite semiparametric mixtures. The **CensMix** package [27] employs parameter estimation in censored linear regression models, where random errors follow a finite mixture of Normal or Student-t distributions.

For Bayesian approaches to fitting mixture models, there are several packages available, such as **BayesMixSurv** [23], **BayesH** [30], **BayesCR** [11], **Ultimixt** [18], and **CUB** [15], among others.

Some packages focusing on model-based clustering, unsupervised, supervised, and semi-supervised classification include **mclust** [10], **GMCM** [5], and **Rmixmod** [20]. The **GLDEX** package [32] considers fitting the mixture of generalized Lambda distributions. The **MitISEM** package [1] analyzes data assuming a mixture of Student-t distribu-

tions using the Importance Sampling weighted expectation–maximization (EM) algorithm. The **MixGHD** [33] deals with the mixture of generalized Hyperbolic distributions, providing results for model-based clustering, classification, and discriminant analysis. When dealing with missing values, the **MixAll** package [16] offers algorithms and methods for fitting parametric mixture models to mixed data. The **SMNCensReg** package [12] implements right, left, or interval-censored regression models under the family of scale mixture of Normal distributions, including Normal, Student-t, Pearson VII, Slash, or Contaminated Normal.

In addition to mixture models, there are approaches focusing on AFT regression and semi-parametric survival models. For example, authors in [17] reviewed a semi-parametric AFT model for the analysis of right censored data, and the **spsurv** package [25] provides tools for semi-parametric survival analysis.

Lastly, for AFT models with unspecified error distribution, the **aftgee** package [8] is available and can provide robust solutions when parameter interpretability is not the main concern.

For a comprehensive list of packages that focus on cluster analysis and finite mixture models, refer to https://CRAN.R-project.org/view=Cluster.

To the best of our knowledge, there is currently no package available for variable selection in finite mixture of accelerated failure time regression models. The only package that focuses on variable selection in FMR models, although without censoring, is **fmrlasso** by [31]. However, this package is limited to the Least Absolute Shrinkage and Selection Operator (LASSO) penalty and common tuning parameter. The package is implemented using R as the base code and functions. It is worth noting that packages written in R are often, if not always, less computationally efficient compared to those written in C for the same purpose. This inefficiency becomes more pronounced when analyzing large datasets.

There are several reasons why we have chosen to develop another software package for mixture models. Firstly, most of the previously mentioned packages primarily focus on non-regression mixture models. Secondly, with the exception of **fmrlasso**, none of them provide variable selection capabilities specifically for FMRs. Thirdly, none of the existing implementations address the case when the data are censored. Apart from these reasons, many of the mentioned packages are developed for specific applications and lack the flexibility to choose different subsets of variables for different components of the mixture model. Our package has been designed to address this limitation, allowing the inclusion of pre-specified subsets of covariates in the model. Additionally, our package offers the option for non-mixture regression as well.

To ensure standardized objects, the **fmrs** package is designed using S4 classes and methods [6, 7] and provides standard outputs. While S3 is simpler and easier to handle, S4 is a formal object-oriented system that allows for dispatching on multiple arguments and provides formal class definitions with helper functions for defining generics and methods. As all the base functions are implemented in C, the **fmrs** package offers reasonable speed. In this paper, all computations were performed using version 2.0.1 of the **fmrs** package and version 4.3.0 of R. Updates and future releases of the latest version of the **fmrs** package will be available on the *Bioconductor - Open Source Software for Bioinformatics*, at https://bioconductor.org/packages/fmrs/. An up-to-date version of this paper is also included as a vignette within the package.

The remainder of this paper is organized as follows. In the subsequent sections, we provide the theoretical background of FMRs, including estimation and variable selection methods. We catalog the functions, penalties, distributions, as well as a comprehensive list of arguments and controls implemented in the package. Furthermore, we offer illustrative guidelines on how to use the **fmrs** package by employing simulated datasets and comparing it with competing methods. We then present a real data analysis conducted on patients with lung cancer. Finally, we provide concluding remarks and outline the roadmap for future extensions of the **fmrs** package.

2. MODELS AND METHODS

We consider sparse estimation, i.e., estimation and variable selection, in two families of mixture models: FMAFTR and FMR. We briefly describe each of these models and then present maximum likelihood estimation (MLE) and maximum penalized likelihood estimation (MPLE).

2.1 Finite Mixture of Accelerated Failure Time Regression Models

Let X be the survival time with support $\mathcal{X} \subset \mathbb{R}^+$ and let $\boldsymbol{Z} = (Z_1, \ldots, Z_d)^\top \in \mathbb{R}^d$ be a vector of covariates that may have an effect on X. Define $T = \min\{Y, C\}$, where $Y = \log X$ and C is the logarithm of the censoring time, which is assumed to be noninformative and independent of X. Additionally, we use δ to denote the censoring indicator, i.e., $\delta = 0$ if the time is censored. It is important to note that we do not directly observe X (or equivalently Y); instead, the observed data are (T, δ).

We say $V = (T, \delta, \boldsymbol{Z})$ follows a finite mixture of AFT regression models of order K if the conditional density of (T, δ) given $\boldsymbol{Z} = \boldsymbol{z}$ has the form:

$$f^*(t, \delta; \boldsymbol{z}, \boldsymbol{\Psi}) = \sum_{k=1}^{K} \pi_k [f(t; \theta_k(\boldsymbol{z}), \sigma_k)]^{\delta} [S(t; \theta_k(\boldsymbol{z}), \sigma_k)]^{1-\delta} \times [f_C(t)]^{1-\delta} [S_C(t)]^{\delta}. \quad (2.1)$$

Here, the π_ks ($0 < \pi_k < 1$, with $\sum_{k=1}^{K} \pi_k = 1$) are the mixing probabilities, and $f_C(.)$ and $S_C(.)$ are the density and survival functions of C, respectively. Note that $f(.)$

and $S(.)$ are the density and survival functions of Y, where $\theta_k(\boldsymbol{z}) = h(\beta_{0k} + \boldsymbol{z}^\top \boldsymbol{\beta}_k)$. In this equation, $h(.)$ is a known link function, β_{0k} and $\boldsymbol{\beta}_k = (\beta_{k1}, \beta_{k2}, \ldots, \beta_{kd})^\top$ are the intercept and regression coefficients, respectively, and σ_k is a dispersion parameter.

It is worth noting that for each component of the mixture specified in (2.1), say the kth component, we have:

$$Y = \log X = h(\beta_{0k} + \boldsymbol{z}^\top \boldsymbol{\beta}_k) = \beta_{0k} + \boldsymbol{z}^\top \boldsymbol{\beta}_k + \sigma_k \epsilon.$$

Here, ϵ has a suitable distribution such as standard normal, extreme value, generalized extreme value, or logistic. A common AFT model in survival analysis is based on the Log-Normal distribution [19] in which $\epsilon \sim N(0,1)$.

The vector of all parameters is:

$$\boldsymbol{\Psi} = (\beta_{01}, \ldots, \beta_{0K}, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K, \sigma_1, \ldots, \sigma_K, \pi_1, \ldots, \pi_{K-1})^\top,$$

which has a length of $d^* = K(d+3) - 1$, increasing with the order of the mixture.

Under the assumption of noninformative censoring,

$$f^*(t, \delta; \boldsymbol{z}, \boldsymbol{\Psi}) \propto \sum_{k=1}^{K} \pi_k [f(t; \theta_k(\boldsymbol{z}), \sigma_k)]^{\delta} [S(t; \theta_k(\boldsymbol{z}), \sigma_k)]^{1-\delta}. \tag{2.2}$$

2.2 Finite Mixture of Regression Models

Let Y be the response variable, and let $\boldsymbol{Z} = (Z_1, \ldots, Z_d)^\top$ be a vector of covariates that may have an effect on Y. We say that $(Y, \boldsymbol{Z})$ follows an FMR model of order K [24] if the conditional density of Y given $\boldsymbol{z}$ has the form:

$$f(y; \boldsymbol{z}, \boldsymbol{\Psi}) = \sum_{k=1}^{K} \pi_k f(y; \theta_k(\boldsymbol{z}), \sigma_k). \tag{2.3}$$

It should be noted that for a given sample when all the observations are failure times, i.e., there is no censoring ($\delta_i = 1$), the density $f^*(t, \delta; \boldsymbol{z}, \boldsymbol{\Psi})$ in (2.2) is proportional to the density of a finite mixture of regression models. Therefore,

$$f^*(y; \boldsymbol{z}, \boldsymbol{\Psi}) \propto f^*(t, \delta = 1; \boldsymbol{z}, \boldsymbol{\Psi}). \tag{2.4}$$

As a result, the MLE of parameters of the FMR model is the same if we use the finite mixture of AFT regression model with no censoring. Therefore, the **fmrs** package can also be used to analyze data using the FMR model.

3. MAXIMUM LIKELIHOOD ESTIMATION IN FMRS

The log-likelihood of FMRs is given as

$$\ell_n(\boldsymbol{\Psi}) = \sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k \left[f(t_i; \theta_k(\boldsymbol{z}_i), \sigma_k)\right]^{\delta_i} \left[S(t_i; \theta_k(\boldsymbol{z}_i), \sigma_k)\right]^{1-\delta_i}. \tag{3.1}$$

The expectation–maximization (EM) algorithm is often used in the mixture of distributions to estimate model parameters. The complete log-likelihood function is then given as

$$\ell_n^c(\boldsymbol{\Psi}) = \sum_{i=1}^{n} \sum_{k=1}^{K} u_{ik} \left[\log \pi_k + \log \left\{ [f(t_i; \theta_k(\boldsymbol{z}_i), \sigma_k)]^{\delta_i} [S(t_i; \theta_k(\boldsymbol{z}_i), \sigma_k)]^{1-\delta_i} \right\}\right],$$

where u_{ik} is the latent variable indicating the membership of the ith individual to kth component of FMRs [29]. Having established the complete log-likelihood function, we follow E- and M-step.

