Canadian Institute of Actuaries | Institut canadien des actuaires

Document de recherche

LRMoE : Un progiciel R pour la modélisation flexible des sinistres d'assurance au moyen d'un modèle de régression de mélange d'experts

Spark C. Tseung, Andrei L. Badescu,
Tsz Chai Fung et X. Sheldon Lin

Février 2022

Document rp222020

This document is available in English

Points saillants des documents

LRMoE.jl : un progiciel qui modélise les sinistres d'assurance au moyen d'un mélange de modèles de régression d'experts

et

LRMoE : un progiciel R pour la modélisation flexible des sinistres d'assurance au moyen d'un modèle de régression de mélange d'experts

Depuis quelques années, nous, chercheurs à l'Université de Toronto (Andrei L. Badescu, X. Sheldon Lin et des doctorants actuels ou anciens) travaillons à des projets de recherche sur le calcul des réserves et la tarification en assurances IARD. Notre objectif est de concevoir des technologies nouvelles et réalisables et des progiciels prêts à utiliser à l'intention des actuaires exerçant dans ce domaine. Ce projet est l'un des résultats de ces efforts. Son financement a été rendu possible grâce à une subvention pour la recherche universitaire de l'Institut canadien des actuaires (ICA), que nous tenons à remercier.

Dans le cadre de ce projet, nous introduisons un nouveau logiciel statistique libre, conçu sur mesure pour les applications actuarielles, qui permet aux praticiens en actuariat de modéliser et d'analyser la fréquence et la sévérité des sinistres d'assurance au moyen d'un modèle de régression multivarié et non linéaire. Le modèle est très flexible : il peut s'adapter à n'importe quel type de base de données positives et prendre en compte la structure de dépendance que laissent entendre les données, en plus d'être réalisable statistiquement.

Nous avons produit deux documents avec progiciels correspondants : l'un donne une courte description d'un progiciel R, et l'autre, d'un progiciel Julia. R est connu de nombreux actuaires, tandis que Julia est un langage de programmation très efficace qui est communément utilisé par les communautés de l'apprentissage automatique et de l'informatique. Le progiciel Julia s'exécute environ quatre fois plus rapidement que le progiciel R. Ces progiciels peuvent être téléchargés respectivement aux adresses https://github.com/sparktseung/LRMoE.jl et https://github.com/sparktseung/LRMoE. Ils offrent des caractéristiques distinctives qu'on ne peut exploiter avec les progiciels existants, les principales étant une couverture plus large des distributions de fréquence et de sévérité et leur version gonflée à zéro, l'estimation de paramètres en présence de censure et/ou de troncature dans les données, et un ensemble de fonctions permettant le calcul des réserves et la tarification en assurance. De plus, les logiciels comportent plusieurs fonctions d'évaluation et de visualisation de modèles qui facilitent la tâche des utilisateurs lorsqu'ils analysent la performance du modèle ajusté et interprètent le modèle dans des contextes d'assurance.

On trouvera de plus amples informations sur le développement des modèles et des méthodes qui sous-tendent nos progiciels dans les ouvrages suivants :

- Fung, T.C., Badescu, A. et X.S. Lin. « A class of mixture of experts models for general insurance: Application to correlated claim frequencies », *ASTIN Bulletin*, **vol. 49**, n° 3, 2019, pp. 647–688.

- Fung, T.C., Badescu, A. et X.S. Lin. « A class of mixture of experts models for general insurance: Theoretical developments », *Insurance: Mathematics and Economics*, **vol. 89**, 2019, p. 111–127.
- Fung, T.C., Badescu, A. et X.S. Lin. « A new class of severity regression models with an application to IBNR prediction », *North American Actuarial Journal*, **vol. 25,** 2020, n^{o} 2, p. 1–26.
- Fung, T.C., Badescu, A. et X.S. Lin. « Fitting censored and truncated regression data using the mixture of experts models », 2021. Sur SSRN : https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3740061.

Ces ouvrages sont disponibles sur demande. Pour toute question ou commentaire, communiquez avec nous.

Andrei L. Badescu et X. Sheldon Lin

Université de Toronto

badescu@utstat.utoronto.ca, sheldon.lin@utoronto.ca

LRMoE : un progiciel R pour la modélisation flexible des sinistres d'assurance au moyen d'un modèle de régression de mélange d'experts

Résumé

Dans le présent document, nous présentons un nouveau progiciel R, **LRMoE**, qui est un logiciel statistique conçu sur mesure pour les applications actuarielles et qui permet aux chercheurs et aux praticiens en actuariat de modéliser et d'analyser la fréquence et la sévérité des sinistres d'assurance au moyen du modèle *Logit-weighted Reduced Mixture of Experts* (LRMoE). **LRMoE** offre plusieurs nouvelles fonctionnalités distinctives utiles pour diverses applications actuarielles et qu'on ne peut généralement pas exploiter au moyen des progiciels existants avec les modèles par mélange. Parmi les principales fonctionnalités, mentionnons une couverture plus large des distributions de fréquence et de sévérité et de leur version gonflée à zéro, la flexibilité de pouvoir varier les classes de distributions pour les différentes composantes, l'estimation de paramètres en présence de censure et/ou de troncature dans les données, et un ensemble de fonctions de calcul des réserves et de tarification en assurance. De plus, le logiciel comporte plusieurs fonctions d'évaluation et de visualisation de modèles qui facilitent la tâche des utilisateurs lorsqu'ils analysent la performance du modèle ajusté et interprètent le modèle dans des contextes d'assurance.

Mots-clés : Analyse de régression multivariée, censure et troncature, algorithme espérance conditionnelle-maximisation, calcul des réserves et tarification en assurance, R.

1. Introduction

Le LRMoE est un modèle de régression flexible créé par Fung et coll. (2019b) et il est considéré comme la version régression d'un modèle de mélange fini avec des poids de mélange (appelés *fonction de contrôle*) qui dépendent des covariables. Nous pouvons interpréter le LRMoE comme un outil de classification des titulaires de police en divers sous-groupes dont les probabilités diffèrent. Dépendantes de la composante sous-groupe à laquelle chaque titulaire de police est affecté, les propriétés distributionnelles de la fréquence ou de la sévérité des sinistres sont régies par les fonctions de la composante mélange (appelées *fonctions d'experts*). La flexibilité, la parcimonie et la maniabilité mathématique du modèle sont justifiées (voir Fung et coll. 2019b), ce qui démontre le solide fondement théorique du LRMoE dans le contexte général de la modélisation des sinistres d'assurance. Pour certains choix de fonctions d'experts, Fung et coll. (2019a) et Fung et coll. (2020) ont construit des algorithmes espérance-maximisation conditionnelle (EMC) permettant un ajustement efficace des modèles de fréquence et de sévérité et ils ont ainsi démontré l'utilité éventuelle du LRMoE pour le calcul des réserves et la tarification en assurance.

Bien que le progiciel R existant **flexmix existant en R** (Leisch 2004, et Grün et Leisch 2008) puisse effectuer une estimation des paramètres pour certains cas particuliers du LRMoE, il n'offre que des choix limités de fonctions d'experts (Poisson, gaussienne, gamma et binomiale) pour l'ajustement du modèle. Miljkovic et Grün (2016) ont profité de la souplesse qu'offre le progiciel flexmix afin de proposer de nouveaux modèles de mélange intégrant d'autres fonctions d'experts (comme log-normale, Weibull et Burr), mais les utilisateurs sont toujours contraints de choisir une seule distribution paramétrique pour toutes les composantes.

Dans le présent document, nous présentons un nouveau progiciel Julia, **LRMoE**, qui est un logiciel statistique conçu sur mesure pour les applications actuarielles et qui permet aux chercheurs et aux praticiens en actuariat de modéliser et d'analyser la fréquence et la sévérité des sinistres d'assurance au moyen du modèle LRMoE. Le progiciel offre plusieurs nouvelles fonctionnalités distinctives utiles pour diverses applications actuarielles et qu'on ne peut généralement pas répliquer au moyen des progiciels existants, dont les suivantes :

- Prise en charge plus large des distributions de fréquence et de sévérité : Outre les distributions de sévérité couvertes par Miljkovic et Grün (2016), le progiciel prend également en charge d'autres fonctions d'experts pour la fréquence qui sont importantes pour la modélisation actuarielle des sinistres, notamment la distribution binomiale négative et la distribution de comptage gamma.

- Distributions gonflées à zéro : Souvent, les actuaires s'intéressent davantage à l'analyse des sinistres agrégés de chaque titulaire de police qu'aux analyses séparées de la fréquence et de la sévérité. Dans cette situation, il est courant d'observer trop de zéros, ce qui motive l'utilisation de fonctions d'experts gonflées à zéro dans le LRMoE, et c'est ce qu'offre le progiciel. À noter que pour que le calcul du LRMoE gonflé à zéro soit efficace, il faut définir une variable latente additionnelle (voir la section 2.2 pour plus de détails), ce qui diminue la souplesse qu'offre le progiciel **flexmix**.

- Possibilités de varier classes de distributions pour les différentes composantes du mélange : Les données d'assurance peuvent présenter des comportements incohérents pour les petites et les grandes valeurs (queue de la distribution). Cela doit être pris en compte par l'utilisation de distributions différentes. L'une des approches consiste à choisir deux distributions et à les combiner à l'aide d'une méthode de dépassement de seuil (voir, par exemple, Lee et coll. 2012 et Scollnik et Sun 2012). Une autre approche consiste à recourir à un modèle de mélange fini fondé sur des distributions de composantes différentes (voir, par exemple, Blostein et Miljkovic 2019). Le progiciel **LRMoE** s'apparente à la dernière approche, les utilisateurs pouvant sélectionner différentes fonctions d'experts pour les différentes composantes du mélange, ce qui permet une modélisation plus flexible et plus réaliste des données.

- Données incomplètes : Dans de nombreuses applications actuarielles, notamment en réassurance, en gestion du risque opérationnel, en tarification avec franchise et en modélisation des réserves, on observe souvent des données censurées et tronquées qu'il faut pouvoir traiter. La censure et la troncature du LRMoE sont introduites par Fung et coll. (2021) dans le cas où les fonctions d'experts sont des distributions gamma

univariées. Le nouveau progiciel élimine cette restriction en permettant aux utilisateurs de prendre en compte des données multivariées qui ont été censurées et tronquées de façon aléatoire à l'aide d'un grand choix de fonctions d'experts.

- Sélection et visualisation du modèle : En plus de la fonction d'ajustement du modèle, le nouveau progiciel offre plusieurs fonctions d'évaluation du modèle (critère d'information d'Akaike [CIA], critère d'information bayésien [CIB]) et de visualisation du modèle (p. ex., probabilités des classes latentes, influence des covariables) qui facilitent la tâche des utilisateurs lorsqu'ils analysent la performance du modèle ajusté et interprètent le modèle dans des contextes d'assurance.
- Calcul des réserves et tarification en assurance : Le progiciel comprend également plusieurs de fonctions pour le calcul des réserves et la tarification (p. ex., la moyenne, la variance, la valeur au risque [VaR], l'espérance conditionnelle unilatérale [ECU]), qui permettent aux actuaires de procéder simultanément à la tarification d'un grand nombre de contrats d'assurance ayant des caractéristiques différentes à l'aide de plusieurs principes de primes.

