Projet de simulation

G13

ISFA / M1 Actuariat 2024-2025

Auteurs :

MELI KOUYEM Frege

D'ALMEIDA KOMIVI Jules

SIANE Ange

MAMA Mariana

Contexte

On considère une assurance concernant les accidents de la route des motards. La pluie ayant un impact important sur les risques encourus par les motards, on décide de modéliser la météo. Et de déterminer le montant probable d'indemnisation.

Organisation du travail en groupe

Membres —

SIANE Ange

D'ALMEIDA Dedekpesse Komivi

MELI KOUYEM Frege

MARIAMA Mama

Organisation

Concernant notre organisation, nous nous sommes répartis le travail lors des premières séances. Ainsi, une personne était chargée d'étudier l'implémentation de msa notamment (Frege MELI), tandis qu'une autre étudiait msb (Ange SIANE). Ensuite avec la coopération de Mariama MAMA nous avons tous contribuer à l'implémentation la chaîne de Markov et le calcul des du nombre d'accident. Très vite, nous avons regroupé le code et chacun a regardé l'entièreté du code. Nous ne pouvons donc pas attribuer une partie du code à une personne spécifique à l'exception de la méthode d'implémentation, d'approximation et d'optimisation de rremb qui a été réalisée par D'ALMEIDA Komivi, aussi l'implémentation de msb.c réalisé par Frege MELI et son optimisation par Ange SIANE.

Outils utilisés

Pour optimiser notre collaboration, nous avons créé un groupe WhatsApp où nous avions la possibilité d'échange entre nous sur le projet. Enfin, le rapport a été rédigé en Rmarkdown. Et tous le code a été effectué grâce au language et grâce à des recherches effectués et aussi l'utilisation de l'IA pour la vérification du code

1 Calcul du coefficient de normalisation λ

1.1 Position du problème

Soit $f^*(x) = (\alpha + |x - x_0|)^{-\eta}$. Pour obtenir une densité de probabilité, nous devons déterminer λ tel que λf^* soit d'intégrale 1 sur $\mathbb{R}^+$.

1.2 Méthode de calcul

1.2.1 Condition de normalisation

On cherche λ tel que :

$$\int_0^\infty \lambda f^*(x)dx = 1$$

avec

$$f^*(x) = (\alpha + |x - x_0|)^{-\eta}$$

1.2.2 Décomposition de l'intégrale

En tenant compte de la valeur absolue, on décompose l'intégrale :

$$\int_0^\infty \lambda f^*(x)dx = \lambda \left(\int_0^{x_0} (\alpha + (x_0 - x))^{-\eta} dx + \int_{x_0}^\infty (\alpha + (x - x_0))^{-\eta} dx \right) = 1$$

1.2.3 Changements de variable

Nous effectuons les changements suivants :

- Pour $[0, x_0]$: $u = \alpha + (x_0 - x)$, $du = -dx$
  - $x = 0 \Rightarrow u = \alpha + x_0$
  - $x = x_0 \Rightarrow u = \alpha$
- Pour $[x_0, \infty[$: $v = \alpha + (x - x_0)$, $dv = dx$
  - $x = x_0 \Rightarrow v = \alpha$
  - $x \to \infty \Rightarrow v \to \infty$

1.2.4 Résolution

L'équation devient :

$$\lambda \left(-\int_{\alpha+x_0}^{\alpha} u^{-\eta} du + \int_\alpha^\infty v^{-\eta} dv \right) = 1$$

Après intégration :

$$\lambda \left[\frac{u^{1-\eta}}{1-\eta} \right]_{\alpha+x_0}^{\alpha} + \lambda \left[\frac{v^{1-\eta}}{1-\eta} \right]_\alpha^\infty = 1$$

Comme $\eta > 1$, $\lim_{v\to\infty} v^{1-\eta} = 0$, donc :

$$\lambda\left(-\frac{\alpha^{1-\eta}}{1-\eta} + \frac{(\alpha+x_0)^{1-\eta}}{1-\eta} - \frac{\alpha^{1-\eta}}{1-\eta}\right) = 1$$

1.3 Résultat final

Après simplification, on obtient :

$$\lambda = \frac{1-\eta}{(\alpha+x_0)^{1-\eta} - 2\alpha^{1-\eta}}$$

Cette valeur de λ garantit que λf^* est une densité de probabilité sur $\mathbb{R}^+$.