In the E-step, $\tau_{ik}^{(m)} = E\left[u_{ik} | \boldsymbol{\Psi}^{(m)}, V_1, \ldots, V_n\right]$ is the conditional expectation of the unobserved variable u_{ik}, where $V_i = (T_i, \delta_i, \boldsymbol{Z}_i)$, and is computed as

$$\tau_{ik}^{(m)} = \frac{\pi_k^{(m)} \left[f(t_i; \theta_k^{(m)}(\boldsymbol{z}_i), \sigma_k^{(m)})\right]^{\delta_i} \left[S(t_i; \theta_k^{(m)}(\boldsymbol{z}_i), \sigma_k^{(m)})\right]^{1-\delta_i}}{\sum_{j=1}^{K} \pi_j^{(m)} \left[f(t_i; \theta_j^{(m)}(\boldsymbol{z}_i), \sigma_j^{(m)})\right]^{\delta_i} \left[S(t_i; \theta_j^{(m)}(\boldsymbol{z}_i), \sigma_j^{(m)})\right]^{1-\delta_i}}. \tag{3.2}$$

The M-step follows by maximizing $Q(\boldsymbol{\Psi}; \boldsymbol{\Psi}^{(m)})$ using the updated $\pi_k^{(m)} = \sum_{i=1}^{n} \tau_{ik}^{(m)} / n$, where

$$\begin{aligned} Q(\boldsymbol{\Psi}; \boldsymbol{\Psi}^{(m)}) = & \sum_{i=1}^{n} \sum_{k=1}^{K} \tau_{ik}^{(m)} \log \pi_k^{(m)} \\ & + \sum_{i=1}^{n} \sum_{k=1}^{K} \tau_{ik}^{(m)} \Big[\delta_i \log f(t_i; \theta_k(\boldsymbol{z}), \sigma_k) \\ & + (1 - \delta_i) \log S(t_i; \theta_k(\boldsymbol{z}), \sigma_k)\Big]. \end{aligned} \tag{3.3}$$

Depending on the form of the sub-distribution, a numerical method may be required to obtain the updated estimates of $\boldsymbol{\Psi}$. For the mixture of Log-Normal AFT and the mixture of Normal models, the M-step of the algorithm has a closed-form solution, as developed below.

3.1 M-Step for Maximizing $Q(.)$ Function in the Mixture of Normal and Mixture of AFT Log-Normal Distributions

Let $\boldsymbol{\tau}_k^{(m)} = \text{diag}\left\{\tau_{ik}^{(m)} : i = 1, \ldots, n\right\}$, $k = 1, \ldots, K$. Denote the pseudo-survival times as:

$$t_{ik}^{(m)} = \delta_i t_i + (1 - \delta_i) \left\{ \boldsymbol{z}_i^\top \boldsymbol{\beta}_k^{(m)} + \sigma_k^{(m)} A(\omega_{ik}^{(m)}) \right\}, \tag{3.4}$$

where $\omega_{ik}^{(m)} = (t_i - \boldsymbol{z}_i^\top \boldsymbol{\beta}_k^{(m)}) / \sigma_k^{(m)}$, $A(\omega) = \phi(\omega)/(1 - \Phi(\omega))$, and $\phi(.)$ and $\Phi(.)$ are the density and cumulative distribution functions of $N(0,1)$, respectively. For each $k = 1, 2, \ldots, K$, let $\boldsymbol{T}_k^{(m)} = (t_{1k}^{(m)}, t_{2k}^{(m)}, \ldots, t_{nk}^{(m)})^\top$, and let $\mathcal{Z} = (\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n)^\top$ be the $n \times d$ dimensional design matrix. The updated estimates of the parameters for the mixture of Log-Normal AFT models (when t is actually the logarithm of t) and the mixture of Normal models for $k = 1, 2, \ldots, K$ are given by

$$\boldsymbol{\beta}_k^{(m+1)} = \left(\mathcal{Z}^\top \boldsymbol{\tau}_k^{(m)} \mathcal{Z}\right)^{-1} \mathcal{Z}^\top \boldsymbol{\tau}_k^{(m)} \boldsymbol{T}_k^{(m)}, \tag{3.5}$$

and

$$\sigma_k^{(m+1)} = \sqrt{\frac{\sum_{i=1}^n \tau_{ik}^{(m)}(t_{ik}^{(m)} - \boldsymbol{z}_i^\top \boldsymbol{\beta}_k^{(m)})^2}{\sum_{i=1}^n \tau_{ik}^{(m)}\Big[\delta_i + (1-\delta_i)\{A(\omega_{ik}^{(m)})[A(\omega_{ik}^{(m)}) - \omega_{ik}^{(m)}]\}\Big]}}. \tag{3.6}$$

3.2 M-Step for Maximizing $Q(.)$ Function in Mixture of AFT Weibull Distributions

There is no closed-form solution for parameter estimation in the Weibull distribution. Hence, we use a numerical method such as the Newton-Raphson algorithm [29]. The iterative algorithm is basically given as

$$\boldsymbol{\beta}^{\text{new}} = \boldsymbol{\beta}^{\text{old}} - I^{-1}(\boldsymbol{\beta}^{\text{old}})\, U(\boldsymbol{\beta}^{\text{old}}),$$

where U and I are the first and second derivative functions of the log-likelihood, respectively, evaluated at $\boldsymbol{\beta}^{\text{old}}$.

Let Y^* follow a Weibull distribution. Then $y = \log y^*$ follows the extreme value distribution with the probability density function (pdf) and cumulative distribution function (cdf) given by

$$f(y; \boldsymbol{x}^\top\boldsymbol{\beta}, \sigma) = \frac{1}{\sigma}\exp\left(\frac{y - \boldsymbol{x}^\top\boldsymbol{\beta}}{\sigma}\right)\exp\left(-\exp\left(\frac{y - \boldsymbol{x}^\top\boldsymbol{\beta}}{\sigma}\right)\right)$$

and

$$F(y; \boldsymbol{x}^\top\boldsymbol{\beta}, \sigma) = 1 - \exp\left(-\exp\left(\frac{y - \boldsymbol{x}^\top\boldsymbol{\beta}}{\sigma}\right)\right),$$

respectively. For the kth component, we have

$$U(\beta_{0k}; \boldsymbol{\beta}_k^{\text{old}}, \boldsymbol{y}) = -\sum_{i=1}^n \frac{t_{ik}^{\text{old}}\delta_i}{\sigma_k^{\text{old}}} + \sum_{i=1}^n \frac{t_{ik}^{\text{old}}}{\sigma_k^{\text{old}}} e^{\left(\frac{y_i - \boldsymbol{x}_i^\top\boldsymbol{\beta}_k^{\text{old}}}{\sigma_k^{\text{old}}}\right)},$$

and

$$U(\beta_{jk}; \boldsymbol{\beta}_k^{\text{old}}, \boldsymbol{y}) = -\sum_{i=1}^n \frac{t_{ik}^{\text{old}}\delta_i}{\sigma_k^{\text{old}}} x_{ij} + \sum_{i=1}^n \frac{t_{ik}^{\text{old}}}{\sigma_k^{\text{old}}} x_{ij} e^{\left(\frac{y_i - \boldsymbol{x}_i^\top\boldsymbol{\beta}_k^{\text{old}}}{\sigma_k^{\text{old}}}\right)},$$

for $j = 1, \ldots, d$. For the second derivatives, we have

$$I(\beta_{jk}, \beta_{rk}; \boldsymbol{\beta}_k^{\text{old}}, \boldsymbol{y}) = -\sum_{i=1}^n \frac{t_{ik}^{\text{old}}}{\sigma_k^{\text{old}}} x_{ij}x_{ir}\exp\left(\frac{y_i - \boldsymbol{x}_i^\top\boldsymbol{\beta}_k^{\text{old}}}{\sigma_k^{\text{old}}}\right)$$

and

$$I(\beta_{jk}, \beta_{0k}; \boldsymbol{\beta}_k^{\text{old}}, \boldsymbol{y}) = -\sum_{i=1}^n \frac{t_{ik}^{\text{old}}}{\sigma_k^{\text{old}}} x_{ij}\exp\left(\frac{y_i - \boldsymbol{x}_i^\top\boldsymbol{\beta}_k^{\text{old}}}{\sigma_k^{\text{old}}}\right),$$

for $j, r = 1, \ldots, d$, and

$$I(\beta_{0k}, \beta_{0k}; \boldsymbol{\beta}_k^{\text{old}}, \boldsymbol{y}) = -\sum_{i=1}^n \frac{t_{ik}^{\text{old}}}{\sigma_k^{\text{old}}}\exp\left(\frac{y_i - \boldsymbol{x}_i^\top\boldsymbol{\beta}_k^{\text{old}}}{\sigma_k^{\text{old}}}\right).$$

Algorithm 1 in Appendix A provides instructions for obtaining MLEs and Ridge estimators.

It is widely acknowledged that finite mixture models, specifically finite mixtures of regression models, are identifiable up to a permutation [14, 9]. As a consequence, in practical applications, it is common for the estimated components to deviate from the order of the simulation setup or the true order in the population (which remains unknown). To establish the correct order, it becomes necessary to possess some knowledge about the regression coefficients' locations and select initial values accordingly. When analyzing real data, one possible approach to rearranging the components is to consider the order of their grand means.

4. VARIABLE SELECTION IN FMRS

Having the current estimates $\boldsymbol{\Psi}^{(m)}$, the penalized $Q(.)$ function is given as

$$\begin{aligned} Q(\boldsymbol{\Psi}; \boldsymbol{\Psi}^{(m)}) = &\sum\nolimits_{i=1}^n \sum\nolimits_{k=1}^K \tau_{ik}^{(m)} \log \pi_k^{(m)} \\ &+ \sum\nolimits_{i=1}^n \sum\nolimits_{k=1}^K \tau_{ik}^{(m)}\Big[\delta_i \log f_Y(t_i; \theta_k(\boldsymbol{z}), \sigma_k) \\ &+ (1-\delta_i)\log S_Y(t_i; \theta_k(\boldsymbol{z}), \sigma_k)\Big] - \mathbf{p}_{\boldsymbol{\lambda}_n}(\boldsymbol{\Psi}), \end{aligned} \tag{4.1}$$

where the penalty is replaced by the local quadratic approximation

$$\mathbf{p}_{\boldsymbol{\lambda}_n}(\boldsymbol{\Psi}) \approx \mathbf{p}_{\boldsymbol{\lambda}_n}(\boldsymbol{\Psi}; \boldsymbol{\Psi}^{(m)}) = n\sum_{k=1}^K \pi_k^{(m)} \sum_{j=1}^d \Big\{ p_{\lambda_{n,k}}(\beta_{kj}^{(m)}) + \frac{p'_{\lambda_{n,k}}(\beta_{kj}^{(m)})}{2\beta_{kj}^{(m)}}\left(\beta_{kj}^2 - (\beta_{kj}^{(m)})^2\right)\Big\}.$$

The maximum penalized likelihood estimator (MPLE), when the sub-distributions are log-normal, is then given as

$$\boldsymbol{\beta}_k^{(m+1)} = \left(\mathcal{Z}^\top \boldsymbol{\tau}_k^{(m)} \mathcal{Z} + \boldsymbol{\Sigma}_k(\boldsymbol{\beta}_k^{(m)})\right)^{-1} \mathcal{Z}^\top \boldsymbol{\tau}_k^{(m)} \boldsymbol{T}_k^{(m)}, \tag{4.2}$$

where

$$\boldsymbol{\Sigma}_k(\boldsymbol{\beta}_k^{(m)}) = \text{diag}\Big\{ n\pi_k^{(m+1)} p'_{\lambda_n,k}(\beta_{kj}^{(m)})/\beta_{kj}^{(m)} : j = 1, 2, \ldots, d\Big\}$$

and $\sigma_k^{(m+1)}$ is equal to (3.6) when replacing (4.2) in (3.6). Having specified a threshold, the elements of $\boldsymbol{\beta}_k^{(m+1)}$ that fall below the threshold will be set to zero, which leads to variable selection [29]. Note that for Weibull, the NR algorithm must be carried out in the M-step of the EM algorithm. Algorithm 3 in Appendix A provides instructions to obtain MPLEs.