Le présent document est organisé comme suit. À la section 2, nous examinerons le modèle LRMoE et l'estimation des paramètres au moyen de l'algorithme EMC. À la section 3, nous utiliserons un ensemble de données simulées pour illustrer la procédure de base d'ajustement du progiciel **LRMoE**. À la section 4, nous traiterons d'autres fonctionnalités du progiciel, comme l'initialisation des paramètres, la visualisation du modèle et la fonction de calcul des primes, qui sont illustrées à l'aide d'un ensemble de données d'assurance automobile recueilli en France. Enfin, à la section 5, nous discuterons de certains points. Par souci de concision, nous ne présenterons que les lignes de code les plus pertinentes pour notre nouveau progiciel. On trouvera le code source, la documentation du progiciel et le code de réplication complet pour tous les exemples du présent document aux adresses https://github.com/sparktseung/LRMoE et https://github.com/ sparktseung/LRMoE-Paper-Demo.

2. Modèle LRMoE et estimation des paramètres

Nous présentons ici un aperçu du modèle LRMoE proposé dans Fung et coll. (2019b) et nous examinons l'algorithme EMC pour l'estimation des paramètres. Par souci de concision, nous supposerons aux sections 2.1 et 2.2 que toutes les variables de réponse (fréquence ou sévérité des sinistres) sont observées exactement. À la section 2.3, nous traiterons de la troncature et de la censure des données pour le modèle LRMoE.

2.1 Modèle logit réduit pondéré de mélange d'experts (LRMoE)

Soit $\boldsymbol{x}_i = (x_{i0}, x_{i1}, \ldots, x_{iP})^T$ le vecteur de covariables de longueur (P +1) pour le titulaire de police i (i = 1, 2, ..., n) avec terme constant $x_{i0} = 1$, et $\boldsymbol{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iD})^T$ le vecteur de longueur D dimensions des variables de réponse, qui peuvent être la fréquence ou la sévérité des sinistres. Soit $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)^T$ et $\boldsymbol{Y} = (y_1, y_2, \ldots, y_n)^T$ toutes les covariables et réponses pour un groupe de n titulaires.

D'après les covariables, le titulaire *i* est classé dans l'une des *g* classes de risque latentes par une fonction de contrôle logit

$$\pi_j(\boldsymbol{x}_i;\boldsymbol{\alpha}_j) = \frac{exp(\boldsymbol{\alpha}_j^T \boldsymbol{x}_i)}{\sum_{j'=1}^{g} exp(\boldsymbol{\alpha}_{j'}^T \boldsymbol{x}_i)}, \quad j = 1, 2, \ldots, g, \tag{1}$$

où $\boldsymbol{\alpha}_j = (\alpha_{j0}, \alpha_{j1}, \ldots, \alpha_{jP})^T$ est un vecteur de coefficients de régression pour la classe latente j.

Étant donné l'attribution de la classe latente j, les variables de réponse $\boldsymbol{y}_i = (y_{i1}, y_{i2}, \ldots, y_{iD})^T$ sont conditionnellement indépendantes, et la fonction de densité de probabilité de d^e dimension (ou fonction de masse de probabilité) est donnée par

$$g_{jd}(y_{id};\delta_{jd},\psi_{jd}) = \delta_{jd}\mathbf{1}\{y_{id}=0\} + (1-\delta_{jd}) f_{jd}(y_{id};\psi_{jd}) \tag{2}$$

où δ_{jd} est une masse de probabilité en zéro, $1\{y_{id}=0\}$ est la fonction indicatrice et $f_{jd}(y_{id};\psi_{jd})$ est la densité d'une distribution couramment utilisée avec paramètres ψ_{jd} pour modéliser les sinistres d'assurance. Le tableau 1 donne la liste des distributions paramétriques prises en charge par le progiciel LRMoE.

Tableau 1 : Distributions prises en charge par le LRMoE

Racine	Distribution	$f_{jd}(y)$	Paramètres
gamma	Gamma	$\frac{1}{\theta^m \Gamma(m)} y^{m-1} e^{-y/\theta}$	$m > 0,\ \theta > 0$
lnorm	Log-normale	$\frac{1}{y\sigma\sqrt{2\pi}} \exp\left[-\frac{(\log y - \mu)^2}{2\sigma^2}\right]$	$\mu \in \mathbb{R},\ \sigma > 0$
invgauss	Gaussienne inverse	$\sqrt{\frac{\lambda}{2\pi y^3}} \exp\left[-\frac{\lambda(y-\mu)^2}{2\mu^2 y}\right]$	$\mu > 0,\ \lambda > 0$
weibull	Weibull	$\frac{k}{\lambda}\left(\frac{y}{\lambda}\right)^{k-1} \exp\left[-\left(\frac{y}{\lambda}\right)^k\right]$	$k > 0,\ \lambda > 0$
burr	Burr	$\frac{ck}{\lambda}\left(\frac{y}{\lambda}\right)^{c-1}\left[1+\left(\frac{y}{\lambda}\right)^c\right]^{-k-1}$	$k > 0,\ c > 0,\ \lambda > 0$
poisson	Poisson	$e^{-\lambda}\frac{\lambda^y}{y!}$	$\lambda > 0$
nbinom	Binomiale négative	$\binom{y+n-1}{n-1} p^n (1-p)^y$	$n \in \mathbb{N}^+,\ 0 < p < 1$
gammacount	Comptage gamma	$\int_0^{ms} \frac{1}{\Gamma(ys)} u^{ys-1} \exp\{-u\} du$ $- \int_0^{ms} \frac{1}{\Gamma((y+1)s)} u^{(y+1)s-1} \exp\{-u\} du$	$m > 0,\ s > 0$
ZI-root	Toutes les distributions précédentes	$g_{jd} = \delta_{jd} \mathbf{1}\{y = 0\} + (1 - \delta_{jd}) f_{jd} \quad 0 < \delta_{jd} < 1$	

À noter qu'une masse de probabilité δ_{jd} en zéro permet une modélisation plus réaliste des données d'assurance gonflées à zéro. Comme convention de nommage, nous désignerons par g_{jd} la *fonction de distribution de composantes/fonction d'experts*, et par f_{jd} sa *partie positive*, bien que les distributions de fréquence des sinistres (p. ex., Poisson) aient également une masse de probabilité en zéro.

Sous LRMoE, la fonction de densité de probabilité conditionnelle (ou fonction de masse de probabilité) de y_i étant donné les covariables $\boldsymbol{x}_i$ est donnée par

$$h(\boldsymbol{y}_i; \boldsymbol{x}_i, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\Psi}) = \sum_{j=1}^{g} \pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j) \times \prod_{d=1}^{D} g_{jd}(y_{id}; \delta_{jd}, \psi_{jd}) \qquad (3)$$

où $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_g)^T$ est une matrice $g \times (P+1)$ de poids de mélange avec $\boldsymbol{\alpha}_g = (0, 0, \ldots, 0)^T$ représentant la classe par défaut, $\boldsymbol{\delta} = (\delta_{jd})_{1 \le j \le g, 1 \le d \le D}$ est une matrice $(g \times D)$ de masses de probabilité en zéro par composante et par dimension, et $\boldsymbol{\Psi} = \{\psi_{jd} : 1 \le j \le g, 1 \le d \le D\}$ est une liste de paramètres pour la partie positive f_{jd} par composante et par dimension. À noter que $\alpha_g = (0, 0, \ldots, 0)^T{}_T$ rend le modèle identifiable, c'est-à-dire qu'il existe une application injective entre les distributions de régression et les paramètres (voir Jiang et Tanner 1999, et Fung et coll. 2019a).

Pour un groupe de n titulaires de police, la fonction de vraisemblance est donnée par

$$L(\boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\Psi}; \boldsymbol{X}, \boldsymbol{Y}) = \prod_{i=1}^{n} h(\boldsymbol{y}_i; \boldsymbol{x}_i, \boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\Psi}) = \prod_{i=1}^{n} \left\{ \sum_{j=1}^{g} \pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j) \times \prod_{d=1}^{D} g_{jd}(y_{id}; \delta_{jd}, \psi_{jd}) \right\}. \quad (4)$$

2.2 Estimation des paramètres

Pour l'estimation des paramètres des modèles à mélange fini, on utilise couramment l'algorithme espérance-maximisation (EM) (voir, par exemple, Dempster et coll. 1977 et McLachlan et Peel 2004). Toutefois, pour le modèle LRMoE, l'étape M nécessite la maximisation d'une fonction non concave sur tous les éléments de $\boldsymbol{\alpha}$. Fung et coll. (2019a) utilise donc l'algorithme EMC (Meng et Rubin 1993) qui divise l'étape M en plusieurs sous-étapes. Nous décrivons ci-dessous l'algorithme EMC intégré au progiciel **LRMoE**.

Nous désignons par $\boldsymbol{\Phi} = (\boldsymbol{\alpha}, \boldsymbol{\delta}, \boldsymbol{\Psi})$ les paramètres à estimer. Pour $i = 1, 2, \ldots, n$, nous introduisons un vecteur aléatoire latent $\boldsymbol{Z}_i = (Z_{i1}, Z_{i2}, \ldots, Z_{ig})^T$, où $Z_{ij} = 1$ si $\boldsymbol{y}_i$ provient de la j^{e} distribution des composantes et $Z_{ij} = 0$ sinon. Pour $d = 1, 2, \ldots, D$, nous posons $Z_{ij} = Z_{ijd0} + Z_{ijd1}$, où $Z_{ijd0} = 0$ et $Z_{ijd1} = 1$ si la d^{e} dimension de y_i provient de la partie positive f_{jd} de la j^{e} composante, et $Z_{ijd0} = 1$ et $Z_{ijd1} = 0$ si elle provient de la surreprésentation de zéros δ_{jd}.

La fonction de log-vraisemblance des données complètes est donnée par

$$
\begin{aligned}
l^{\text{com}}(\boldsymbol{\Phi}; \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}) &= \sum_{i=1}^{n}\sum_{j=1}^{g} Z_{ij}\left\{\log \pi_j(\boldsymbol{x}_i;\boldsymbol{\alpha}_j) + \sum_{d=1}^{D}\log g_{jd}(y_{id};\delta_{jd},\psi_{jd})\right\} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{g} Z_{ij}\log \pi_j(\boldsymbol{x}_i;\boldsymbol{\alpha}_j) + \sum_{i=1}^{n}\sum_{j=1}^{g}\sum_{d=1}^{D}\{Z_{ijd0}\log\delta_{jd} + Z_{ijd1}\log(1-\delta_{jd})\} \\
&+ \sum_{i=1}^{n}\sum_{j=1}^{g}\sum_{d=1}^{D} Z_{ijd1}\log f_{jd}(y_{id};\psi_{jd}).
\end{aligned}
$$

Étape E

Pour chaque *i* = 1, 2, ... , *n*, le vecteur aléatoire $\boldsymbol{Z}_i$ suit une distribution multinomiale avec paramètre de comptage égal à 1 et probabilités $(\pi_1(\boldsymbol{x}_i;\boldsymbol{\alpha}_1), \pi_2(\boldsymbol{x}_i;\boldsymbol{\alpha}_2), \ldots, \pi_g(\boldsymbol{x}_i;\boldsymbol{\alpha}_g))$. Sachant que $Z_{ij} = 1$, la distribution conditionnelle de Z_{ijd0} est une Bernoulli avec probabilité δ_{jd}. Par conséquent, à la t^{e} itération, les espérances a posteriori de Z_{ij}, Z_{ijd0} et Z_{ijd1} sont

$$
\begin{aligned}
z_{ij}^{(t)} &= \mathsf{E}\{Z_{ij}|\boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} = \mathsf{P}\{Z_{ij}=1|\boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} \\
&= \frac{\pi_j(\boldsymbol{x}_i;\boldsymbol{\alpha}_j^{(t-1)}) \times \prod_{d=1}^{D} g_{jd}(y_{id};\delta_{jd}^{(t-1)},\psi_{jd}^{(t-1)})}{\sum_{j'=1}^{g}\pi_{j'}(\boldsymbol{x}_i;\boldsymbol{\alpha}_{j'}^{(t-1)}) \times \prod_{d=1}^{D} g_{j'd}(y_{id};\delta_{j'd}^{(t-1)},\psi_{j'd}^{(t-1)})},
\end{aligned}
\tag{6}
$$

$$
\begin{aligned}
z_{ijd0}^{(t)} &= \mathsf{E}\{Z_{ijd0}|\boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} \\
&= \mathsf{P}\{Z_{ijd0}=1|\boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}, Z_{ij}=1\} \times \mathsf{P}\{Z_{ij}=1|\boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}\} \\
&= \frac{\delta_{jd}^{(t-1)}\mathbf{1}\{y_{id}=0\}}{\delta_{jd}^{(t-1)}\mathbf{1}\{y_{id}=0\} + (1-\delta_{jd}^{(t-1)})F_{jd}(0;\psi_{jd}^{(t-1)})} \times z_{ij}^{(t)},
\end{aligned}
\tag{7}
$$

et

$$
z_{ijd1}^{(t)} = z_{ij}^{(t)} - z_{ijd0}^{(t)} \tag{8}
$$

où F_{jd} est la fonction de répartition de la partie positive f_{jd}.