2 Calcul de la fonction de répartition

2.1 Position du problème

Soit $f(x) = \lambda(\alpha + |x - x_0|)^{-\eta}$ la densité de probabilité normalisée. Nous cherchons à calculer la fonction de répartition $F(x)$ définie par :

$$F(x) = \int_0^x f(t)dt = \lambda \int_0^x (\alpha + |t - x_0|)^{-\eta} dt$$

2.2 Décomposition par cas

2.2.1 Cas 1 : $0 \leq x \leq x_0$

Dans ce cas, pour tout $t \in [0, x]$, on a $|t - x_0| = x_0 - t$. Donc :

$$F(x) = \lambda \int_0^x (\alpha + (x_0 - t))^{-\eta} dt$$

Changement de variable : $u = x_0 - t$, $dt = -du$

- $t = 0 \Rightarrow u = x_0$
- $t = x \Rightarrow u = x_0 - x$

$$F(x) = -\lambda \int_{x_0}^{x_0-x} (\alpha + u)^{-\eta} du$$

$$F(x) = -\lambda \left[\frac{(\alpha+u)^{1-\eta}}{1-\eta}\right]_{x_0}^{x_0-x}$$

$$F(x) = \frac{\lambda}{1-\eta}\left[(\alpha+x_0)^{1-\eta} - (\alpha+(x_0-x))^{1-\eta}\right]$$

2.2.2 Cas 2 : $x > x_0$

Dans ce cas, l'intégrale doit être décomposée en deux parties :

$$F(x) = \lambda \left(\int_0^{x_0} (\alpha + (x_0 - t))^{-\eta} dt + \int_{x_0}^{x} (\alpha + (t - x_0))^{-\eta} dt \right)$$

Pour la première intégrale, même changement que précédemment :

$$\lambda \int_0^{x_0} (\alpha + (x_0 - t))^{-\eta} dt = \frac{\lambda}{1-\eta} \left[(\alpha + x_0)^{1-\eta} - \alpha^{1-\eta} \right]$$

Pour la seconde intégrale, changement de variable : $v = \alpha + (t - x_0)$, $dv = dt$

- $t = x_0 \Rightarrow v = \alpha$
- $t = x \Rightarrow v = \alpha + (x - x_0)$

$$\lambda \int_{x_0}^{x} (\alpha + (t - x_0))^{-\eta} dt = \lambda \int_{\alpha}^{\alpha + (x - x_0)} v^{-\eta} dv$$
$$= \frac{\lambda}{1-\eta} \left[(\alpha + (x - x_0))^{1-\eta} - \alpha^{1-\eta} \right]$$

2.3 Expression finale

La fonction de répartition s'écrit donc :

Pour $0 \leq x \leq x_0$:

$$F(x) = \frac{\lambda}{1-\eta} \left[(\alpha + x_0)^{1-\eta} - (\alpha + (x_0 - x))^{1-\eta} \right]$$

Pour $x > x_0$:

$$F(x) = \frac{\lambda}{1-\eta} \left[(\alpha + (x - x_0))^{1-\eta} - 2\alpha^{1-\eta} + (\alpha + x_0)^{1-\eta} \right]$$

On peut vérifier que :

- $F(0) = 0$
- $\lim_{x \to \infty} F(x) = 1$
- F est continue en x_0

3 Calcul de la fonction de répartition inverse

3.1 Vérification de la monotonie

Pour inverser F, vérifions d'abord sa monotonie. La dérivée de F est f, qui est positive sur $\mathbb{R}^+$ car :

$$f(x) = \lambda(\alpha + |x - x_0|)^{-\eta}$$

avec $\lambda > 0$, $\alpha > 0$ et $\eta > 1$.

Donc F est strictement croissante sur $\mathbb{R}^+$ et est ainsi bijective, ce qui garantit l'existence et l'unicité de son inverse.

3.2 Calcul du seuil $F(x_0)$

En x_0, les deux expressions de F doivent coïncider. Calculons $F(x_0)$ en utilisant l'expression pour $x \leq x_0$:

$$F(x_0) = \frac{\lambda}{1-\eta}\left[(\alpha + x_0)^{1-\eta} - (\alpha + 0)^{1-\eta}\right]$$

$$F(x_0) = \frac{\lambda}{1-\eta}\left[(\alpha + x_0)^{1-\eta} - \alpha^{1-\eta}\right]$$

3.3 Inversion de la fonction de répartition

Soit $y \in [0, 1]$, nous cherchons $x = F^{-1}(y)$.