It is known that the fitted model of FMRs depends on the choice of initial values. Furthermore, it is argued that the MLE could be a set of good initial values to obtain MPLEs. The following penalty functions are implemented in the **fmrs** package:

- LASSO: $\frac{p_n(\theta;\lambda)}{n^2} = \lambda|\theta|$;
- adaptive LASSO: $\frac{p_n(\theta;\lambda)}{n^2} = \lambda w|\theta|$, for some (possibly random) known weights w;
- The Minimax Concave Penalty (MCP): $\frac{p_n'(\theta;\lambda)}{n^2} = \text{sgn}(\theta)\ \frac{(a\lambda-|\theta|)_+}{a}$;
- Smoothly Clipped Absolute Deviation (SCAD): $\frac{p_n'(\theta;\lambda)}{n^2} = \text{sgn}(\theta)\left\{\lambda I(|\theta| \le \lambda) + \frac{(a\lambda-|\theta|)_+}{a-1} I(|\theta| > \lambda)\right\}$,

where $p_n'(\cdot;\lambda)$ is the first derivative of penalty with respect to θ and $(x)_+ = \max\{0, x\}$.

4.1 Choice of Tuning Parameters

Two approaches are available for choosing tuning parameters: the common approach and the component-wise approach. If the common tuning parameter approach is adopted, one can choose the value that minimizes the BIC from a set of candidates within the interval $(0, \lambda_{\max}]$, where $\lambda_{\max}$ is a pre-specified value. This approach is suitable for datasets with sufficiently large sample sizes, and it reduces the computational burden when a common tuning parameter is used.

On the other hand, if we adopt the component-wise approach, we will search for the optimal values $(\lambda_1, \ldots, \lambda_K)$ from a set of candidate values derived from the interval $(0, \lambda_{\max}]$. We choose the combination that minimizes the component-wise BIC, which is defined below.

Let $\tilde{\tau}_{ik}$ be the MLE in (3.2). For a given k, the component-wise log-likelihood is defined as

$$\tilde{\ell}_{nk}(\boldsymbol{\Psi}_k) = \sum_{i=1}^n \tilde{\tau}_{ik} \log\Big\{[f_Y(t_i, \theta_k(\boldsymbol{z}_i), \sigma_k)]^{\delta_i} [S_Y(t_i, \theta_k(\boldsymbol{z}_i), \sigma_k)]^{1-\delta_i}\Big\}. \tag{4.3}$$

Let $\widehat{\boldsymbol{\Psi}}_k(\lambda_k)$ be the MPLE under (4.3) for a given λ_k; i.e.,

$$\widehat{\boldsymbol{\Psi}}_k(\lambda_k) = \arg\max_{\boldsymbol{\Psi}_k}\left[\tilde{\ell}_{nk}(\boldsymbol{\Psi}_k) - n\pi_k \sum_{j=1}^d p_{\lambda_k}(|\beta_{jk}|)\right].$$

We define the component-wise BIC as

$$\text{BIC}_k(\lambda_k) = -2\tilde{\ell}_{nk}(\widehat{\boldsymbol{\Psi}}_k(\lambda_k)) + \text{DF}(\lambda_k) \times \log n_k, \tag{4.4}$$

where $n_k = \sum_{i=1}^n \tilde{\tau}_{ik}$, $\text{DF}(\lambda_k) = \sum_{j=1}^d I(\hat{\beta}_{kj} \neq 0)$ is the number of estimated non-zero coefficients, and $\tilde{\ell}_{nk}$ is evaluated at $\widehat{\boldsymbol{\Psi}}_k(\lambda_k)$. The component-wise tuning parameter is then chosen as

$$\hat{\lambda}_k = \arg\min_{\lambda_k \in (0, \lambda_{\max}]} \text{BIC}_k(\lambda_k), \text{ for } k = 1, \ldots, K.$$

By choosing a grid of values in the interval $(0, \lambda_{\max}]$ with a length of L, the total number of searches required to find K tuning parameters is reduced to $K \times L$. In contrast, using the non-component-wise approach would require L^K searches to find K tuning parameters. By adopting the component-wise approach, we significantly reduce the number of searches. It is important to note that although this approach has not been theoretically studied, our simulations have shown promising results. Algorithm 2 in Appendix A provides instructions for obtaining the tuning parameters.

4.2 Choice of Mixture Order

So far, we assumed that the order of FMRs is known a priori. However, in many real applications, such information is not available. For order selection in mixture model setups, information criteria such as BIC have been extensively studied [24, 14]. In **fmrs**, we suggest using BIC for order selection. Consider the possible values $K \in \{1, \ldots, K_{\max}\}$ for the order of mixture, where $K_{\max}$ is a pre-specified upper bound. The optimal order, denoted as $\hat{K}$, is chosen as the one that minimizes the quantity

$$\text{BIC}^*(K) = -2\ell_n(\widehat{\boldsymbol{\Psi}}_{n,K}) + \left[(3K-1) + \sum_{k=1}^K \sum_{j=1}^d \text{I}(\hat{\beta}_{jk} \neq 0)\right] \log n,$$

where ℓ_n in (3.1) is evaluated at $\widehat{\boldsymbol{\Psi}}_{n,K}$, the vector of estimated parameters with order K. Note that this approach has yet to be theoretically studied. In simulation studies, however, promising results are observed.

5. PRELIMINARIES AND MAIN FUNCTIONS

In the following subsections, we present comprehensive lists of all available arguments and provide detailed descriptions of the main functions in the **fmrs** package.

5.1 Preliminaries

The command 'help(package = "fmrs")' returns a list of all available arguments in **fmrs**. The basic functions are implemented in the C programming language, which greatly improves memory management and package speed. The package utilizes S4 objects and methods. Table 1 presents a list of all S4 classes along with their descriptions. Table 2 provides a comprehensive list of S4 methods and functions, including their arguments and associated default values. Table 3 displays a list of arguments, controls, and notations, along with their default values and descriptions. Currently, the FMAFTR supports Log-Normal and Weibull distributions, while the FMR includes the Normal distribution in the package.

5.2 Description of the Main Functions

The **fmrs** package has two main classes and four main functions. The classes are as follows:

- The `fmrsfit` class

The class `fmrsfit` is designed to store fitted models obtained through MLE and MPLE using the functions `fmrs.mle` and `fmrs.varsel`. It stores various information such as the

*Table 1. List of S4 classes in the **fmrs** package.*

class	Slot	Value	Description
`fmrsfit`	`y`	A length-`nobs` numeric vector	Response vector
	`delta`	A length-`nobs` numeric vector	Censoring vector
	`x`	A dimension-`nobs-ncov` numeric matrix	Covariates
	`nobs`	A length-one numeric vector	Number of observations
	`ncov`	A length-one numeric vector	Number of covariates
	`ncomp`	A length-one numeric vector	Order of the mixture
	`coefficients`	A length-(`ncov`+1)-`ncomp` numeric matrix	Fitted regression coefficients
	`dispersion`	A length-`ncomp` numeric vector	Fitted dispersions
	`mixProp`	A length-`ncomp` numeric vector	Fitted mixing proportions
	`logLik`	A length-one numeric vector	Log-likelihood
	`BIC`	A length-one numeric vector	Bayesian information criteria
	`nIterEMconv`	A length-one numeric vector	Number of the EM algorithm iterations
	`disFamily`	A length-one character vector	Name of sub-distributions
	`penFamily`	A length-one character vector	Penalty function
	`lambPen`	A length-`ncomp` numeric vector	Tuning parameters
	`lamRidge`	A length-one numeric vector	Ridge tuning parameter
	`MCPGam`	A length-one numeric vector	MCP's extra tuning parameter
	`SICAGam`	A length-one numeric vector	SICA's extra tuning parameter
	`model`	A length-one character vector	Fitted model
	`fitted`	A dimension-`nobs-ncomp` numeric matrix	Predicted values
	`residuals`	A dimension-`nobs-ncomp` numeric matrix	Predicted residuals
	`weights`	A dimension-`nobs-ncomp` numeric matrix	Predicted weights
	`activeset`	A dimension-(`ncov+1`)-`ncomp` matrix	Parameters that must be active in model
	`selectedset`	A dimension-(`ncov`)-`ncomp` matrix	Parameters selected via variable selection
`fmrstunpar`	`ncomp`	A length-one numeric vector	Order of mixture
	`ncov`	A length-one numeric vector	Number of covariates
	`lambPen`	A length-`ncomp` numeric vector	Tuning parameters
	`MCPGam`	A length-`ncomp` numeric vector	MCP's extra tuning parameters
	`SICAGam`	A length-`ncomp` numeric vector	SICA's extra tuning parameters
	`disFamily`	A length-one character vector	Name of sub-distributions
	`penFamily`	A length-one character vector	Penalty function
	`lamRidge`	A length-one numeric vector	Ridge tuning parameter
	`model`	A length-one character vector	Fitted model
	`activeset`	A dimension-(`ncov+1`)-`ncomp` matrix	Parameters that must be active in model