Étape MC

À l'étape MC, nous cherchons à maximiser $Q(\boldsymbol{\Phi}; \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y})$, qui peut être décomposée en trois parties comme suit

$$Q(\boldsymbol{\Phi}; \boldsymbol{\Phi}^{(t-1)}, \boldsymbol{X}, \boldsymbol{Y}) = Q_{\boldsymbol{\alpha}}^{(t)} + Q_{\boldsymbol{\delta}}^{(t)} + Q_{\boldsymbol{\Psi}}^{(t)} \tag{9}$$

où

$$Q_{\boldsymbol{\alpha}}^{(t)} = \sum_{i=1}^{n} \sum_{j=1}^{g} z_{ij}^{(t)} \log \pi_j(\boldsymbol{x}_i; \boldsymbol{\alpha}_j), \tag{10}$$

$$Q_{\boldsymbol{\delta}}^{(t)} = \sum_{i=1}^{n} \sum_{j=1}^{g} \sum_{d=1}^{D} \left\{ z_{ijd0}^{(t)} \log \delta_{jd} + z_{ijd1}^{(t)} \log(1 - \delta_{jd}) \right\}, \tag{11}$$

et

$$Q_{\boldsymbol{\Psi}}^{(t)} = \sum_{i=1}^{n} \sum_{j=1}^{g} \sum_{d=1}^{D} z_{ijd1}^{(t)} \log f_{jd}(y_{id}; \boldsymbol{\psi}_{jd}). \tag{12}$$

Pour maximiser $Q_{\boldsymbol{\alpha}}^{(t)}$, nous utilisons la même maximisation conditionnelle que celle décrite dans Fung et coll. (2019a). Nous la maximisons d'abord par rapport à $\boldsymbol{\alpha}_1$ avec $\boldsymbol{\alpha}_j$ fixé à $\boldsymbol{\alpha}_j^{(t-1)}$ pour $j = 2, 3, \ldots, g - 1$. La prochaine étape consiste à maximiser par rapport à $\boldsymbol{\alpha}_2$ avec $\boldsymbol{\alpha}^{(t)}$ à jour et un autre $\boldsymbol{\alpha}_j$ fixé à $\boldsymbol{\alpha}_j^{(t-1)}$ pour $j = 3, 4, \ldots, g - 1$. Le processus se poursuit jusqu'à ce que tous les $\boldsymbol{\alpha}$ aient été mis à jour. Pour obtenir chaque $\boldsymbol{\alpha}_j^{(t)}$, on utilise la méthode des moindres carrés repondérés itérativement (Jordan et Jacobs 1994) jusqu'à la convergence.

Pour $Q_{\boldsymbol{\delta}}^{(t)}$, chaque δ_{jd} peut-être mis à jour en utilisant la solution en forme close suivante :

$$\delta_{jd}^{(t)} = \frac{\sum_{i=1}^{n} z_{ijd0}^{(t)}}{\sum_{i=1}^{n} (z_{ijd0}^{(t)} + z_{ijd1}^{(t)})}. \tag{13}$$

La maximisation de $Q_{\boldsymbol{\Psi}}^{(t)}$ peut aussi être divisée en plus petits problèmes par composante et par dimension. Pour la mise à jour de chaque $\psi_{jd}^{(t)}$, les solutions en forme close ne sont disponibles que pour des distributions très spéciales (p. ex., Poisson, log-normale). L'optimisation

numérique est utilisée dans la plupart des cas, surtout lorsque les observations y_i ne sont pas observées exactement (voir la section 2.3).

Comme il en a été question dans McLachlan et Peel (2004), un mélange de distributions de la sévérité peut avoir une vraisemblance non bornée, ce qui donne lieu à de faux modèles avec des valeurs estimées pour les paramètres extrêmement grandes ou petites. Dans le progiciel **LRMoE**, nous adoptons la même approche maximale a posteriori que dans Fung et coll. (2020), qui utilise des distributions a priori appropriées pour pénaliser les valeurs estimées des paramètres ajustés (voir la section 3). Si nous incluons des fonctions de pénalité, c'est pour éviter d'obtenir un faux modèle du fait que la fonction de log-vraisemblance est non bornée. La pénalité elle-même devrait être assez faible pour qu'elle ait un impact négligeable sur le modèle ajusté. Par contre, les fonctions de pénalité évitent les paramètres qui divergent vers des valeurs déraisonnables et augmentent la robustesse du modèle ainsi obtenu. Nous recommandons aux lecteurs qui souhaitent obtenir des informations détaillées sur les justifications et les exécutions de consulter la section 4 de Fung et coll. (2020).

2.3 LRMoE avec censure et troncature

La censure et la troncature sont courantes dans les ensembles de données d'assurance et elles doivent être traitées. Par exemple, lorsqu'une limite d'assurance s'applique, les montants des sinistres supérieurs à la limite prennent uniquement pour valeur cette limite, créant ainsi une censure à droite des données complètes sur les sinistres; lorsqu'une franchise de police s'applique, les montants des sinistres inférieurs à la franchise ne sont pas déclarés à l'assureur, ce qui entraîne une troncature à gauche.

Fung et coll. (2021) ont traité du modèle LRMoE avec censure et troncature, où toutes les distributions de composantes sont des Gamma. Pour l'estimation des paramètres avec censure et troncature des données, l'algorithme EMC de la section 2.2 est légèrement modifié, avec une étape E supplémentaire pour éliminer les incertitudes découlant de la censure et de la troncature. Puisque l'objectif premier du présent document est de démontrer l'application du progiciel **LRMoE**, nous omettrons les détails et invitons les lecteurs intéressés à consulter l'ouvrage cité.

Pour toutes les distributions incluses dans le progiciel **LRMoE**, celui-ci peut effectuer l'estimation des paramètres en cas de troncature et de censure des données. Par conséquent, les données que l'utilisateur doit saisir pour les modèles de mélange diffèrent légèrement de celles qu'il doit saisir dans les progiciels existants. À la section 3, nous donnons un exemple détaillé de l'ajustement de modèle dans notre progiciel.

2.4 Tarification et calcul de réserves dans le LRMoE

La structure du modèle LRMoE permet de calculer facilement des quantités pertinentes pour la tarification et le calcul des réserves actuarielles (voir Fung et coll. 2019b). Au niveau des titulaires de police, les moments et les mesures courantes de la dépendance de y_i (p. ex., le tau de Kendall et le rho de Spearman) peuvent être calculés sous des formes simples. La VaR et l'ECU peuvent aussi être résolues numériquement sans grande difficulté. On peut appliquer plusieurs principes pour calculer les primes des contrats d'assurance, dont la prime pure, la

prime basée sur l'écart-type, la prime basée sur l'espérance limitée et la prime *stop-loss*. Les mesures de risque peuvent également être calculées pour chaque titulaire de police (p. ex., la VaR à 99 %). Au niveau du portefeuille, on peut effectuer une simulation pour obtenir la distribution des sinistres agrégés de l'ensemble des titulaires, ce qui est utile pour calculer les primes et la réserve totale pour sinistres. Le processus de simulation est facilité par le simulateur de données inclus dans notre progiciel (voir la section 4.5).

3. Exemple : Ensemble de données simulées

Nous montrerons ici comment ajuster un modèle LRMoE à l'aide de LRMoEDemoData, un ensemble de données simulées inclus dans le progiciel. Les variables de LRMoEDemoData sont décrites au tableau 2 et l'ensemble de données est généré par le modèle LRMoE défini au tableau 3. La description détaillée de l'ensemble de données de démonstration se trouve sur le site Web du progiciel. On peut charger l'ensemble de données en saisissant les lignes suivantes.

Tableau 2 : Description de LRMoEDemoData

Covariable[1]	**Nom**	**Description**
x_{i0}	intercept	Constante 1
x_{i1}	sex	1 pour homme, 0 pour femme
x_{i2}	agedriver	Âge du conducteur : 20–80
x_{i3}	agecar	Âge de la voiture : 0–10
x_{i4}	region	1 pour région urbaine, 0 pour région rurale
Réponse	**Nom**	**Description**
y_{i1}	Y[,1]	Nombre de sinistres relatifs à la branche d'assurance 1
y_{i2}	Y[,2]	Sévérité des sinistres relatifs à la branche d'assurance 2
Index des lignes	**Y[,1]**	**Y[,2]**
1–6000	Aucune censure ni troncature	Aucune censure ni troncature
6001–8000	Aucune censure ni troncature	Troncature à gauche à 5[2]
8001–10000	Aucune censure ni troncature	Censure à droite à 100

[1] Toutes les covariables sont générées de façon aléatoire, indépendante et uniforme.
[2] L'ensemble complet de données (X, Y) contient 10 000 lignes. En raison de la troncature à gauche Y[,2], 172 lignes de données sont retirées et l'ensemble de données observé (X_obs, Y_obs) compte seulement 9 828 lignes.

```
# The package and source code are available on github.
# The installation requires two lines of code.

> library(devtools)
```

```
> install_github("sparktseung/LRMoE")

# Load DemoData which contains four matrices
# X, Y are complete datasets with 10,000 rows
# X.obs, Y.obs are subject to data truncation and censoring

> library(LRMoE)
> data(LRMoEDemoData)
```