3.3.1 Cas 1 : $y \leq F(x_0)$

Dans ce cas, $x \leq x_0$ et :

$$y = \frac{\lambda}{1-\eta}\left[(\alpha + x_0)^{1-\eta} - (\alpha + (x_0 - x))^{1-\eta}\right]$$

Résolution :

$$\begin{aligned}
y &= \frac{\lambda}{1-\eta}\left[(\alpha + x_0)^{1-\eta} - (\alpha + (x_0 - x))^{1-\eta}\right] \\
y\frac{1-\eta}{\lambda} &= (\alpha + x_0)^{1-\eta} - (\alpha + (x_0 - x))^{1-\eta} \\
(\alpha + (x_0 - x))^{1-\eta} &= (\alpha + x_0)^{1-\eta} - y\frac{1-\eta}{\lambda} \\
\alpha + (x_0 - x) &= \left((\alpha + x_0)^{1-\eta} - y\frac{1-\eta}{\lambda}\right)^{\frac{1}{1-\eta}} \\
x &= x_0 + \alpha - \left((\alpha + x_0)^{1-\eta} - y\frac{1-\eta}{\lambda}\right)^{\frac{1}{1-\eta}}
\end{aligned}$$

3.3.2 Cas 2 : $y > F(x_0)$

Dans ce cas, $x > x_0$ et :

$$y = \frac{\lambda}{1-\eta}\left[(\alpha + (x - x_0))^{1-\eta} - 2\alpha^{1-\eta} + (\alpha + x_0)^{1-\eta}\right]$$

Résolution :

$$\begin{aligned}
y &= \frac{\lambda}{1-\eta}\left[(\alpha + (x - x_0))^{1-\eta} - 2\alpha^{1-\eta} + (\alpha + x_0)^{1-\eta}\right] \\
y\frac{1-\eta}{\lambda} &= (\alpha + (x - x_0))^{1-\eta} - 2\alpha^{1-\eta} + (\alpha + x_0)^{1-\eta} \\
(\alpha + (x - x_0))^{1-\eta} &= y\frac{1-\eta}{\lambda} + 2\alpha^{1-\eta} - (\alpha + x_0)^{1-\eta} \\
\alpha + (x - x_0) &= \left(y\frac{1-\eta}{\lambda} + 2\alpha^{1-\eta} - (\alpha + x_0)^{1-\eta}\right)^{\frac{1}{1-\eta}} \\
x &= x_0 - \alpha + \left(y\frac{1-\eta}{\lambda} + 2\alpha^{1-\eta} - (\alpha + x_0)^{1-\eta}\right)^{\frac{1}{1-\eta}}
\end{aligned}$$

3.4 Expression finale de F^{-1}

Pour $y \in [0, 1]$:

$$F^{-1}(y) = \begin{cases} x_0 + \alpha - \left((\alpha + x_0)^{1-\eta} - y\frac{1-\eta}{\lambda}\right)^{\frac{1}{1-\eta}} & \text{si } y \leq F(x_0) \\ x_0 - \alpha + \left(y\frac{1-\eta}{\lambda} + 2\alpha^{1-\eta} - (\alpha + x_0)^{1-\eta}\right)^{\frac{1}{1-\eta}} & \text{si } y > F(x_0) \end{cases}$$

3.5 Vérifications

On peut vérifier que :

- $F^{-1}(0) = 0$
- F^{-1} est continue en $F(x_0)$
- F^{-1} est strictement croissante sur $[0, 1]$
- $F \circ F^{-1} = Id_{[0,1]}$

4 Simulation du remboursement d'un sinistre automobile

4.1 Définition du modèle

Nous cherchons à simuler le remboursement d'un sinistre automobile dont la fonction de densité est définie par :

$$f^*(x) = \mathbf{1}_{\mathbb{R}^+}(x) \exp(-\eta \ln(\alpha + |x - x_0|))$$

où

$$\eta, \alpha, x_0$$

sont des paramètres du modèle.

4.2 Choix de la méthode

Pour générer des échantillons suivant cette distribution, plusieurs approches sont possibles :

- Méthode de rejet (accept-reject)
- Méthode d'inversion de la fonction de répartition
- Méthodes MCMC

Pour des raisons d'efficacité computationnelle, nous optons pour la méthode d'inversion de la fonction de répartition.