*Table 2. List of S4 methods and functions in the **fmrs** package.*

Generic Name	Description
`fmrs.gendata`	Generates a dataset from `FMRs` under the specified setting.
`fmrs.mle`	Performs MLE and ridge regression for `FMRs`.
`fmrs.tunsel`	Computes component-wise tuning parameters based on a BIC for `FMRs`.
`fmrs.varsel`	Performs variable selection and computes penalized MLE for `FMRs`.
`BIC`	Provides the estimated BIC of an `FMRs` from an `fmrsfit-class`
`coefficients`	Provides the estimated regression coefficients from the `FMRs` from an `fmrsfit-class`
`dispersion`	Provides the estimated dispersions of the fitted `FMRs` from an `fmrsfit-class`
`fitted`	Provides the fitted response of the fitted `FMRs` from an `fmrsfit-class`
`logLik`	Provides the estimated logLikelihood of an `FMRs` from an `fmrsfit-class`
`mixProp`	Provides the estimated mixing proportions of an `FMRs` from an `fmrsfit-class`
`ncomp`	Provides the order of an `FMRs` from an `fmrsfit-class`
`ncov`	Provides the number of covariates of an `FMRs` from an `fmrsfit-class`
`nobs`	Provides the number of observations in an `FMRs` from an `fmrsfit-class`
`residuals`	Provides the residuals of the fitted `FMRs` from an `fmrsfit-class`
`summary`	Displays estimated coefficients, dispersions, and mixing proportions or selected tuning parameters
`weights`	Provides the weights of fitted observations for each observation under all components of an `FMRs` model

Table 3. List of arguments, controls and notations in the ***fmrs*** *package.*

Name	Default	Description
Arguments		
`y`		Response vector
`delta`		Censoring indicator vector
`x`		Design matrix (covariates)
`nObs`		A numeric value represents the number of observations (sample size)
`nComp`	2	A numeric value represents the order of mixture in `FMRs`
`nCov`		A numeric value represents the number of covariates in design matrix
`mixProp`		A vector of mixing proportions which their sum must be one
`rho`		A numeric value in [-1, 1] which represents the correlation between covariates of design matrix
`dispersion`		A vector of positive values for dispersion parameters of sub-distributions in `FMRs`
`coeff`		A vector of length `nComp`*(`nCov+1`) of all regression coefficients including intercepts.
`umax`		A numeric value that represents the upper bound in Uniform distribution for censoring
`disFamily`	`"lnorm"`	A sub-distribution family. The choices are `"norm"` for FMR, `"lnorm"` for FMAFTR with Log-Normal sub-distribution,`"weibull"` for FMAFTR with Weibull sub-distribution.
Controls		
`conveps`	`1e-08`	A positive number for avoiding `NaN` in computing divisions
`convepsEM`	`1e-08`	A positive value for threshold of convergence in the EM algorithm
`convepsNR`	`1e-08`	A positive value for threshold of convergence in the Newton-Raphson algorithm
`gamMixPor`	`1`	Proportion of mixing parameters in the penalty function. The value must belong to the interval $[0, 1]$. If `gamMixPor = 0`, the penalty structure is no longer a mixture.
`initCoeff`		A vector of initial values for coefficients including intercepts
`initmixProp`		A vector of initial values for the proportion of components
`initDispersion`		A vector of initial values for standard deviations
`lambPen`		A vector of lambda for penalty
`lambRidge`	`0`	Lambda for the ridge penalty or Elastic Net
`lambMCP`		Extra tuning parameter for the MCP penalty
`lambSICA`	`5`	Extra tuning parameter for the SICA penalty
`LambMin`	`0.01`	A positive value for the minimum value of tuning parameters
`LambMax`	`1.0`	A positive value for the maximum value of tuning parameters
`nLamb`	`100`	An integer for the number of tuning parameters between `LambMin` and `LambMax`
`nIterEM`	`400`	Maximum number of iterations for the EM algorithm
`nIterNR`	`2`	Maximum number of iterations for the Newton-Raphson algorithm
`penFamily`	`"lasso"`	The penalty used in variable selection method. The available options are `"lasso"`, `"adplasso"`, `"mcp"`, `"scad"`, `"sica"` and `"hard"`.
`NRpor`	`2`	A positive integer for the maximum number of searches in the Newton-Raphson algorithm
`activeset`		A 0-1 matrix that shows which coefficients must be active in the model. This could be an `oracleset` as well.
`cutpoint`	`0.05`	A positive integer for setting the estimates to zero if they are too small.
Notations		
`FMRs`		Finite Mixture of Regression Models including `FMAFTR` and `FMR`
`FMAFTR`		Finite Mixture of Accelerated Failure Time Regression
`FMR`		Finite Mixture of Regression

response variable, design matrix, censoring information, parameter estimates, fitted values, posterior probabilities, and evaluation criteria like log-likelihood and BIC. The R function `summary` provides a standard output of this information.

- The `fmrstunpar` class

The class `fmrstunpar` is introduced to store tuning parameters obtained using the component-wise BIC approach in (4.4), which can be utilized in the `fmrs.varsel` function.

The package introduces 17 functions, with four of them being the main functions. The arguments for these functions are described in Tables 1-3. Below, we provide a brief introduction to these functions.

- The `fmrs.gendata` function

The R function `fmrs.gendata` is used to simulate a dataset from FMRs. It has the following form:
`fmrs.gendata(nObs, nComp, nCov, coeff, dispersion, mixProp, rho, umax, disFamily, ...)`,
where `nObs`, `nComp`, `nCov`, `coeff`, `dispersion`, `mixProp` and

`disFamily` represent the sample size, the order of FMRs, the number of regression covariates, the regression coefficients, the dispersions of errors, the mixing proportions, and the distribution of components of FMRs, respectively.

It is important to note that `rho` (i.e., ρ) is used in the variance-covariance matrix to simulate the design matrix $\mathbf{X}$ from a multivariate Gaussian distribution with mean $\mathbf{0}$ and variance-covariance matrix $\Sigma_X = (\rho^{|l-m|})$. The right censoring times are generated using a Uniform distribution with lower and upper bounds of 0 and `umax`, respectively. Depending on the choice of `disFamily`, the function `fmrs.gendata` generates a dataset from FMAFTR or FMR. The default value is `disFamily = "lnorm"`. If `disFamily = "norm"`, the function ignores the censoring parameter `umax` and generates a dataset from FMR with Normal sub-distributions. On the other hand, if `disFamily = "lnorm"` or `disFamily = "weibull"`, the function produces a dataset from FMAFTR with Log-Normal or Weibull sub-distribution. Consequently, `fmrs.gendata` returns a list containing a vector of responses `y`, a matrix of covariates `x`, a vector of censoring indicators `delta`, and the name of the sub-distributions of the mixture model.

- The `fmrs.mle` function

The C function `fmrs.mle` returns the MLE for the parameters of FMRs. It has the following form:

```
fmrs.mle(y, delta, x, nComp, disFamily,
initCoeff, initDispersion, initmixProp,
oracleset, ... ).
```

Here, `delta`, `initCoeff`, `initDispersion`, `initmixProp`, and `oracleset` represent the censoring indicators, the initial values for regression coefficients, the dispersions, and the mixing proportions, and the set of oracle covariates that should be included in each component of the mixture model, respectively. The remaining arguments in `fmrs.mle` are controlling parameters.

This function returns a fitted FMRs model that includes the MLE of regression parameters, standard deviations, and mixing proportions based on the EM algorithm. The output also includes the log-likelihood and BIC for the fitted model, the maximum number of iterations used in the EM algorithm, and the type of the fitted FMRs (i.e., `FMAFTR` or `FMR`). To perform Ridge regression, a positive value must be chosen for `lambRidge`, which is the tuning parameter of the Ridge penalty.

The default values for the arguments are as follows: `nComp = 2`, `disFamily = "lnorm"`, `lambRidge = 0`, `nIterEM = 400`, `nIterNR = 2`, `conveps = 1e-08`, `convepsEM = 1e-08`, `convepsNR = 1e-08`, and `NRpor = 2`.

- The `fmrs.tunsel` function

The C function `fmrs.tunsel` is used to search for a data-driven tuning parameter from a selected set of values. It has the following form:

```
fmrs.tunsel(y, delta, x, nComp, disFamily,
initCoeff, initDispersion, initmixProp,
penFamily, lambRidge, oracleset, lambMCP,
lambSICA, LambMin, LambMax, nLamb, ...).
```

Here, `penFamily`, `lambMCP`, and `lambSICA` represent the penalty function, the hyper-parameter for MCP, and the hyper-parameter for SICA penalty. Additionally, `LambMin`, `LambMax`, and `nLamb` specify the minimum, maximum, and the number of tuning parameters used to obtain the optimal tuning parameter based on the component-wise tuning parameter selection approach. The function returns an `fmrstunpar` class.

The default values for the arguments are as follows: `disFamily = "lnorm"`, `penFamily = "lasso"`, `lambRidge = 0`, `nIterEM = 400`, `nIterNR = 2`, `conveps = 1e-08`, `convepsEM = 1e-08`, `convepsNR = 1e-08`, `NRpor = 2`, `gamMixPor = 1`, `cutpoint = 0.05`, `LambMin = 0.01`, `LambMax = 1`, and `nLamb = 100`.

- The `fmrs.varsel` function

The C function `fmrs.varsel` is used to perform variable selection (MPLE) for the parameters of FMRs. It has the following form:

```
fmrs.varsel(y, delta, x, nComp, disFamily,
initCoeff, initDispersion, initmixProp,
penFamily, lambPen, lambRidge, oracleset,
lambMCP, lambSICA, ... ).
```

Here, `lambPen` represents the set of tuning parameters for the penalty function. The function returns an `fmrstfit` class that stores the values of MPLE for the regression parameters.

The default values for the arguments are as follows: `disFamily = "lnorm"`, `penFamily = "lasso"`, `lambRidge = 0`, `nIterEM = 2000`, `nIterNR = 2`, `conveps = 1e-08`, `convepsEM = 1e-08`, `convepsNR = 1e-08`, `NRpor = 2`, `gamMixPor = 1`, and `cutpoint = 0.05`.

- Additional functions

In addition to the main functions, we have introduced several auxiliary functions to extract and report the results obtained from the main functions. These functions are listed in Table 2. One such example is the `summary` function, which summarizes the results of all functions in a standard manner.