Pour ce qui est de traiter la troncature et la censure des données, les données que l'utilisateur doit saisir relativement à la réponse ***Y*** diffèrent de celles qu'il doit saisir dans les progiciels existants. Pour chaque dimension *d* de l'observation $\boldsymbol{y}_i$, au lieu d'une seule entrée numérique, un quadruple $0 \leq t_{id}^l \leq y_{id}^l \leq y_{id}^u \leq t_{id}^u \leq \infty$ est nécessaire, où t_{id}^l et t_{id}^u sont les bornes inférieure et supérieure de la troncature, et y_{id}^l et y_{id}^u sont les bornes inférieure et supérieure de la censure. La valeur exacte de y_{id} se situe entre les bornes de la censure, c'est-à-dire que $y_{id}^l \leq y_{id} \leq y_{id}^u$. Pour un échantillon de taille *n*, une $n \times (4D)\text{-matrix}$ est nécessaire, où chaque bloc $n \times 4$ décrit une dimension de ***Y***. Nous présentons ci-après des exemples de lignes de Y.obs dans DemoData.

```
# Complete data: no truncation, no censoring
> Y.obs[1,]
```

tl.1	yl.1	yu.1	tu.1	tl.2	yl.2	yu.2	tu.2
0.0000	5.0000	5.0000	Inf	0.0000	40.2224	40.2224	Inf

```
# Y[,2] is left-truncated at 5
> Y.obs[6002,]
```

tl.1	yl.1	yu.1	tu.1	tl.2	yl.2	yu.2	tu.2
0.0000	0.0000	0.0000	Inf	5.0000	81.3488	81.3488	Inf

```
# Y[,2] is right-censored at 100
> Y.obs[7884,]
```

tl.1	yl.1	yu.1	tu.1	tl.2	yl.2	yu.2	tu.2
0	7	7	Inf	0	100	Inf	Inf

Tableau 3 : Vrai modèle de LRMoEDemoData

Coefficients de régression logit :

$$\boldsymbol{\alpha} = \begin{bmatrix} -0.50 & 1.00 & -0.05 & 0.10 & 1.25 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Distributions des composantes :

	j = 1			*j* = 2		
	Comp.dist	δ_{jd}	$\boldsymbol{\psi}_{jd}$	Comp.dist	δ_{jd}	$\boldsymbol{\psi}_{jd}$
d = 1	Poisson	0	$(\lambda = 6)$	ZI-Comptage Gamma	0.20	$m = 30, s = 0.50$
d = 2	lnorm	0	$(\mu = 4.0, \sigma = 0.30)$	invgauss	0	$(\mu = 20.0, \lambda = 20.0)$

Tableau 4 : Modèle ajusté 1 de LRMoEDemoData

Coefficients de régression logit :

$$\hat{\boldsymbol{\alpha}} = \begin{bmatrix} -0.3927 & 0.9837 & -0.0493 & 0.0910 & 1.1527 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Distributions des composantes :

	j = 1			*j* = 2		
	Comp.dist	δ_{jd}	$\boldsymbol{\psi}_{jd}$	Comp.dist	δ_{jd}	$\boldsymbol{\psi}_{jd}$
d = 1	Poisson	0	$\lambda = 5.77$	ZI-Comptage Gamma	0.19	$m = 29.52, s = 0.49$
d = 2	lnorm	0	$\mu = 4.01, \sigma = 0.30$	invgauss	0	$\mu = 19.92, \lambda = 22.06$

Tableau 5 : Modèle ajusté 2 de LRMoEDemoData

Coefficients de régression logit :

$$\tilde{\alpha} = \begin{bmatrix} -0.3674 & 0.9811 & -0.0494 & 0.0917 & 1.1483 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Distributions des composantes :

	j = 1			*j* = 2		
	Comp.dist	δ_{jd}	ψ_{jd}	Comp.dist	δ_{jd}	ψ_{jd}
d = 1	ZI-Poisson	0.009	$\lambda = 5.87$	ZI-nbinom	0.195	$n = 30, p = 0.499$
d = 2	burr	0	$k = 1.12, c = 5.57, \lambda = 56.99$	gamma	0	$m = 1.67, \theta = 11.53$

Pour ajuster un modèle LRMoE, l'utilisateur doit spécifier les données suivantes : nombre de classes latentes (n.comp), distributions des composantes à utiliser (comp.dist) pour chaque dimension et chaque composante, et estimation initiale des paramètres du modèle (alpha.init pour les coefficients de régression logit, zero.init pour les probabilités de surreprésentation de zéros et params.init pour les paramètres de la partie positive des distributions de composantes).

À titre illustratif, nous supposons d'abord que le choix des distributions des composantes par l'utilisateur coïncide avec celui du vrai modèle et que les premières estimations des paramètres sont proches des vraies valeurs. Comme le montre l'exemple suivant de code d'initialisation, l'utilisateur a choisi un mélange à deux composants. Par exemple, pour la première dimension de la variable de réponse, l'utilisateur spécifie que la deuxième composante latente comp.dist[1,2] est une distribution de comptage gamma gonflée à zéro, où l'estimation initiale de zéro.init[1,2] avec surreprésentation de zéros est de 0,5 et les paramètres de comptage gamma params.init[[1]][[2]] sont (40; 0,8).

```
# Number of latent classes (component distributions)
  n.comp = 2    # = g
  # Number of response dimension
  dim.m = 2     # = d

  # A matrix of strings to specify component distributions # by dimension (row) and by
  component (column)
  # Dimension is d * g
  comp.dist = matrix( c("poisson", "ZI-gammacount",
  "lnorm",        "invgauss"), nrow = dim.m, byrow = TRUE)

  # Initial guesses of alpha: logit regression weights # by component (row) and by covariate
  (column)
  # Last row must be zero for default class # Dimension is g * (P+1)
  alpha.init = matrix( c(0, 0, 0, 0, 0,
  0, 0, 0, 0, 0),
  nrow = n.comp, byrow = TRUE)

  # Initial guesses of zero-inflation probabilities
  # by dimension (row) and by component (column)
  # Must be zero for non-zero-inflated component distributions
  zero.init = matrix( c(0, 0.5,
  0,      0),
  nrow = dim.m, byrow = TRUE)

  # Initial guesses of component distribution parameters # It is a d-length list (by dimension),
  # where each element is a g-length (by component) list of vectors
  params.init = list( list(c(10), c(40, 0.8)),
  list(c(3, 1), c(15, 15)))
```

Par défaut, la fonction d'ajustement impose une pénalité à la grandeur des paramètres afin d'éviter les faux modèles qui ont des valeurs de paramètres extrêmement grandes ou petites. Cela se fait en spécifiant des hyperparamètres pour les distributions a priori des paramètres du modèle. Pour simplifier les choses, on attribue des distributions a priori gamma aux paramètres positifs, des distributions a priori normales de moyenne nulle aux paramètres réels, tandis que les paramètres dont le domaine est borné ne sont pas pénalisés du tout.

Par exemple, les lignes suivantes indiquent que les distributions a priori sont les coefficients de régression logit $\alpha_{jp} \sim N(0, 5^2)$, la moyenne logarithmique $\mu \sim N(0, 1^2)$ de la log-normale, tandis que l'écart-type logarithmique de la log-normale suit une distribution gamma avec paramètre de forme 9 et paramètre d'échelle 0,5. Aucune pénalité n'est imposée aux probabilités de surreprésentation de zéros.

```
# Penalty for alpha: a single numeric for all logit regression weights
> hyper.alpha = 5


# Penalty for component distribution parameters
# List structure is the similar to params.init
> hyper.params = list( list(c(9, 0.5),                    c(9, 0.5, 9, 0.5)),
                       list(c(1, 9, 0.5), c(9, 0.5, 9, 0.5))
                     )
```

Lorsque tous les arguments d'entrée sont bien définis, la fonction d'ajustement du modèle peut être appelée comme suit. Il est possible d'imprimer à l'écran les valeurs intermédiaires des paramètres en fixant print = TRUE.

```
> fitted.model = LRMoEFit(Y = Y.obs, X = X.obs, n.comp = n.comp, comp.dist = comp.dist,
                          alpha.init = alpha.init,
                          zero.init = zero.init, params.init = params.init, penalty = TRUE,
                          hyper.alpha = hyper.alpha, hyper.params = hyper.params, print =
                          FALSE)
```

La fonction d'ajustement produira une liste qui contient la spécification du modèle (n.comp, comp.dist), les initialisations (p. ex., alpha.init), les paramètres estimés (p. ex., alpha.fit), la log-vraisemblance (ll, ll.np sans pénalité) et les critères d'information du modèle ajusté (Akaike, bayésien). Ils peuvent être inspectés au moyen des méthodes R standards. Des exemples sont donnés ci-dessous.

```
# Check LRMoE model specification
> fitted.model$comp.dist
```

comp 1 comp 2

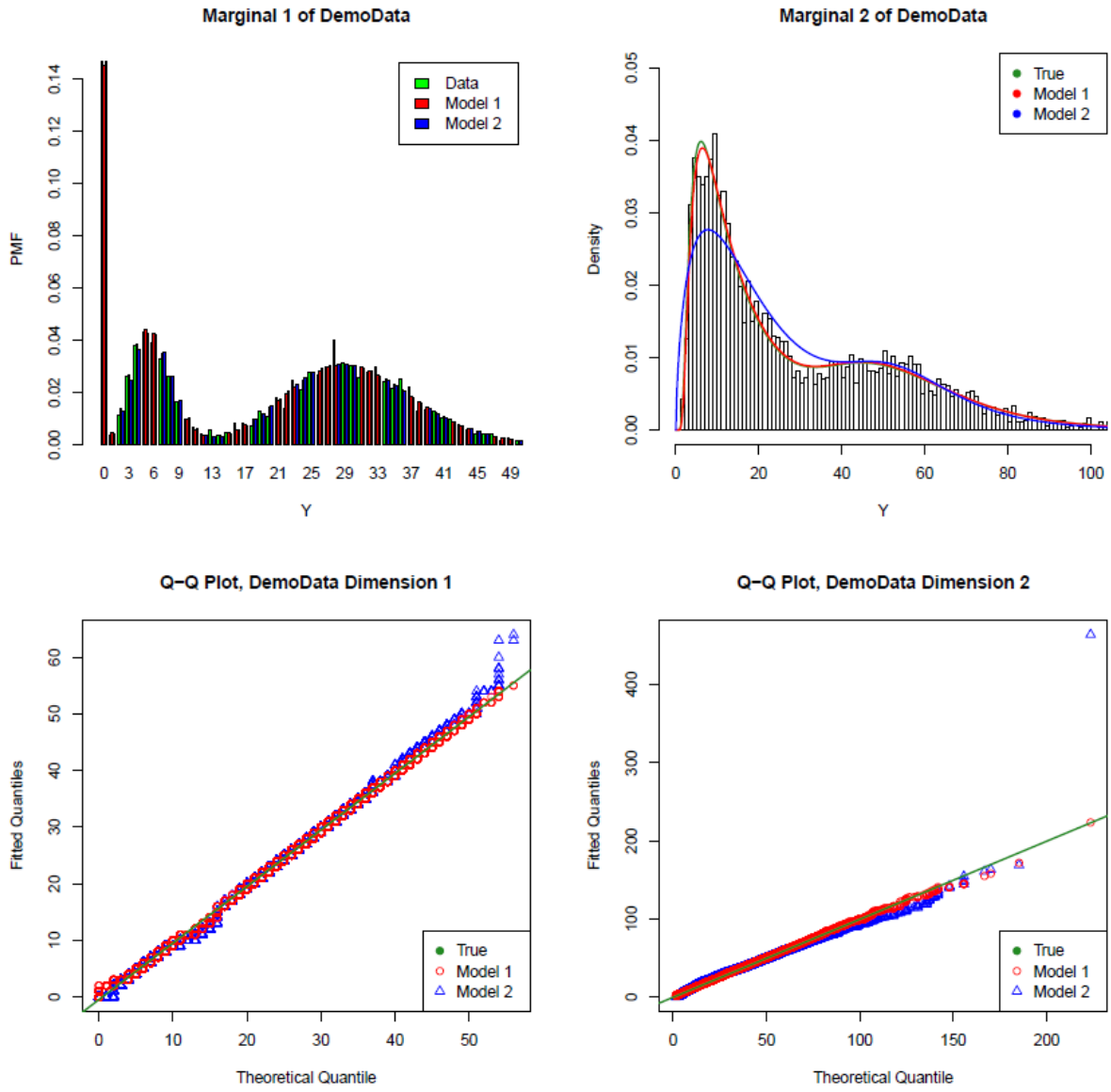
dim 1 "poisson" "ZI-gammacount" dim 2 "lnorm"
"invgauss"

 # Inspect fitted logit regression coefficients

> *fitted.model$alpha.fit*

	intercept	sex	agedriver	agecar	region
1	-0,3926849	0,9836549	-0,04925036	0,0910102	1,152698
comp. 2	0,0000000	0,0000000	0,0000000	0,0000000	0,0000000

Figure 1 : Résultats de l'ajustement de DemoData



*Cette figure est seulement disponible en anglais.