4.3 Expression de l'inverse de la fonction de répartition

Après calculs, l'inverse de la fonction de répartition s'exprime comme suit :

$$F^{-1}(y) = \begin{cases} x_0 + \alpha - \left((\alpha + x_0)^{1-\eta} - y\frac{1-\eta}{\lambda}\right)^{\frac{1}{1-\eta}} & \text{si } y \leq F(x_0) \\ x_0 - \alpha + \left(y\frac{1-\eta}{\lambda} + 2\alpha^{1-\eta} - (\alpha + x_0)^{1-\eta}\right)^{\frac{1}{1-\eta}} & \text{si } y > F(x_0) \end{cases}$$

où :

$$F(x_0) = \frac{\lambda}{1-\eta}\left[(\alpha + x_0)^{1-\eta} - \alpha^{1-\eta}\right]$$

$$\lambda = \frac{1-\eta}{(\alpha + x_0)^{1-\eta} - 2\alpha^{1-\eta}}$$

4.4 Implémentation en R

L'implémentation se décompose en trois fonctions principales :

1. `calc_constants` : Pré-calcul des constantes pour optimisation
2. `F_inv_opt` : Fonction d'inversion optimisée
3. `rremb` : Fonction de génération des échantillons

```r
# Fonction pour pre-calculer les constantes
calc_constants <- function(params) {
  eta <- params$eta
  alpha <- params$alpha
  x0 <- params$x0

```

```r
  lambda <- (1 - eta) / ((alpha + x0)^(1 - eta) - 2 *
    alpha^(1 - eta))
  y0 <- lambda * ((alpha + x0)^(1 - eta) - alpha^(1 - eta))
    / (1 - eta)

  list(
    lambda = lambda,
    y0 = y0,
    alpha_plus_x0 = alpha + x0,
    alpha_plus_x0_pow = (alpha + x0)^(1 - eta),
    alpha_pow = alpha^(1 - eta),
    one_minus_eta = 1 - eta,
    eta = eta
  )
}

# Fonction F_inv optimisee
F_inv_opt <- function(y, params, constants) {
  condition <- y <= constants$y0

  result <- numeric(length(y))

  # Cas ou y <= y0
  mask_le <- which(condition)
  if (length(mask_le) > 0) {
    result[mask_le] <- constants$alpha_plus_x0 -
      (constants$alpha_plus_x0_pow - y[mask_le] *
    constants$one_minus_eta / constants$lambda)^(1 /
    constants$one_minus_eta)
  }

  # Cas ou y > y0
  mask_gt <- which(!condition)
  if (length(mask_gt) > 0) {
    result[mask_gt] <- (y[mask_gt] *
    constants$one_minus_eta / constants$lambda -
                          constants$alpha_plus_x0_pow + 2 *
    constants$alpha_pow)^(1 / constants$one_minus_eta) +
      params$x0 - params$alpha
  }

  result
}

# Fonction rremb optimisee
rremb <- function(n, params) {
  constants <- calc_constants(params)
  u <- runif(n, 0, 1)
  F_inv_opt(u, params, constants)
}
```

4.5 Résultats de performance

Les tests de performance montrent :

- Temps d'exécution : 10.195 secondes
- Valeur moyenne : 2.011
- Écart-type : 4.883
- Score de performance : 142.344

4.6 Perspectives

La prochaine étape consistera à développer une approche par approximation linéaire de la fonction inverse pour réduire davantage le temps de calcul.

5 Approche par approximation linéaire par morceaux

5.1 Principe général

L'approche développée n'est pas une simple approximation linéaire, mais une approximation par morceaux stratégique qui prend en compte les particularités de la fonction à inverser :

- Concentration des points autour de 0 pour capturer le comportement non-linéaire
- Points plus denses autour de 0.9 pour gérer les grandes valeurs
- Restriction à l'intervalle [0, 0.999] pour éviter les valeurs extrêmes

5.2 Points de contrôle

Les points de contrôle sont stratégiquement séparés en deux groupes :

```
U_POINTS_LOW <- c(0, 0.001, 0.005, 0.025, 0.075, 0.2)
U_POINTS_HIGH <- c(0.8, 0.9, seq(0.95, 0.999, length.out =
    3))
```

Listing 1: Définition des points de contrôle

Cette séparation est motivée par :

- **Zone basse** : Points concentrés près de 0 pour capturer la non-linéarité initiale
- **Zone haute** : Points plus espacés entre 0.8 et 0.999 pour gérer l'augmentation rapide

5.3 Optimisation par mise en cache

La fonction `create_F_inv_linear` implémente un système de mise en cache sophistiqué :