6. THE FMRS PACKAGE IN ACTION

6.1 Example 1: FMAFTR Model with Log-Normal Sub-Distributions

In order to illustrate the application of **fmrs**, we begin by generating a dataset from an FMAFT model. It is worth noting that for a comprehensive simulation study, users can refer to [29].

We generate the covariates from a multivariate normal distribution with a dimension of 10 and a sample size of

500. The mean vector is set to $\mathbf{0}$, and the variance-covariance matrix is $\Sigma = (0.25^{|l-m|})$. Subsequently, we simulate time-to-event data from a finite mixture of two components using AFT regression models with Log-Normal sub-distributions. We load the necessary libraries and assign the parameters of the model. The parameter values chosen for this simulation are provided in the following code:

```
> library(fmrs); set.seed(1980)
> myk <- 2; myd <- 10; myn <- 500; myrho <- 0.25
> mydisp <- c(1,1); myu <- 40; mypi <- c(0.4,0.6)
> coeff1 <- c(2,2,-1,-2,1,2,0,0,0,0,0)
> coeff2 <- c(-1,-1,1,2,0,0,0,0,-1,2,-2)
```

One can use `fmrs.gendata` to generate data from an FMAFTR model as follows:

```
> dat <-
+  fmrs.gendata(nObs = myn, nComp = myk,
+  nCov = myd, mixProp = mypi,
+  coeff = c(coeff1, coeff2),
+  dispersion = mydisp, rho = myrho,
+  umax = myu, disFamily = "lnorm")
```

Regrettably, the use of R for random generation produces distinct datasets across various machines and operating systems. Therefore, we have included alternative Python code to ensure reproducibility.

With the simulated dataset in hand, we proceed to estimate the MLEs of the model parameters using the `fmrs.mle` function. It is worth noting that the initial values for the regression parameters are generated from a standard normal distribution.

```
> dat <-
+  list(y = as.numeric(unlist(read.csv(
+  paste0(PathData,'Ydata_1.csv'),header=TRUE))),
+  x = as.matrix(read.csv(
+  paste0(PathData,'Xdata_1.csv'),header=TRUE)),
+  delta = as.numeric(unlist(read.csv(
+  paste0(PathData,'Ddata_1.csv'),header=TRUE))))
> res.mle <-
+  fmrs.mle(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = rnorm(myk * myd + myk),
+  initDispersion = rep(1, myk),
+  initmixProp = rep(1 / myk, myk))
> summary(res.mle)
----------------------
Fitted Model:
----------------------
  Finite Mixture of Accelerated Failure Time
                Regression Models
          Log-Normal Sub-Distributions
  2 Components; 10 Covariates; 500 samples.

Coefficients:
               Comp.1       Comp.2
Intercept  2.02454239 -1.060113719
X0         2.02532034 -0.941695429
X1        -0.96496554  1.021660992
X2        -2.00238318  1.874530674
X3         1.00386599 -0.039067909
X4         2.14597060 -0.073005941
X5         0.12657840 -0.002645082
X6        -0.13008430 -0.006336930
X7         0.09310156 -0.949417514
X8         0.03849414  1.932369807
X9        -0.01741908 -1.964006855

Dispersion:
   Comp.1   Comp.2
 1.019993 0.931589

Mixing Proportions:
    Comp.1    Comp.2
 0.4458544 0.5541456

LogLik: -779.0462; BIC: 1713.458
```

It is evident that the MLEs of the regression coefficients are not equal to zero. As a result, the MLE approach alone cannot provide a sparse solution. To achieve sparsity, we utilize the variable selection method developed by [29]. Once we obtain the MLEs, the next step is to determine a set of suitable tuning parameters. This can be accomplished by employing the component-wise approach implemented in the `fmrs.tunsel` function. However, in certain scenarios, it is worthwhile to explore whether the common tuning parameter approach yields superior results. This can be investigated through data-driven simulation studies, for example.

```
> res.lam <-
+  fmrs.tunsel(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = "adplasso", LambMin = 0.01,
+  LambMax = 1, nLamb = 100)
> summary(res.lam)
----------------------
Selected Tuning Parameters:
----------------------
  Finite Mixture of Accelerated Failure Time
         Regression Models
    Log-Normal Sub-Distributions
  2 Components; adplasso Penalty;

Component-wise lambda:
 Comp.1 Comp.2
 0.0199   0.01
```

We have utilized the MLE estimates as initial values to obtain the tuning parameters. In this phase, the same set of values is employed to conduct variable selection with an adaptive LASSO penalty.

```
> res.var <-
+  fmrs.varsel(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = "adplasso",
+  lambPen = slot(res.lam,"lambPen"))
> summary(res.var)
----------------------
Fitted Model:
----------------------
  Finite Mixture of Accelerated Failure Time
                 Regression Models
           Log-Normal Sub-Distributions
  2 Components; 10 Covariates; 500 samples.

Coefficients:
              Comp.1     Comp.2
Intercept  1.9592133 -1.0601621
X0         1.9748971 -0.9307360
X1        -0.8960131  1.0125806
X2        -1.9616150  1.8569027
X3         0.9668803  0.0000000
X4         2.1228268  0.0000000
X5         0.0000000  0.0000000
X6         0.0000000  0.0000000
X7         0.0000000 -0.9328832
X8         0.0000000  1.9244269
X9         0.0000000 -1.9519150

Selected Set:
   Comp.1 Comp.2
X0      1      1
X1      1      1
X2      1      1
X3      1      0
X4      1      0
X5      0      0
X6      0      0
X7      0      1
X8      0      1
X9      0      1

Dispersion:
   Comp.1    Comp.2
 1.024904 0.9397894

Mixing Proportions:
    Comp.1    Comp.2
 0.4466108 0.5533892

LogLik: -782.0566; BIC: 1663.547
```

6.1.1 Common Tuning Parameters

If we desire to select a common tuning parameter, there is no need to execute any additional functions after `fmrs.mle(.)`. Instead, we can employ a for-loop command to search for the optimal fit and the common tuning parameter using `fmrs.varsel(.)`. The following code demonstrates an example of this process.

```
> set.seed(1980)
> res.mle <-
+  fmrs.mle(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = rnorm(myk * myd + myk),
+  initDispersion = rep(1, myk),
+  initmixProp = rep(1 / myk, myk))
> summary(res.mle)
----------------------
Fitted Model:
----------------------
  Finite Mixture of Accelerated Failure Time
                 Regression Models
           Log-Normal Sub-Distributions
  2 Components; 10 Covariates; 500 samples.

Coefficients:
              Comp.1       Comp.2
Intercept  2.02454239 -1.060113719
X0         2.02532034 -0.941695429
X1        -0.96496554  1.021660992
X2        -2.00238318  1.874530674
X3         1.00386599 -0.039067909
X4         2.14597060 -0.073005941
X5         0.12657840 -0.002645082
X6        -0.13008430 -0.006336930
X7         0.09310156 -0.949417514
X8         0.03849414  1.932369807
X9        -0.01741908 -1.964006855

Dispersion:
   Comp.1   Comp.2
 1.019993 0.931589

Mixing Proportions:
    Comp.1    Comp.2
 0.4458544 0.5541456

LogLik: -779.0462; BIC: 1713.458

> BICmin <- 1e+22
> optLam <- 0
> Ctune <- seq(0.01, 1, length.out = 100)
```

```
> for (j in seq_along(Ctune)) {
+  holdfit <-
+  fmrs.varsel(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = "adplasso",
+  lambPen = rep(Ctune[j], myk))
+  if (BIC(holdfit) < BICmin) {
+  BICmin <- BIC(holdfit)
+  optLam <- Ctune[j]
+  res.var <- holdfit
+  }
+  }
> summary(res.var)
----------------------
Fitted Model:
----------------------
  Finite Mixture of Accelerated Failure Time
                 Regression Models
           Log-Normal Sub-Distributions
  2 Components; 10 Covariates; 500 samples.

Coefficients:
              Comp.1     Comp.2
Intercept  1.9839202 -1.0592957
X0         2.0015958 -0.9319706
X1        -0.9248311  1.0128721
X2        -1.9919941  1.8565197
X3         1.0070240  0.0000000
X4         2.1402970  0.0000000
X5         0.0000000  0.0000000
X6         0.0000000  0.0000000
X7         0.0000000 -0.9320393
X8         0.0000000  1.9249533
X9         0.0000000 -1.9510538

Selected Set:
   Comp.1 Comp.2
X0      1      1
X1      1      1
X2      1      1
X3      1      0
X4      1      0
X5      0      0
X6      0      0
X7      0      1
X8      0      1
X9      0      1

Dispersion:
   Comp.1    Comp.2
 1.025767 0.9402267

Mixing Proportions:
    Comp.1    Comp.2
 0.4460635 0.5539365

LogLik: -781.7193; BIC: 1662.872
> optLam
[1] 0.01
```

6.2 Example 2: FMR Model with Normal Sub-Distributions

As mentioned in Section 2, the MLE and MPLE of an FMR model can be obtained by disregarding the censoring in the FMAFTR setting. We select the following parameters to generate the data from an FMR model.