 # Inspect fitted parameters for the second dimension of Y

fitted.model$params.fit[["dim 2"]] # or fitted.model$params.fit[[2]]

$'comp 1'

Meanlog sdlog

4.0 0.3

$'comp 2'

Mean scale

19.77823 21.42792

Les résultats de l'ajustement du modèle sont résumés au tableau 4. Les estimations des paramètres sont assez proches des vraies valeurs. Compte tenu des bruits aléatoires simulés et de la perte d'information due à la censure et à la troncature, la fonction d'ajustement est en mesure d'identifier le modèle véritable lorsqu'il est connu.

En pratique, lorsque le vrai modèle sous-jacent n'est pas connu, l'utilisateur doit effectuer une analyse préliminaire de l'ensemble de données pour déterminer la spécification du modèle et l'initialisation des paramètres. Le tableau 5 renferme les estimations des paramètres d'un autre modèle LRMoE spécifié par l'utilisateur pour DemoData, modèle qui présente des distributions de composantes assez différentes par rapport au vrai modèle.

Les valeurs ajustées de log-vraisemblance des modèles 1 et 2 sont respectivement de -72 885,57 et -73 374,48. La figure 1 présente une comparaison graphique des deux modèles. Bien que les deux modèles aient une performance d'ajustement similaire en ce qui concerne la fréquence des sinistres, le modèle 2 est nettement plus mauvais pour l'ajustement des valeurs petites et extrêmes de la sévérité des sinistres.

4. Exemple : Ensemble de données réelles

Nous montrons ici comment ajuster un modèle LRMoE à un ensemble de données réelles d'assurance. Étant donné que nous avons abordé la procédure d'ajustement de base à la section précédente, nous traiterons ici du traitement préalable des données de notre progiciel, y compris l'initialisation des paramètres, la parallélisation de calcul, les fonctions de calcul actuariel des primes et la visualisation du modèle.

4.1 Description et prétraitement des données

Nous examinons l'ensemble de données sur les sinistres d'assurance automobile en France qui est inclus dans le progiciel R **CASdatasets** (Dutang et Charpentier 2019), qui peut être chargé comme suit.

```
> library(CASdatasets)
> data(freMTPLfreq, freMTPLsev)
```

L'ensemble de données freMTPLfreq contient des enregistrements de l'identificateur de police, de l'information sur le titulaire et sur le nombre de sinistres, tandis que freMTPLsev ne contient que l'identificateur de police et, le cas échéant, le montant des sinistres. La variable PolicyID sert d'identificateur unique qui relie ces deux ensembles de données. Certains titulaires ont enregistré plusieurs sinistres, ce qui produit des enregistrements en double de PolicyID dans freMTPLsev. À des fins de démonstration, nous regroupons les montants des sinistres par titulaire de police et choisissons uniquement la sévérité des sinistres comme variable de réponse.

```
# Aggregate claim amounts
> sev.aggre = aggregate(freMTPLsev$ClaimAmount,
                        by = list(PolicyID = freMTPLsev$PolicyID), FUN = "sum")
> colnames(sev.aggre)[2] = "ClaimAmount"


# Match two datasets by PolicyID. NA values correspond to no claims.
> df.all = merge(freMTPLfreq, sev.aggre, by = "PolicyID", all = TRUE)
> df.all$ClaimAmount[is.na(df.all$ClaimAmount)] = 0
```

L'ensemble de données qui en résulte comporte 413 169 observations. La distribution des montants des sinistres indique une forte surreprésentation de zéros, car moins de 4 % des titulaires de police ont enregistré un sinistre. En ce qui concerne les montants positifs des sinistres, la distribution est asymétrique à droite, multimodale et a une queue lourde. Les variables de l'ensemble de données regroupées sont décrites au tableau 6.

Tableau 6 : Description des données sur l'assurance automobile en France

Covariable	**Nom**	**Description**
x_{i0}	Intercept	Constante 1. Classe par défaut des variables catégoriques.
x_{i1}	CarAge	Âge du véhicule en années. Domaine : $0 \sim 100$
x_{i2}	DriverAge	Âge du conducteur en années. Domaine : $18 \sim 99$
$x_{i3} \sim x_{i,13}$	Power	Puissance du véhicule, sous forme de variable catégorique ordonnée : $\mathrm{d} \sim \mathrm{o}$. La valeur par défaut est « d ».
$x_{i,14} \sim x_{i,19}$	Brand	Marque du véhicule : 7 catégories. La valeur par défaut est « Fiat ».
$x_{i,20}$	Gas	Type de carburant du véhicule : diesel ou essence ordinaire. La valeur par défaut est « Diesel ».
$x_{i,21} \sim x_{i,29}$	Region	Région d'émission de la police en France : 10 catégories. La valeur par défaut est « Aquitaine »
Réponse	**Nom**	**Description**
y_{i1}	ClaimAmount	Montant du sinistre du titulaire de police

Les fonctions R standards servent à convertir la trame de données ci-dessus en matrices de covariables et la réponse requise par le progiciel **LRMoE**.

```
# Make Y matrix
> sample.size = nrow(df.all)
> Y = matrix( c(rep(0, sample.size),                # = tl
                df.all$ClaimAmount,    # = yl
                df.all$ClaimAmount,    # = yu
                rep(Inf, sample.size)  # = tu
                ),
           ncol = 4, byrow = FALSE)
# Make X matrix
> X.continuous = cbind(df.all$CarAge, df.all$DriverAge)
> X.power = model.matrix(~df.all$Power, data = df.all)          # Default is 'd'
```

```
> X.brand = model.matrix(~df.all$Brand, data = df.all)          # Default is 'Fiat'
> X.gas = model.matrix(~df.all$Gas, data = df.all)              # Default is 'Diesel'
> X.region = model.matrix(~df.all$Region, data = df.all) # Default is 'Aquitaine'
> X = matrix(cbind(rep(1, sample.size), # Intercept
                   X.continuous,
                   X.power[,-1], X.brand[,-1], X.gas[,-1], X.region[,-1]), nrow =
           sample.size, byrow = FALSE)
# Omitted code to name columns of X: colnames(X) = c(...)
```

4.2 Initialisation des paramètres

Vu que la procédure d'ajustement du LRMoE passe par une optimisation multivariée, une bonne initialisation des paramètres mènera souvent à une convergence plus rapide, comparativement à une estimation non informative. Gui et coll. (2018) ont proposé une procédure d'initialisation pour un mélange de distributions d'Erlang d'ajustement, qui a recours au partitionnement en k-moyennes et à la méthode des moments avec clusters. C'est ce qui a été utilisé dans Fung et coll. (2020) et cela produit des valeurs de départ des paramètres qui sont relativement bonnes.

Notre ensemble comprend une fonction d'initialisation qui applique la méthode des moments avec clusters à toutes les distributions de composantes et qui est utilisée conjointement avec la fonction de base kmeans. Une analyse préliminaire est nécessaire pour déterminer le nombre de clusters (composantes) à utiliser. Étant donné que la partie positive de toutes les distributions de notre progiciel est unimodale, un point de départ heuristique consiste à examiner l'histogramme empirique des données et à compter le nombre de pics (voir la figure 3). À titre d'exemple, nous décrivons ci-après la procédure d'initialisation d'un LRMoE à trois composantes.

```
# Number of components
> n.comp = 3
# Compute parameter initialization using CMM for severity
> init.3 = CMMSeverity(df.all$ClaimAmount, data.matrix(df.all.st), 3)
```

La fonction CMMSeverity produira une liste dans laquelle chaque élément contiendra la proportion du cluster, la surreprésentation de zéros et l'initialisation des paramètres des distributions de composantes. L'initialisation peut être inspectée au moyen des méthodes R standards.

Le tableau 7 résume la liste init.3 produite par l'exemple de code.

Tableau 7 : Exemple d'initialisation des paramètres

	Composante	1	2	3
Proportion		0,21	0,25	0,54
Surreprésentation de zéros		0,96	0,97	0,96
Données positives :				
- Moyenne		1736,80	2052,66	2487,96
- Coefficient de variation		2,21	3,00	11,26
- Asymétrie		10,50	12,54	57,72
- Aplatissement		143,77	193,40	3814,04
- Initialisation des paramètres				
Gamma	forme	0,20	0,01	0,008
	échelle	8509,01	18 474,62	315 719,40
Log-normale	moyenne logarithmique	6,57	6,48	5,39
	écart-type logarithmique	1,33	1,52	2,20
Gaussienne inverse	moyenne	1736,80	2052,66	3487,96
	échelle	354,51	228,06	19,61
Weibull[1]	forme[1]	3	3	3
	échelle	2605,21	3078,99	3731,94
Burr[1]	Forme1[1]	2	2	2
	Forme2[1]	5	5	5
	échelle	12 770,62	15 093,07	18 293,83

[1] Réglé à constant pour la stabilité numérique. Le paramètre d'échelle est obtenu par correspondance avec le premier moment.

La proportion de chaque cluster et sa surreprésentation de zéros peuvent être utilisées pour initialiser alpha.init et zero.init. Les statistiques sommaires des données positives sont fournies pour aider l'utilisateur à choisir une combinaison de distributions de composantes. Il faut éviter l'initialisation avec des valeurs de paramètre extrêmement grandes ou petites (p. ex., Gamma(0,0008; 315 714,40) pour la composante 3).

 # All components have a quite large proportion of zero, # so component distributions should be zero-inflated.

```
> comp.dist = matrix(c("ZI-lnorm", "ZI-lnorm", "ZI-lnorm"),
                     nrow = 1, byrow = TRUE)
# Assuming no knowledge of covariate influence on response,
# initialize only the intercept coefficient, where 3 is the reference group
> alpha.init = matrix(0, nrow = 3, ncol = 30)
> alpha.init[,1] = log(c(0.21, 0.25, 0.54)) - log(0.54)
# Initialize zero inflation: all components are zero-inflated.
> zero.init = matrix(0, nrow = 1, ncol = 3)
> zero.init[1,] = c(0.96, 0.97, 0.96)
# Initialize component distribution parameters.
> params.init = list( list(c(6.57, 1.33), c(6.48, 1.52), c(5.39, 2.20)) )
```

4.3 Calculs parallèles

Bien que le modèle LRMoE soit très flexible, l'utilisateur doit essayer différentes combinaisons de distributions de composantes ou différentes initialisations de paramètres afin de trouver le modèle qui s'ajuste le mieux aux données. Si la fonction d'ajustement est appelée séquentiellement pour chaque spécification du modèle, le temps de calcul varie en fonction du nombre de modèles LRMoE candidats, ce qui n'est pas souhaitable, surtout dans le cas de grands ensembles de données. Nous montrerons ici comment intégrer notre progiciel avec doParallel (Microsoft Corporation et Weston 2019) afin que plusieurs modèles LRMoE puissent être ajustés simultanément.

Nous supposons que l'utilisateur a spécifié un ensemble de modèles de modèles LRMoE candidats. Chacun d'eux est représenté sous forme de liste, qui contient une chaîne model.name en tant qu'identificateur, spécification du modèle et initialisation des paramètres, tel qu'il est décrit à la section 3.

```
# The model.list contains a list of LRMoE model candidates to fit. # model.list = list(model.1,
model.2, ...)
# Each model is structured as a list of inputs # required by the fitting function.
# model.1 = list(model.name, n.comp, comp.dist,
                 alpha.init, zero.init, params.init, hyper.alpha, hyper.params)
```

La fonction d'ajustement sera appelée dans une fonction de boucle do_fitting, où chaque itération produit un modèle LRMoE. Pour faciliter l'analyse des résultats, il est recommandé de sauvegarder le modèle ajusté en tant que fichier de type .Rda, ainsi que le fichier de sortie intermédiaire en tant que fichier de type .txt. De plus, l'utilisateur peut choisir d'obtenir des mises à jour par courriel sur l'état d'exécution si le code est exécuté sur un serveur à distance (voir, par exemple, Premraj 2015).

```
do_fitting = function(X, Y, model)
{
  model.name = toString(paste(model$model.name, sep="")) rda.name =
  toString(paste(model.name, ".Rda", sep=""))
  output.name = toString(paste(model.name, ".txt", sep=""))
  # Set a file to save the intermediate update of parameters sink (file = output.name)
  tryCatch({model.fit = LRMoE.fit(Y = Y, X = X,
                n.comp = model$n.comp, comp.dist = model$comp.dist, alpha.init =
                model$alpha.init, zero.init = model$zero.init, params.init =
                model$params.init,
                penalty = TRUE,
                hyper.alpha = model$hyper.alpha, hyper.params = model$hyper.params,
                print = TRUE)
            save(model.fit, file = rda.name) # Save fitted model
            },
            error=function(e){"Error!"; print("Error!")}
           )
  # Save intermediate update of parameters for the current model sink()
  # Optional: use mailR to get running status. Code is omitted.
}
```

Enfin, la fonction do_fitting est appelée en parallèle. Selon les ressources informatiques de l'utilisateur, le nombre de modèles à ajuster en parallèle (ncore) peut différer.

```
# Specify how many models to fit in parallel
> ncore = 5
> n.run = length(model.list)
# Make computing clusters: standard procedure
> cl = makePSOCKcluster(ncore)
> registerDoParallel(cl)
```

 # Call fitting functions in parallel

> foreach(b = 1:n.run) %dopar% {do_fitting(X, Y, model.list[[b]])}

 # Stop computing clusters: standard procedure

> stopCluster(cl)

4.4 Résultats d'ajustement et sélection de modèles

À titre illustratif, nous ajustons seulement un certain nombre de LRMoE à tout l'ensemble de données françaises d'assurance automobile. Le tableau 8 résume la log-vraisemblance, le CIA et le CIB ajustés.

Tableau 8 : Résultats d'ajustement des données françaises pour l'assurance automobile

***g* = 3**	**Log-vraisemblance**	**CIA**	**CIB**	***g* = 4**	**Log-vraisemblance**	**CIA**	**CIB**
lll	-183 913	367 965	368 719	lill	-183 521	367 246	368 361
iwl	-183 928	367 993	368 748	llll	-183 749	367 702	368 817
wll	-184 078	368 293	369 047	bill	-183 784	367 774	368 900
llb	-184 099	368 337	369 102	will	-183 798	367 800	368 915
lib	-184 112	368 363	369 129	bilw	-183 954	368 114	369 240
***g* = 5**	**Log-vraisemblance**	**CIA**	**CIB**	***g* = 6**	**Log-vraisemblance**	**CIA**	**CIB**
bwliw	-183 576	367 424	368 910	llblll	-183 444	367 226	369 074
bllil	-183 590	367 452	368 938	liwibl	-183 450	367 238	369 085
lllil	-183 603	367 476	368 951	llwlll	-183 466	367 269	369 105
lllll	-183 604	367 478	368 954	liwlll	-183 471	367 279	369 115
wllil	-183 654	367 578	369 054	liblll	-183 491	367 321	369 168

Note : Les distributions de composantes sont représentées par la première lettre; par exemple, l pour log-normale, b pour Burr, etc. Toutes les composantes sont gonflées à zéro. Le critère d'arrêt est une amélioration de la log-vraisemblance < 0.05, ou 500 itérations. Pour chaque *g*, les valeurs de log-vraisemblance sont en ordre décroissant. Les valeurs optimales globales sont en caractères gras.

Dans la plupart des cas, l'ajout de composantes augmentera la log-vraisemblance ajustée (p. ex., considérons les modèles iwl, will, wllil et liwlll). Les modèles sélectionnés par le CIA et le CIB ont respectivement six et quatre composantes. Dans ce contexte particulier, le critère d'information bayésien pénalise fortement les modèles comportant beaucoup de composantes, car la taille de l'échantillon est grande et l'ajout d'une composante augmente grosso modo le nombre de paramètres de 30 (le nombre de coefficients de régression logit).

Outre le CIA et le CIB, la validation croisée constitue un autre critère de sélection de modèles qui permet d'éviter le surajustement d'un trop grand nombre de composantes latentes. Par exemple, Gui et coll. (2018) ont étudié une validation croisée à 10 blocs pour un mélange de distributions d'Erlang d'ajustement, où la log-vraisemblance moyenne sur les ensembles de test est utilisée comme fonction de score pour sélectionner le nombre optimal de composantes.

Pour mettre en place la validation croisée, notre progiciel fournit une fonction de calcul de la log-vraisemblance d'un modèle ajusté à un ensemble de données de test, qui peut être intégrée à la fonction do_fitting ci-dessus avec des données additionnelles des ensembles de test X.test et Y.test.

```
# X.test, Y.test: Test sets formatted as required by LRMoE #
model.fit: A fitted model returned by LRMoE.fit function

> LRMoE.loglik(X = X.test, Y = Y.test, model = model.fit)
```

4.5 Fonctions de calcul de primes et de réserves

Notre progiciel comprend un ensemble de fonctions se rapportant au calcul actuariel des primes, au calcul des réserves, à la gestion du risque, y compris le calcul de la moyenne, de la variance, de la VaR, de l'ECU, de l'espérance limitée $E[(Y \quad u)]$ et de la prime *stop-loss* $E[(Y - d)_+]$ de la variable de réponse. Ces fonctions commencent par la racine predict, suivie des quantités d'intérêt appropriées (moyenne, VaR, quantile, ECU, limite, excédent) et les arguments de fonction correspondants.

À titre d'illustration, considérons les trois titulaires de police du tableau 9, où A n'a pas enregistré de sinistre, B en a enregistré un de taille moyenne et C en a enregistré un gros. Voici un exemple de code permettant de calculer différentes quantités d'intérêt :

```
# Mean and variance of claim amount of Policyholders A, B and C.

# Variance is infinite due to Burr component.

> PredictMeanPrior(X[c(1, 33, 96),],
```

Tableau 9 : Titulaires sélectionnés parmi l'ensemble de données françaises sur l'assurance automobile

	Sinistre	ID	Âge de la voiture	Âge du conducteur	Puissance du véhicule	Marque	Type de carburant	Région
A	0	1	0	46	g	JK	Diesel	A
B	302	33	1	61	g	JK	Essence ordinaire	IF
C	10 870	96	0	51	j	JK	Essence ordinaire	IF

Tableau 10 : Calcul des tarifs pour les titulaires sélectionnés parmi l'ensemble de données françaises sur l'assurance automobile

Principe de calcul des primes		A	B	C
Au niveau individuel	$E[Y]$	97,49	90,25	84,65
	$E[(Y-1000)_+]$	78,76	69,81	58,29
	$E[Y \wedge 100000]$	63,24	60,86	63,33
Au niveau du portefeuille	VaR(90)	108,65	100,59	94,34
	VaR(95)	121,14	112,15	105,19
	ECU(70)	117,20	108,51	101,77
	ECU(80)	126,58	117,19	109,91
	ECU(90)	149,28	138,20	129,62

Note : Pour ce qui est des principes de calcul des primes au niveau du portefeuille, la répartition est fonction du montant relatif de la prime pure, où le poids pour le titulaire de police *i* est $w_i = E[Y_i] / \sum_j E[Y_j]$. Pour chaque titulaire et chaque principe, le calcul est effectué en fonction de la probabilité a priori (colonne de gauche) et en fonction de la probabilité a posteriori (colonne de droite).

+ model.fit$alpha.fit, model.fit$comp.dist,

+ model.fit$zero.fit, model.fit$params.fit)

[,1]

[1,] 97.48500

[2,] 90.25394

[3,] 84.64872

 # 99% VaR of claim amount of Policyholders A, B and C.

> PredictVaRPrior(X[c(1, 33, 96),],

+ model.fit$alpha.fit, model.fit$comp.dist,

+ model.fit$zero.fit, model.fit$params.fit,

+ prob = 0.99)

[,1]

[1,] 1209.099

[2,] 1203.652

[3,] 1249.037

 # Mean excess of claim amount (d=1000) of Policyholders A, B and C.

```
> PredictLimExPrior(X[c(1, 33, 96),],
+ model.fit$alpha.fit, model.fit$comp.dist,
+ model.fit$zero.fit, model.fit$params.fit,
+ limit = 1000)
[,1]
[1,] 18.72962
[2,] 18.28337
[3,] 26.36306
```

Le calcul actuariel des primes peut s'effectuer en fonction des distributions individuelles des sinistres ou de la distribution agrégée des sinistres de l'ensemble du portefeuille. Ces deux méthodes peuvent être réalisées au moyen des fonctions intégrées dans notre progiciel.

Au niveau individuel, les fonctions précitées peuvent être appelées directement pour calculer la prime pure $E[Y]$, ainsi que la valeur de la prime lorsqu'il existe une franchise de police ($E[(Y - d)_+]$) ou une limite d'assurance ($E[(Y \wedge u)]$).

La première partie du tableau 10 résume le calcul des primes des titulaires A, B et C, selon les principes de calcul au niveau individuel.

Au niveau du portefeuille, la fonction dataset.simulator (voir la section 4.6) simulera une valeur de réponse (c.-à-d. le montant du sinistre) pour chaque titulaire de police, qui peut être additionnée aux sinistres agrégés du portefeuille selon un scénario possible. En réalisant plusieurs simulations répétées de la totalité du portefeuille, on obtiendra la distribution empirique des sinistres agrégés. La VaR et l'ECU des sinistres agrégés peuvent être obtenues à partir de l'échantillon simulé et sont utiles pour établir la réserve totale de l'assureur. En outre, la VaR et l'ECU peuvent être réaffectées aux titulaires sous forme d'une prime majorée, selon un régime de pondération qui reflète le degré de risque relatif des titulaires (p. ex., la grandeur relative de leur prime pure). La seconde partie du tableau 10 résume le calcul des primes des titulaires A, B et C selon les principes de calcul au niveau du portefeuille.

4.6 Visualisation des modèles

Après avoir ajusté et choisi un modèle LRMoE adéquat, l'utilisateur peut le visualiser au moyen des fonctions de traçage intégrées du progiciel ou en créant des tracés personnalisés au moyen des fonctions de simulation génériques (p. ex., data.similator) combinées à des fonctionnalités de traçage de base de R ou d'autres progiciels spécialisés (p. ex., **ggplot2**). Nous utiliserons ici le modèle llblll à six composantes pour la démonstration.