By specifying `"norm"` for `disFamily` in `fmrs.gendata`, we generate a dataset from an FMR model using the following code:

```
> set.seed(1980)
> dat <-
+  fmrs.gendata(nObs = myn, nComp = myk,
+  umax = myu,coeff = c(coeff1, coeff2),
+  dispersion = mydisp, mixProp = mypi,
+  rho = myrho, disFamily = "norm", nCov = myd)
```

Similar to the above, we use Python to generate the data.

```
> set.seed(1980)
> dat <-
+  list(y = as.numeric(unlist(read.csv(
+  paste0(PathData,'Ydata_2.csv'),header=TRUE))),
+  x = as.matrix(read.csv(
+  paste0(PathData,'Xdata_2.csv'),header=TRUE)))
```

The MLE of the FMR model parameters are obtained as follows:

```
> res.mle <-
+  fmrs.mle(y = dat$y, x = dat$x,
+  disFamily = "norm", nComp = myk,
+  initCoeff = rnorm(myk * myd + myk),
+  initDispersion = rep(1,myk),
+  initmixProp = rep(1 / myk, myk))
> summary(res.mle)
----------------------
Fitted Model:
----------------------
  Finite Mixture of Regression Models
  2 Components; 10 Covariates; 500 samples.

Coefficients:
               Comp.1      Comp.2
Intercept  1.949067724 -1.07600740
X0         1.828278272 -0.95309672
X1        -0.913203579  0.88183494
X2        -2.103336565  2.00573346
```

```
X3          0.761163611  0.04434990
X4          2.085979948 -0.01370548
X5         -0.060527376  0.02220583
X6         -0.203188863 -0.04046472
X7          0.215037392 -0.99520378
X8          0.057290643  2.06305705
X9         -0.002688772 -1.97832175

Dispersion:
    Comp.1    Comp.2
 0.8960443 0.9867858

Mixing Proportions:
    Comp.1    Comp.2
 0.3693712 0.6306288

LogLik: -941.7167; BIC: 2038.799
```

The following code is used in selecting the component-wise tuning parameters:

```
> res.lam <-
+  fmrs.tunsel(y = dat$y, x = dat$x,
+  disFamily = "norm", nComp = myk,
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = "adplasso", LambMin = 0.01,
+  LambMax = 1, nLamb = 100)
> summary(res.lam)
----------------------
Selected Tuning Parameters:
----------------------
  Finite Mixture of Regression Models
  2 Components; adplasso Penalty;

Component-wise lambda:
 Comp.1 Comp.2
   0.01   0.01
```

Having selected the tuning parameters, we perform variable selection in FMR as follows:

```
> res.var <-
+  fmrs.varsel(y = dat$y, x = dat$x,
+  disFamily = "norm", nComp = myk,
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = "adplasso",
+  lambPen = slot(res.lam, "lambPen"))
> summary(res.var)
----------------------
Fitted Model:
----------------------
  Finite Mixture of Regression Models
  2 Components; 10 Covariates; 500 samples.

Coefficients:
              Comp.1     Comp.2
Intercept  1.9452912 -1.0864495
X0         1.8117090 -0.9395877
X1        -0.8956017  0.8635743
X2        -2.1003460  2.0107557
X3         0.7569757  0.0000000
X4         2.0638455  0.0000000
X5         0.0000000  0.0000000
X6        -0.1142514  0.0000000
X7         0.1386943 -0.9973441
X8         0.0000000  2.0545577
X9         0.0000000 -1.9664867

Selected Set:
   Comp.1 Comp.2
X0      1      1
X1      1      1
X2      1      1
X3      1      0
X4      1      0
X5      0      0
X6      1      0
X7      1      1
X8      0      1
X9      0      1

Dispersion:
    Comp.1    Comp.2
 0.8996044 0.9968098

Mixing Proportions:
    Comp.1    Comp.2
 0.3668194 0.6331806

LogLik: -943.7634; BIC: 1999.39
```

6.3 Example 3: Comparison with the Existing Methods

We conducted a simulation study to compare the performance of **fmrs** with existing methods. Currently, **fmrlasso** is the only method available that offers a variable selection in FMR, and there is no package providing variable selection in FMAFTR.

In our simulation study, we consider the model $y_i = \mathbf{x}_i\boldsymbol{\beta} + e_i$, $i = 1, \ldots, n$, where $e_i \overset{iid}{\sim} N(0, \sigma^2)$. For the FMR model, we set $K = 2$, $n = 500$, $d = 5$, $\pi = [0.4, 0.6]$, $\rho = 0.25$, $\sigma = [1, 1]$, $\boldsymbol{\beta}_1 = [-1, 2, 0, 0, -1, 2]^\top$, $\boldsymbol{\beta}_2 = [2, 1, 1, 0, 0, 0]^\top$.

We generated $r = 100$ datasets and evaluated the performance. The simulation results are presented in Table 4. The results demonstrate that **fmrs** outperforms **fmrlasso** in terms of correctly identifying zero or non-zero coefficients. The codes are given in the supplementary materials.

Table 4. Percentage of correctly identified regression coefficients as zero or non-zero.

Package	β_{11}	β_{21}	β_{31}	β_{41}	β_{51}	β_{12}	β_{22}	β_{32}	β_{42}	β_{52}
fmr	100	100	88	89	89	100	66	77	100	100
fmrlasso	100	100	63	55	54	100	47	61	100	100

It is worth noting that **fmrs** significantly outperforms **fmrlasso** in terms of computational speed. In the aforementioned simulation study, where BIC was used, **fmrs** was found to be over 20 times faster than **fmrlasso**. Moreover, unlike **fmrlasso**, **fmrs** provides variable selection capabilities for various penalties such as MCP, SCAD, and others. Additionally, **fmrs** can handle variable selection in the presence of censored observations, which is a limitation of **fmrlasso**.

6.4 Example 4: Non-Mixture Models

As stated by the reviewers, the non-mixture versions of the models implemented in `fmrs` are highly beneficial for researchers. Therefore, we have added these models to the package. The users need to specify $nComp = 1$ to fit such models. A sample code for the accelerated failure time regression model is given as follows:

```
> library(fmrs); set.seed(1980)
> myk <- 1; myd <- 10; myn <- 500; myrho <- 0.25
> mydisp <- c(1); myu <- 40; mypi <- c(1.0)
> coeff1 <- c(2,2,-1,-2,1,2,0,0,0,0,0)
> dat <-
+  list(y = as.numeric(unlist(read.csv(
+  paste0(PathData,'Ydata_4.csv'),header=TRUE))),
+  x = as.matrix(read.csv(
+  paste0(PathData,'Xdata_4.csv'),header=TRUE)),
+  delta = as.numeric(unlist(read.csv(
+  paste0(PathData,'Ddata_4.csv'),header=TRUE))))
> res.mle <-
+  fmrs.mle(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = rnorm(myk * myd + myk),
+  initDispersion = rep(1, myk),
+  initmixProp = rep(1 / myk, myk))
> summary(res.mle)
----------------------
Fitted Model:
----------------------
  Accelerated Failure Time
              Regression Models
          Log-Normal Distribution
  1 Component; 10 Covariates; 500 samples.

Coefficients:
                Comp.1
Intercept  1.981058507
X0         2.016923696
X1        -1.031801844
X2        -1.967974101
X3         0.947853244
X4         2.066622625
X5         0.010931878
X6        -0.095261647
X7         0.112314056
X8        -0.004505504
X9         0.052504637

Dispersions:
   Comp.1
 1.011018

LogLik: -461.7918; BIC: 998.159

> res.lam <-
+  fmrs.tunsel(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = "adplasso", LambMin = 0.01,
+  LambMax = 1, nLamb = 100)
> summary(res.lam)
----------------------
Selected Tuning Parameters:
----------------------
  Accelerated Failure Time
              Regression Models
          Log-Normal Distribution
  1 Components; adplasso Penalty;

Component-wise lambda:
 Comp.1
 0.0199

> res.var <-
+  fmrs.varsel(y = dat$y, delta = dat$delta,
+  x = dat$x, nComp = myk,
+  disFamily = "lnorm",
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = "adplasso",
```

```
+  lambPen = slot(res.lam,"lambPen"))
> summary(res.var)
----------------------
Fitted Model:
----------------------
  Accelerated Failure Time
              Regression Models
           Log-Normal Distribution
  1 Component; 10 Covariates; 500 samples.

Coefficients:
              Comp.1
Intercept  1.9434740
X0         1.9668506
X1        -0.9992472
X2        -1.9408830
X3         0.9151896
X4         2.0320537
X5         0.0000000
X6         0.0000000
X7         0.0000000
X8         0.0000000
X9         0.0000000

Selected Set:
   Comp.1
X0      1
X1      1
X2      1
X3      1
X4      1
X5      0
X6      0
X7      0
X8      0
X9      0

Dispersions:
   Comp.1
 1.014697

LogLik: -465.0646; BIC: 973.6314
```

7. ANALYZING LUNG CANCER DATA

In this section, we analyze lung cancer data obtained from the **survival** package [22]. The dataset consists of information from 228 subjects with 7 covariates. The data includes the survival time of patients with lung cancer along with their censoring status. The following information is available for each subject:

- `time`: Survival time in days;
- `status`: censoring status 1=censored, 2=dead;
- `age`: Age in years;
- `sex`: Male=1 Female=2;
- `ph.ecog`: Eastern Cooperative Oncology Group (ECOG) performance status (0=good 5=dead);
- `ph.karno`: Karnofsky performance score (bad=0-good=100) rated by physician;
- `pat.karno`: Karnofsky performance score as rated by the patient;
- `meal.cal`: Calories consumed at meals;
- `wt.loss`: Weight loss in the last six months.

In our analysis, we first remove the variable `meal.cal` due to a large number of missing values. Next, we exclude subjects with missing information, resulting in a reduced number of subjects (patients) to 210. The analysis includes the remaining 6 covariates.

We perform variable selection using **fmrs**. We fit FMAFTR models of order K ranging from 2 to 5 using the order selection technique described in Section 4.2 and the component-wise technique described in Section 4.1. We use the following parameter values: `cutpoint = 0.001`, `LambMin = 0.001`, `LambMax = 1`, and `nLamb = 1000`. The code is given as follows:

```
> library(survival)
> lung.r <-lung[,-c(1,9)]
> r.lung <-lung.r[-which(is.na(rowSums(lung.r))),]
> r.lung$sex <- as.numeric(r.lung$sex - 1)
> Days <- r.lung$time; Status <- r.lung$status - 1
> Features <- as.matrix(r.lung[, -c(1, 2)])
> Tol <- 0.001; d <- 6
> Final.Model <- list()
> for (K in 2:5) {
+  res.mle <-
+  fmrs.mle(y = Days, x = Features,
+  delta = Status,
+  nComp = K, disFamily = 'lnorm',
+  initCoeff = rnorm(K * d + K),
+  initDispersion = rep(1, K),
+  initmixProp = rep(1 / K, K))
+  res.lam <-
+  fmrs.tunsel(y = Days, x = Features,
+  delta = Status,
+  nComp = ncomp(res.mle),
+  disFamily = 'lnorm',
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
+  initmixProp = mixProp(res.mle),
+  penFamily = 'adplasso',
+  cutpoint = Tol, LambMin = 0.001,
+  LambMax = 1, nLamb = 1000)
+  res.var <-
+  fmrs.varsel(y = Days, x = Features,
+  delta = Status,
+  nComp = ncomp(res.mle),
+  disFamily = 'lnorm',
+  initCoeff = c(coefficients(res.mle)),
+  initDispersion = dispersion(res.mle),
```

```
+  initmixProp = mixProp(res.mle),
+  penFamily = 'adplasso',
+  lambPen = slot(res.lam, 'lambPen'),
+  cutpoint = Tol)
+  if (K == 2) {
+  Final.Model <-
+  list(res.mle = res.mle, res.lam = res.lam,
+  res.var = res.var)
+  Max_BIC <- BIC(res.var)
+  } else {
+  if (BIC(res.var) <= Max_BIC) {
+  Final.Model <-
+  list(res.mle = res.mle, res.lam = res.lam,
+  res.var =  res.var)
+  Max_BIC <- BIC(res.var)
+  }
+  }
+  }
> summary(Final.Model$res.var)
----------------------
Fitted Model:
----------------------
  Finite Mixture of Accelerated Failure Time
                Regression Models
          Log-Normal Sub-Distributions
  2 Components; 6 Covariates; 210 samples.

Coefficients:
               Comp.1      Comp.2
Intercept 3.725023502 12.69267171
age       0.000000000 -0.06016423
sex       0.401502006  0.00000000
ph.ecog   0.000000000 -0.72846678
ph.karno  0.019597502 -0.03484284
pat.karno 0.004004021  0.00000000
wt.loss   0.000000000  0.00000000

Selected Set:
          Comp.1 Comp.2
age            0      1
sex            1      0
ph.ecog        0      1
ph.karno       1      1
pat.karno      1      0
wt.loss        0      0

Dispersion:
    Comp.1   Comp.2
 0.5232071 1.368616

Mixing Proportions:
    Comp.1    Comp.2
 0.6927942 0.3072058

LogLik: -226.1958; BIC: 511.2097
```

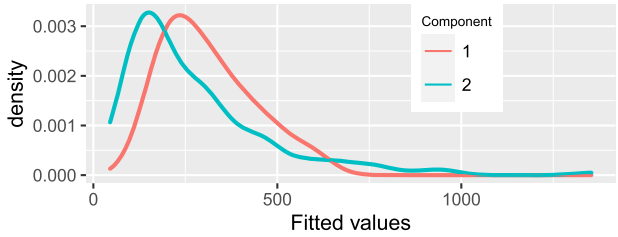

Figure 1: Density of fitted values for the lung cancer data.

The results show that a mixture of two components is selected, with approximately 69% of the patients classified in Component 1. In Component 1, the variables `sex`, `ph.karno`, and `pat.karno` are selected with positive effects. On the other hand, in Component 2, the variables `age`, `ph.ecog`, and `ph.karno` are selected with negative effects. Component 1 represents a more aggressive form of the disease, characterized by a lower survival time (see Figure 1).

8. CONCLUDING REMARKS

We have developed the R package **fmrs** to perform sparse estimation in finite mixture models, including the finite mixture of accelerated failure time and the finite mixture of regression models. The main functions in the package are implemented in C language to enhance computational efficiency. The R functions are written using S4-methods. The package also includes ridge regression, and it implements various penalty functions. Our tests show that the **fmrs** package outperforms **fmrlasso** in terms of computational time.

Censoring is a crucial aspect of time-to-event data, and ignoring it can significantly deteriorate the performance of variable selection methods. Additionally, heterogeneity of effects is common in many time-to-event datasets, and ignoring it can lead to misleading analyses [29].

The classical statistical theory assumes the validity of statistical inference (tests and confidence intervals) when model selection and model fitting are performed separately [3]. If the focus of data analysis is solely on MLE and statistical inference such as goodness-of-fit tests, one can obtain the variance-covariance matrix from the Hessian matrix and perform inference accordingly. However, variable selection methods produce stochastic models, which invalidate classical inferences. Therefore, post-selection methods must be developed to address this issue, which is beyond the scope of this paper.

We recommend users repeat the EM algorithm with different initialization. The best initialization is the one that yields the highest likelihood value. It is worth noting that several initialization strategies have been proposed in the literature [4].

The method presented in [29] is suitable for scenarios with a small number of variables (p) and a large number of

observations (n). In cases where the total number of parameters ($p^* = K(p+3)-1$) approaches or exceeds n, one may utilize the **fmrs** package. However, it is essential to approach such cases with caution and carefully select an optimal ridge tuning parameter. Moreover, for high-dimensional settings, the development of new methods is required.

To expand the scope of our package, we plan to include additional sub-distribution functions, such as the Generalized Gamma and Gompertz distributions, in the near future. For the Gamma and Generalized Gamma distributions, we will investigate closed-form solutions for parameter estimation based on the method of moments and implement them in the package.

SUPPLEMENTARY MATERIAL

Supplementary materials are available online with this paper at the New England Journal of Statistics in Data Science website which includes version 2.0.1 of the **fmrs** package, the R and Python codes as well as simulated datasets ('`CodesAndData.zip`' file) for reproducibility and simulation studies.

APPENDIX A. ALGORITHMS

Algorithm 1 MLE of FMAFTR with Log-Normal sub-distributions.

Data Read data (t_i, δ_i, Z_i) for $i = 1, \ldots, n$; **Set** $t_i' = \log t_i$
Initialize TOL, MaxIter, K, d, m=0, Diff =2
Initialize $\boldsymbol{\Psi}^{(0)} = (\beta_{01}, \ldots, \beta_{0K}, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K, \pi_1, \ldots, \pi_K)$
while m $\leq$ MaxIter & Diff $>$ TOL **do**
  E-Step:
  Update $\boldsymbol{\tau}^{(\mathrm{m})} = [\tau_{ik}^{(\mathrm{m})}]$ similar to Eq. (3.2) using $\boldsymbol{\Psi}^{(\mathrm{m})}$
  Calculate $\boldsymbol{T}^{(\mathrm{m})} = [t_{ik}^{(\mathrm{m})}]$ similar to Eq. (3.4) using $\boldsymbol{\Psi}^{(\mathrm{m})}$
  M-Step:
  for $k = 1$ **to** K **do**
    Construct $\boldsymbol{Z}^\top \boldsymbol{\tau}_k^{(\mathrm{m})} \boldsymbol{Z}$ & $\boldsymbol{Z}^\top \boldsymbol{\tau}_k^{(\mathrm{m})} \boldsymbol{T}_k^{(\mathrm{m})}$
    Update $\beta_{0k}^{(\mathrm{m}+1)}$ and $\boldsymbol{\beta}_k^{(\mathrm{m}+1)}$ based on Eq. (3.5)
    Update $\sigma_k^{(\mathrm{m}+1)}$ similar to Eq. (3.6)
    Update $\pi_k^{(m+1)} = \sum_{i=1}^n \tau_{ik}^{(\mathrm{m})}/n$
  end for
  Calculate Diff $= \|\boldsymbol{\Psi}^{(\mathrm{m}+1)} - \boldsymbol{\Psi}^{(\mathrm{m})}\|$
  Set m = m + 1
end while

Algorithm 2 Component-wise tuning parameter selection in FMAFTR with Log-Normal.

Data Read data (t_i, δ_i, Z_i) for $i = 1, \ldots, n$; **Set** $t_i' = \log t_i$
Initialize K, d, M, ϵ, λ_{Max}, α, `cutpoint`
Choose $\boldsymbol{\lambda}$: candidate set in $(0, \lambda_{\mathrm{Max}}]$
Set $\overline{\boldsymbol{\Psi}}$ obtained from Algorithm 1 (full model)
Calculate $\overline{\boldsymbol{\tau}} = [\overline{\tau}_{ik}]$ similar to Eq. (3.2) using $\overline{\boldsymbol{\Psi}}$
for $k = 1$ **to** K **do**
  for $m = 1$ **to** M **do**
    Calculate $\overline{\boldsymbol{T}} = [\overline{t}_{ik}]$ similar to Eq. (3.4) using $\overline{\boldsymbol{\Psi}}$
    Choose λ_m
    Calculate $\Sigma_k = n\overline{\pi}_k^{\alpha} p'_{\lambda_m}(\overline{\boldsymbol{\beta}}_k)/(|\overline{\boldsymbol{\beta}}_k| + \epsilon)$ and $\boldsymbol{Z}^\top \overline{\boldsymbol{\tau}}_k \boldsymbol{Z} + \Sigma_k$ and $\boldsymbol{Z}^\top \overline{\boldsymbol{\tau}}_k \boldsymbol{T}_k$
    Construct $\boldsymbol{Z}^\top \boldsymbol{\tau}^{(\mathrm{m})} \boldsymbol{Z}$ & $\boldsymbol{Z}^\top \boldsymbol{\tau}^{(\mathrm{m})} \boldsymbol{T}^{(\mathrm{m})}$
    Update $\beta_{0k}^{(\mathrm{new})}$ and $\boldsymbol{\beta}_k^{(\mathrm{new})}$ similar to Eq. (4.2)
    Update $\boldsymbol{T} = [t_{ik}]$ using $\beta_{0k}^{(\mathrm{new})}$ & $\boldsymbol{\beta}_k^{(\mathrm{new})}$
    Update $\sigma_k^{(\mathrm{new})}$ similar to Eq. (3.6) with $\beta_{0k}^{(\mathrm{new})}$ & $\boldsymbol{\beta}_k^{(\mathrm{new})}$
    if $|\beta_{kj}| \leq$ `cutpoint` **then**
      selection$[k][j] = 0$
    else
      selection$[k][j] = 1$
    end if
    Calculate BIC$[m][k]$ base on Eq. (4.4)
  end for
  Choose the λ that minimizes the BIC and report it.
end for

Algorithm 3 Variable selection in FMAFTR with Log-Normal sub-distributions.