Probabilités des classes latentes

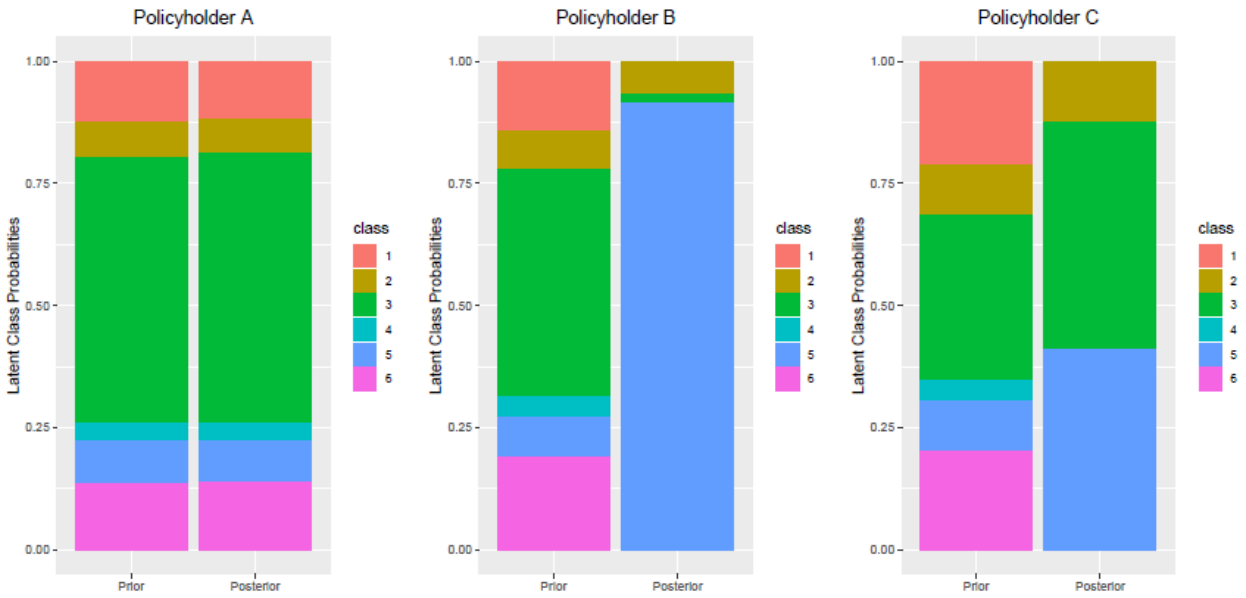
La régression logit dans le LRMoE affecte chaque titulaire de police à une classe de risque latente en se basant sur les covariables. Étant donné un mode ajusté et un vecteur de covariables, la probabilité des classes latentes peut être calculée et visualisée à l'aide des fonctions intégrées predict.class.prob et plot.ind.class.prob .

Dans le cas des titulaires dont l'historique des sinistres est connu, il peut être plus instructif de tenir compte des probabilités a posteriori des classes latentes en appelant les fonctions correspondantes des probabilités a posteriori. Considérons de nouveau les titulaires au tableau 9. Le code servant à calculer et à tracer les probabilités a posteriori est illustré ci-dessous, et les tracés se trouvent à la figure 2.

```
# Predict latent class probabilities, based on covariates and a model
> PredictClassPrior(X[c(1,33,96),], model.fit$alpha.fit)
```

	comp 1	comp 2	comp 3	comp 4	comp 5	comp 6
[1,]	0,1213162	0,07381480	0,5416097	0,03613851	0,08855876	0,1385620
[2,]	0,1404270	0,07858723	0,4647998	0,04097918	0,08278394	0,1924228
[3,]	0,2082516	0,10479211	0,3364129	0,04198693	0,10295946	0,2055970

Figure 2 : Probabilités a priori et a posteriori des classes latentes pour certains titulaires



Note : La composante 3 (Burr(1,41; 1; 24 073)) correspond à la queue, tandis que la composante 5 (Log-normale(6,23; 1,57)) correspond au plus gros pic de l'ensemble de données. Cette figure est seulement disponible en anglais.

```
# Predict posterior probabilities, based on covariates, history and a model
> PredictClassPosterior(X[c(1,33,96),], Y[c(1,33,96),],
```

```
+          model.fit$alpha.fit, model.fit$comp.dist,
+          model.fit$zero.fit, model.fit$params.fit)
```

	comp 1	comp 2	comp 3	comp 4	comp 5	comp 6
[1,]	1,174421e-01	0,06931930	0,5521331	3,399789e-02	0,08667928	1,404283 e-01
[2,]	2,577365e-294	0,06546023	0,0185750	1,509372 e-230	0,91596478	8,194626e-25
[3,]	0,000000e+00	0,12230349	0,4648738	0,000000e+00	0,41282273	0,000000e+00

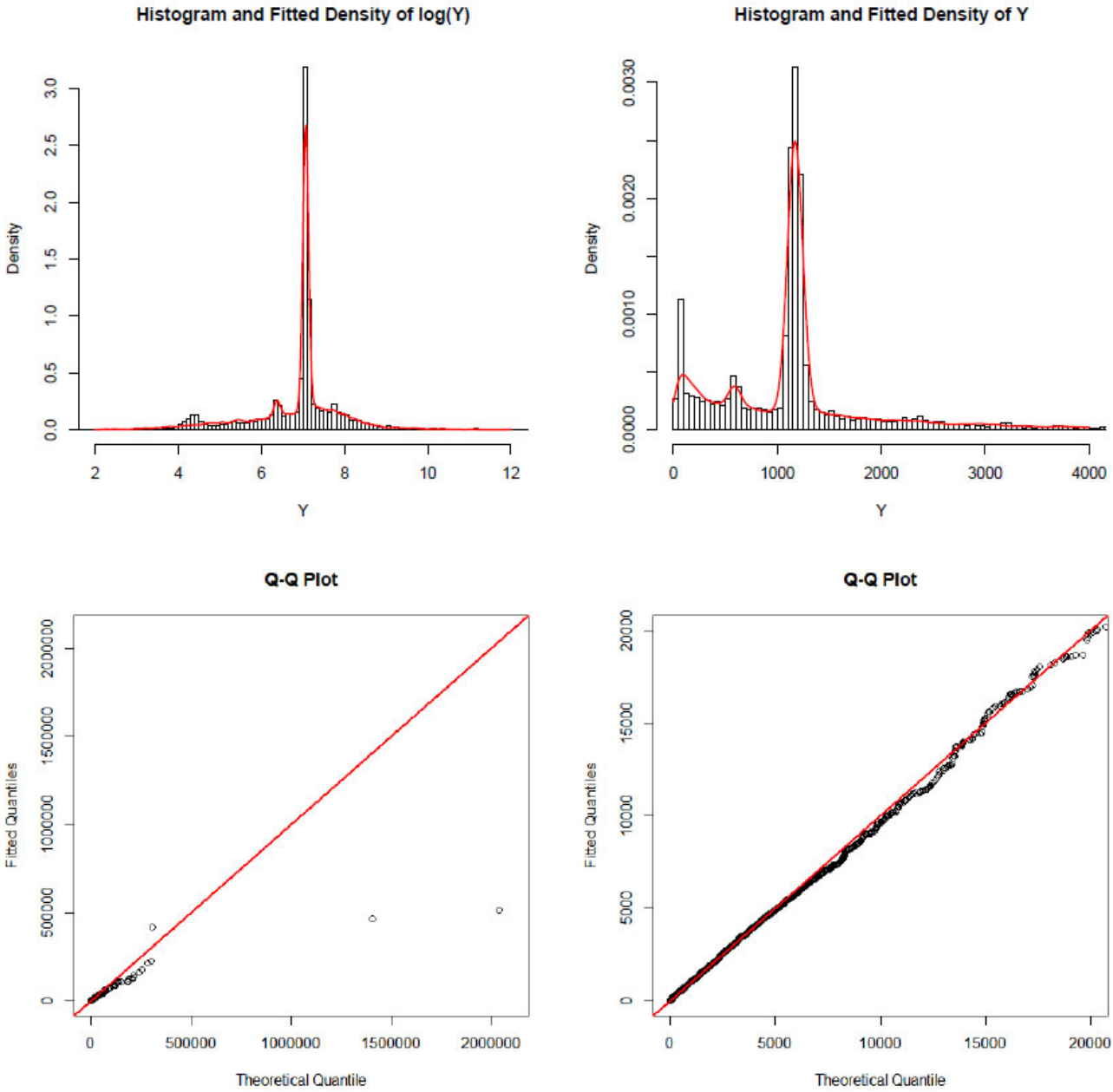
```
# Plot latent class probabilities for Policyholder A
# Function returns a ggplot2 object: optional to edit plot title
> PlotPropPosterior(Y[1,], X.1, model.fit$alpha.fit, model.fit$comp.dist,
+          model.fit$zero.fit, model.fit$params.fit) +
+          ggtitle("Policyholder A")
```

Qualité globale de l'ajustement

On peut examiner la qualité globale de l'ajustement en comparant l'histogramme empirique à la courbe de densité ajustée et au diagramme Q-Q des quantiles empiriques et ajustés. Ces tracés peuvent être produits avec la fonction data.simulator comprise dans notre progiciel, combinée aux fonctions de traçage de base dans R. Les graphiques correspondants sont présentés à la figure 3.

```
# Simulate exact responses, given a set of covariates and a fitted model
> sim.size = nrow(X)
> model.sim = SimYSet(X, model.fit$alpha.fit, model.fit$comp.dist,
                                  model.fit$zero.fit, model.fit$params.fit)
```

Figure 3 : Qualité globale de l'ajustement des sinistres positifs dans l'ensemble de données françaises sur l'assurance automobile



Note : (À gauche : tous les points de données; à droite : points de données non extrêmes). Cette figure est seulement disponible en anglais.

 # Only use positive values for plotting density

> Y.pos = Y[which(Y[,2]>0),2]

> sim.Y.pos = model.sim[which(model.sim[,1]>0),1]

```
# Use standard R functions for plotting histograms (all data points)
>     hist(log(Y.pos), breaks = 100, xlim = c(2, 12), probability = TRUE,
+     xlab = "Y", main = "Histogram and Fitted Density of log(Y)")
>     lines(density(log(sim.Y.pos), from = 2, to = 12), col = "red", lwd = 2)

# Use standard R functions for Q-Q plots
>     qqplot(Y[,2], model.sim[,1], main = "Q-Q Plot",
+     xlab = "Theoretical Quantile", ylab = "Fitted Quantiles")
>     abline(a = 0, b = 1, col = "red", lwd = 2)
```

Influence des covariables

En apprentissage automatique, on utilise couramment le diagramme de dépendance partielle pour étudier l'influence d'une covariable particulière sur la réponse, en supposant l'indépendance entre les covariables (Friedman 2001). Par exemple, l'effet marginal d'une covariable sur le montant moyen des sinistres peut être obtenu en suivant les étapes suivantes :

1. Attribuer une valeur particulière à la covariable (disons, marque de véhicule = « F ») pour tous les titulaires, tandis que les autres covariables demeurent inchangées.
2. Utiliser la fonction PredictMeanPrior pour calculer les valeurs de réponse moyennes par titulaire de police (voir la section 4.5).
3. Calculer la moyenne globale de toutes les valeurs à l'étape (2), qui fait la moyenne des effets des autres covariables et donne la valeur de réponse moyenne lorsque la marque de véhicule est de type « F ».
4. Répéter les étapes (1) à (3) pour une plage de valeurs de la même covariable d'intérêt; par exemple, substituer « VAS » à « F » comme marque de véhicule.