Data Read data (t_i, δ_i, Z_i) for $i = 1, \ldots, n$
Set $t_i' = \log t_i$
Initialize K, d, ϵ, α, TOL, MaxIter, m=0, Diff=2, `cutpoint`
Initialize $\boldsymbol{\Psi}^{(0)}$ using Algorithm 1 (full model)
Initialize $\lambda_1, \ldots, \lambda_K$ using Algorithm 2
while m $\leq$ MaxIter & Diff $>$ TOL **do**
  E-Step:
  Update $\boldsymbol{\tau}^{(\mathrm{m})} = [\tau_{ik}^{(\mathrm{m})}]$ similar to Eq. (3.2) using $\boldsymbol{\Psi}^{(\mathrm{m})}$
  Calculate $\boldsymbol{T}^{(\mathrm{m})} = [t_{ik}^{(\mathrm{m})}]$ similar to Eq. (3.4) using $\boldsymbol{\Psi}^{(\mathrm{m})}$
  M-Step:
  for $k = 1$ **to** K **do**
    Calculate $\Sigma_k = n[\pi_k^{(\mathrm{m})}]^{\alpha} p'_{\lambda_m}(\boldsymbol{\beta}_k^{(\mathrm{m})})/(|\boldsymbol{\beta}_k^{(\mathrm{m})}| + \epsilon)$
    Construct $\boldsymbol{Z}^\top \boldsymbol{\tau}_k^{(\mathrm{m})} \boldsymbol{Z} + \Sigma_k$ & $\boldsymbol{Z}^\top \boldsymbol{\tau}_k^{(\mathrm{m})} \boldsymbol{T}_k^{(\mathrm{m})}$
    Update $\beta_{0k}^{(\mathrm{m}+1)}$ and $\boldsymbol{\beta}_k^{(\mathrm{m}+1)}$ based on Eq. (4.2)
    Update $\boldsymbol{T}^{(\mathrm{m})} = [t_{ik}^{(\mathrm{m})}]$ using $\beta_{0k}^{(\mathrm{m})}$ & $\boldsymbol{\beta}_k^{(\mathrm{m})}$
    Update $\sigma_k^{(\mathrm{m}+1)}$ similar to Eq. (3.6) using $\beta_{0k}^{(\mathrm{m})}$ & $\boldsymbol{\beta}_k^{(\mathrm{m})}$
    Update $\pi_k^{(\mathrm{m}+1)} = \sum_{i=1}^n \tau_{ik}^{(m)}/n$
    if $|\beta_{kj}| \leq$ `cutpoint` **then**
      selection$[k][j] = 0$
    else
      selection$[k][j] = 1$
    end if
  end for
  Calculate Diff $= \|\boldsymbol{\Psi}^{(\mathrm{m}+1)} - \boldsymbol{\Psi}^{(\mathrm{m})}\|$
  Set m = m + 1
end while
Report $\widehat{\boldsymbol{\Psi}}$, Log-likelihood, BIC

ACKNOWLEDGEMENTS

The author would like to express his gratitude to the Editor, the Associate Editor, and the two anonymous referees for their valuable and insightful comments, which greatly enhanced the quality of this paper and the **fmrs** package. Additionally, the author would like to extend his thanks to Professor Masoud Asgharian and Abbas Khalili from McGill University, as well as Shili Lin from Ohio State University, for their invaluable contributions to this research. The author would also like to acknowledge Samuel Black and Professor Kazem Taghva from the University of Nevada-Las Vegas for their assistance with C programming.

FUNDING

Farhad Shokoohi is supported by the University of Nevada - Las Vegas, through the Startup Grant PG18929.

Accepted 4 September 2023

REFERENCES

[1] Basturk, N., Hoogerheide, L. F., Opschoor, A. and van Dijk, H. K. (2015). MitISEM: Mixture of Student t Distributions using Importance Sampling and Expectation Maximization. https://cran.r-project.org/web/packages/MitISEM/.

[2] Benaglia, T., Chauveau, D., Hunter, D. R. and Young, D. (2009). mixtools: An R Package for Analyzing Finite Mixture Models. *Journal of Statistical Software* **32**(6) 1–29.

[3] Berk, R., Brown, L., Buja, A., Zhang, K. and Zhao, L. (2013). Valid post-selection inference. *The Annals of Statistics* **41**(2) 802–837. https://doi.org/10.1214/12-AOS1077.

[4] Biernacki, C., Celeux, G. and Govaert, G. (2003). Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models. *Computational Statistics & Data Analysis* **41**(3) 561–575. Recent Developments in Mixture Model. https://doi.org/10.1016/S0167-9473(02)00163-9. MR1968069

[5] Bilgrau, A. E., Eriksen, P. S., Rasmussen, J. G., Johnsen, H. E., Dybkaer, K. and Boegsted, M. (2016). GMCM: Unsupervised Clustering and Meta-Analysis Using Gaussian Mixture Copula Models. *Journal of Statistical Software* **70**(2) 1–23. https://doi.org/10.18637/jss.v070.i02.

[6] Chambers, J. M. (1998) *Programming with Data*. Springer-Verlag, New York.

[7] Chambers, J. M. (2008) *Software for Data Analysis: Programming with R*. Springer, New York.

[8] Chiou, S. H., Kang, S. and Yan, J. (2014). Fitting Accelerated Failure Time Models in Routine Survival Analysis with R Package aftgee. *Journal of Statistical Software* **61**(11) 1–23.

[9] Fraley, C., Raftery, A. E. and Scrucca, L. (2002). Inference for finite mixture models. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* **64**(3) 491–507.

[10] Fraley, C., Raftery, A. E., Murphy, T. B. and Scrucca, L. (2012). mclust Version 4 for R: Normal Mixture Modeling for Model-Based Clustering, Classification, and Density Estimation. https://doi.org/10.1007/s11222-009-9138-7. MR2725403

[11] Garay, A. M., Massuia, M. B. and Lachos, V. H. (2015). BayesCR: Bayesian Analysis of Censored Regression Models Under Scale Mixture of Skew Normal Distributions. https://cran.r-project.org/web/packages/BayesCR/.

[12] Garay, A. M., Massuia, M. B. and Lachos, V. (2015). SMNCensReg: Fitting Univariate Censored Regression Model Under the Family of Scale Mixture of Normal Distributions. https://cran.r-project.org/web/packages/SMNCensReg/.

[13] Grün, B. and Leisch, F. (2008). FlexMix Version 2: Finite Mixtures with Concomitant Variables and Varying and Constant Parameters. *Journal of Statistical Software* **28**(4) 1–35.

[14] Hennig, C. (2004). Identifiability of mixtures of regression models. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* **66**(3) 593–615.

[15] Iannario, M. and Piccolo, D. (2015). CUB: A Class of Mixture Models for Ordinal Data. https://cran.r-project.org/web/packages/CUB/.

[16] Iovleff, S. (2015). MixAll: Clustering Heterogenous data with Missing Values. https://cran.r-project.org/web/packages/MixAll/.

[17] Jin, Z. (2016). Semiparametric accelerated failure time model for the analysis of right censored data. *Communications for Statistical Applications and Methods* **23**(6) 467–478.

[18] Kamary, K. and Lee, K. (2015). Ultimixt: Bayesian Analysis of a Non-Informative Parametrization for Gaussian Mixture Distributions. https://cran.r-project.org/web/packages/Ultimixt/index.html.

[19] Lawless, J. F. (2003) *Statistical Models and Methods for Lifetime Data*, 2nd ed. Wiley Series in Probability and Statistics. MR1940115

[20] Lebret, R., Iovleff, S., Langrognet, F., Biernacki, C., Celeux, G. and Govaert, G. (2015). Rmixmod: The R Package of the Model-Based Unsupervised, Supervised, and Semi-Supervised Classification Mixmod Library. *Journal of Statistical Software* **67**(6) 1–29. https://doi.org/10.18637/jss.v067.i06.

[21] Lee, S. X. and McLachlan, G. J. (2013). EMMIXuskew: An R Package for Fitting Mixtures of Multivariate Skew t Distributions via the EM Algorithm. *Journal of Statistical Software* **55**(12) 1–22. https://doi.org/10.1007/s11222-012-9362-4. MR3165547

[22] Loprinzi, C. L., Laurie, J. A., Wieand, H. S. et al. (1994). Prospective evaluation of prognostic variables from patient-completed questionnaires. North Central Cancer Treatment Group. *Journal of Clinical Oncology* **12**(3) 601–607. PMID: 8120560. https://doi.org/10.1200/JCO.1994.12.3.601.

[23] Mahani, A. S. and Sharabiani, M. T. A. (2015). BayesMixSurv: Bayesian Mixture Survival Models using Additive Mixture-of-Weibull Hazards, with Lasso Shrinkage and Stratification. https://cran.r-project.org/web/packages/BayesMixSurv/.

[24] McLachlan, G. and Peel, D. (2004) *Finite Mixture Models*. John Wiley & Sons. https://doi.org/10.1002/0471721182. MR1789474

[25] Panaro, R. V. (2020). *spsurv: An R package for semi-parametric survival analysis*. 2003.10548.

[26] Prates, M. O., Cabral, C. R. B. and Lachos, V. H. (2013). mixsmsn: Fitting Finite Mixture of Scale Mixture of Skew-Normal Distributions. *Journal of Statistical Software* **54**(12) 1–20.

[27] Sanchez, L. B. and Lachos, V. H. (2015). CensMixReg: Censored Linear Mixture Regression Models. https://cran.r-project.org/web/packages/CensMixReg/.

[28] Schlattmann, P., Hoehne, J. and Verba, M. (2015). CAMAN: Finite Mixture Models and Meta-Analysis Tools - Based on C.A.MAN. https://cran.r-project.org/web/packages/CAMAN/.

[29] Shokoohi, F., Khalili, A., Asgharian, M. and Lin, S. (2019). Capturing heterogeneity of covariate effects in hidden subpopulations in the presence of censoring and large number of covariates. *The Annals of Applied Statistics* **13**(1) 444–465. https://doi.org/10.1214/18-AOAS1198. MR3937436

[30] Silva, R. R. (2016). BayesH: Bayesian Regression Model with Mixture of Two Scaled Inverse Chi Square as Hyperprior. https://cran.r-project.org/web/packages/BayesH/.

[31] Städler, N. (2010). fmrlasso: Lasso for Finite Mixture of Regressions. http://mukherjeelab.nki.nl/stadler/fmrlasso_1.0.tar.gz.

[32] Su, S., Wuertz, D., Maechler, M., Rmetrics et al. (2015). GLDEX: Fitting Single and Mixture of Generalised Lambda Distributions (RS and FMKL) using Various Methods. https://cran.r-project.org/web/packages/GLDEX/.

[33] Tortora, C., Browne, R. P., Franczak, B. C. and McNi-

CHOLAS, P. D. (2015). MixGHD: Model Based Clustering, Classification and Discriminant Analysis Using the Mixture of Generalized Hyperbolic Distributions. https://cran.r-project.org/web/packages/MixGHD/.

Farhad Shokoohi. Department of Mathematical Sciences, University of Nevada-Las Vegas, Las Vegas, USA. E-mail address: farhad.shokoohi@unlv.edu