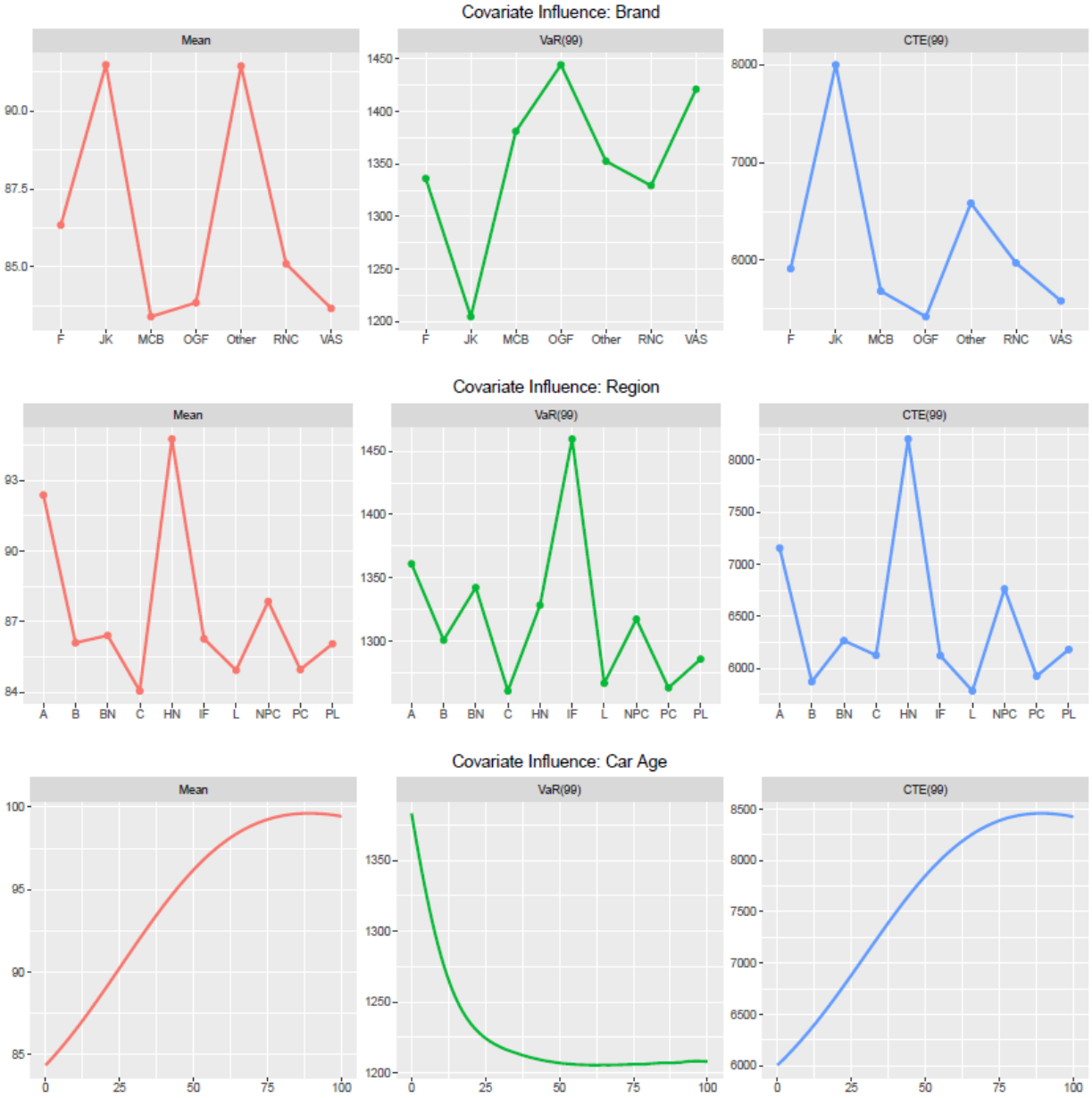
La procédure ci-dessus peut être généralisée à des covariables continues et à d'autres quantités d'intérêt, comme le quantile de la variable de réponse. Le tracé de l'influence des covariables illustre de façon graphique comment les caractéristiques d'un titulaire de police sont associées à une mesure particulière de la variable réponse. Par exemple, le tracé de la moyenne (ou VaR/ECU) de la variable de réponse en fonction de la marque du véhicule reflète la relation entre le degré de risque global (ou risque de queue de distribution) d'un titulaire de police et la marque du véhicule. Pour ce qui est de la mise en œuvre, un exemple de script est fourni sur le site Web de démonstration du progiciel.

Tableau 11 : Influence des covariables : marque de véhicule

Marque	Moyenne	VaR(99)	ECU(99)
F	86,34	1336,12	5913,98
JK	91,48	1204,76	7999,49
MCB	83,41	1381,02	5684,49
OGF	83,85	1444,18	5422,59
Divers	91,44	1352,56	6583,96
RNC	85,10	1329,38	5972,67
VAS	83,67	1421,00	5583,46

La figure 4 montre l'influence des covariables marque, région et âge du véhicule, qui peut être interprétée comme suit. En ce qui concerne la marque du véhicule, le modèle JK (voitures japonaises sauf Nissan, ou coréennes) est généralement plus risqué. Sur le plan du risque de queue de distribution, il est intéressant de constater que le modèle JK présente la VaR la plus faible, mais l'ECU la plus élevée comparativement aux autres modèles. Par ailleurs, comparativement aux autres régions, les polices émises dans les régions A (Aquitaine) et HN (Haute-Normandie) peuvent être considérées comme plus risquées. De plus, plus les voitures sont âgées, plus elles sont risquées.

Figure 4 : Influence des covariables



*Cette figure est seulement disponible en anglais.

5. Sommaire et perspectives

Nous avons présenté ici un nouveau progiciel R, **LRMoE**, qui est utile à la modélisation actuarielle des sinistres. Dans cette première version, le progiciel répond au besoin le plus fondamental, à savoir ajuster un modèle LRMoE, en plus d'aider l'utilisateur à visualiser le modèle ajusté et à calculer la prime d'assurance. Nous avons prévu plusieurs projets, notamment :

- Outils de sélection de modèles : Avec la version actuelle, la sélection des modèles s'effectue au moyen du CIA, du CIB ou de la validation croisée, ce qui oblige l'utilisateur à choisir et à exécuter une sélection de modèles. Certaines procédures automatisées de sélection de modèles pourraient être intégrées au progiciel (p. ex., pénalité de type SCAD dans Fan et Li 2001, et Yin et Lin 2016).
- Outils de sélection de caractéristiques : Les ensembles de données contiennent habituellement un grand nombre de covariables, mais ils ne sont pas tous importants en tant que prédicteurs de la réponse. Même si les tracés de l'influence des covariables peuvent donner une idée de leur importance relative, il est peut-être plus utile de quantifier leur influence et d'offrir une fonction permettant de choisir automatiquement les covariables les plus influentes.
- Interface utilisateur : À la section 4, l'étape de prétraitement des données exige de l'utilisateur qu'il convertisse manuellement les données dans un format particulier requis par le progiciel. La meilleure façon d'y parvenir serait d'ajouter des fonctions génériques de prétraitement dans le progiciel ou de permettre l'interprétation de la formule (comme LRMoE.fit(formula = y ~ x)).

Ouvrages de référence

Blostein, M. et T. Miljkovic. « On modeling left-truncated loss data using mixtures of distributions », *Insurance: Mathematics and Economics*, **vol. 85**, 2019, pp. 35–46, ISSN 0167-6687.

Dempster, A.P., Laird, N.M. et D.B. Rubin. « Maximum likelihood from incomplete data via the EM algorithm » *Journal of the Royal Statistical Society: Series B (Methodological)*, **vol. 39**, n° 1, 1977, pp. 1–22.

Dutang, C. et A. Charpentier. *CASdatasets : Insurance datasets*, version 1.0-10 du progiciel R, 2019.

Fan, J. et R. Li. « Variable selection via nonconcave penalized likelihood and its oracle properties », *Journal of the American Statistical Association*, **vol. 96**, n° 456, 2001, pp. 1348–1360.

Friedman, J.H. « Greedy function approximation: A gradient boosting machine », *Annals of Statistics*, **vol. 29**, n° 5, 2001, pp. 1189–1232.

Fung, T.C., Badescu, A.L. et X.S. Lin. « A class of mixture of experts models for general insurance: Application to correlated claim frequencies », *ASTIN Bulletin*, **vol. 49**, n° 3, 2019a, pp. 647–688.

Fung, T.C., Badescu, A.L. et X.S. Lin. « A class of mixture of experts models for general insurance: Theoretical developments », *Insurance: Mathematics and Economics*, **vol. 89**, 2019b, pp. 111–127.

Fung, T.C., Badescu, A.L. et X.S. Lin. « A new class of severity regression models with an application to IBNR prediction », *North American Actuarial Journal*, **vol. 25**, n° 2, 2020, pp. 1–26.

Fung, T.C., Badescu, A.L. et X.S. Lin. « Fitting censored and truncated regression data using the mixture of experts models », 2021. Sur SSRN: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3740061

Grün, B. et F. Leisch. « FlexMix version 2: Finite mixtures with concomitant variables and varying and constant parameters », *Journal of Statistical Software*, **vol. 28**, n° 4, 2008, pp. 1–35.

Gui, W., Huang, R. et X.S. Lin. « Fitting the Erlang mixture model to data via a GEM-CMM algorithm », *Journal of Computational and Applied Mathematics*, **vol. 343**, 2018, pp. 189–205.

Jiang, W., et M.A. Tanner. « On the identifiability of mixtures-of-experts », *Neural Networks*, **vol. 12**, n° 9, 1999, pp. 1253–1258.

Jordan, M.I. et R.A. Jacobs. « Hierarchical mixtures of experts and the EM algorithm », *Neural Computation*, **vol. 6**, n° 2, 1994, pp. 181–214.

Lee, D., Li, W.K. et T.S.T Wong. « Modeling insurance claims via a mixture exponential model combined with peaks-over-threshold approach », *Insurance: Mathematics and Economics*, **vol. 51**, n° 3, 2012, pp. 538–550.

Leisch, F. « FlexMix: A general framework for finite mixture models and latent class regression in R », *Journal of Statistical Software*, **vol. 11**, n° 8, 2004, pp. 1–18.

McLachlan, G. et D. Peel. *Finite Mixture Models*, John Wiley & Sons, 2004.

Meng, X.L. et D.B. Rubin. « Maximum likelihood estimation via the ECM algorithm: A general framework », *Biometrika*, vol. **80**, n° 2, 1993, pp. 267–278, ISSN 0006-3444.

Microsoft Corporation et S. Weston. *DoParallel: Foreach Parallel Adaptor for the "parallel" Package*, version 1.0.15 du progiciel R, 2019, https://CRAN.R-project.org/package=doParallel

Miljkovic, T. et B. Grün. « Modeling loss data using mixtures of distributions », *Insurance: Mathematics and Economics*, **vol. 70**, 2016, pp. 387–396, ISSN 0167-6687.

Premraj, R. (2015). *MailR: A utility to send emails from R*, version 0.4.1 du progiciel R, 2015, https://CRAN.R-project.org/package=mailR

Scollnik, D.P. et C. Sun. « Modeling with Weibull-Pareto models », *North American Actuarial Journal*, **vol. 16**, n° 2, 2012, pp. 260–272.

Yin, C. et X.S. Lin. « Efficient estimation of Erlang mixtures using iSCAD penalty with insurance application », *ASTIN Bulletin*, **vol. 46**, n° 3, 2016, pp. 779–799.

Institut canadien des actuaires
360, rue Albert, bureau 1740
Ottawa, ON K1R 7X7
613-236-8196
siege.social@cia-ica.ca

cia-ica.ca

voiraudeladurisque.ca

L'Institut canadien des actuaires (ICA) est l'organisme de qualification et de gouvernance de la profession actuarielle au Canada. Nous élaborons et maintenons des normes rigoureuses, partageons notre expertise en gestion du risque et faisons progresser la science actuarielle pour le bien-être financier de la société. Nos plus de 6 000 membres utilisent leurs connaissances en mathématiques, en statistiques, en analyses de données et en affaires dans le but de prodiguer des services et des conseils de la plus haute qualité pour aider à assurer la sécurité financière de toute la population canadienne.