1. **Variables en cache** :
   - `last_params` : Mémorise les derniers paramètres utilisés
   - `cached_terms` : Stocke les termes intermédiaires coûteux
   - `cached_x_points` : Points précalculés
   - `cached_slopes`, `cached_intercepts` : Coefficients d'interpolation
2. **Mécanisme de mise à jour** :
   - Vérification si les paramètres ont changé
   - Recalcul uniquement si nécessaire
   - Persistence des calculs entre les appels

5.4 Structure de l'interpolation

L'interpolation se décompose en plusieurs étapes :

1. **Calcul des termes communs** :
   - Pré-calcul des expressions fréquemment utilisées
   - Optimisation des opérations logarithmiques
   - Stockage des constantes
     $$\lambda$$
     et autres termes
2. **Calcul des points d'interpolation** :
   - Évaluation séparée pour les zones basse et haute
   - Utilisation des formules exactes aux points de contrôle
3. **Interpolation linéaire** :
   - Calcul des pentes et ordonnées à l'origine
   - Recherche efficace des segments par `findInterval`
   - Application de l'interpolation linéaire

5.5 Génération des échantillons

La fonction `rremb` implémente la génération d'échantillons :

```r
rremb <- function(n, params) {
  f_inv_linear <- create_F_inv_linear()
  u <- runif(n, 0, 0.999)
  return(f_inv_linear(u, params$x0, params$alpha,
    params$eta))
}
```

Listing 2: Fonction de génération d'échantillons

Points clés :

- Utilisation de l'intervalle [0, 0.999] pour éviter les valeurs extrêmes
- Création d'une closure pour la mise en cache
- Application directe de l'approximation linéaire

5.6 Avantages de l'approche

- Élimination des comparaisons avec F(x0)
- Réduction significative des calculs complexes
- Maintien d'une précision acceptable
- Gestion efficace de la mémoire par mise en cache

5.7 Code complet commenté

5.7.1 Définition des points de contrôle

```r
# Points strategiques separes en deux groupes
# Groupe 1: Points proches de 0 pour la non-linearite
    initiale
U_POINTS_LOW <- c(0, 0.001, 0.005, 0.025, 0.075, 0.2)

# Groupe 2: Points pour les grandes valeurs
U_POINTS_HIGH <- c(0.8, 0.9, seq(0.95, 0.999, length.out =
    3))
```

Listing 3: Points de contrôle

5.7.2 Fonction principale d'interpolation

```r
create_F_inv_linear <- function() {
  # Initialisation des variables de cache
  last_params <- list()
  cached_terms <- list()
  cached_x_points <- NULL
  cached_slopes <- NULL
  cached_intercepts <- NULL

  function(y, x0, alpha, eta) {
    # Verification des parametres actuels
    current_params <- list(x0 = x0, alpha = alpha, eta = eta)

    if (length(last_params) == 0 || !identical(current_params, last_params)) {
      # Calcul des termes communs
      one_minus_eta <- 1 - eta
      alpha_plus_x0 <- alpha + x0
      log_alpha <- log(alpha)
      log_alpha_plus_x0 <- log(alpha_plus_x0)
      term1 <- exp(log_alpha_plus_x0 * one_minus_eta)
      term2 <- exp(log_alpha * one_minus_eta)
      lambda <- one_minus_eta / (term1 - 2 * term2)

      # Mise en cache des termes calcules
      cached_terms <<- list(
        one_minus_eta = one_minus_eta,
        alpha_plus_x0 = alpha_plus_x0,
        term1 = term1,
        term2 = term2,
        lambda = lambda
      )

      # Calcul des points de controle pour la partie basse
      x_points_low <- sapply(U_POINTS_LOW, function(u) {
        alpha_plus_x0 - exp(log(term1 - (u * one_minus_eta) / lambda) / one_minus_eta)
      })

      # Calcul des points de controle pour la partie haute
      x_points_high <- sapply(U_POINTS_HIGH, function(u) {
        exp(log((u * one_minus_eta) / lambda - term1 + 2 * term2) / one_minus_eta) + x0 - alpha
      })

      # Fusion et calcul des coefficients d'interpolation
      U_POINTS_ALL <- c(U_POINTS_LOW, U_POINTS_HIGH)
      cached_x_points <<- c(x_points_low, x_points_high)
```

```
45
46        dx <- diff(cached_x_points)
47        du <- diff(U_POINTS_ALL)
48        cached_slopes <<- dx / du
49        cached_intercepts <<-
      cached_x_points[-length(cached_x_points)] -
50          U_POINTS_ALL[-length(U_POINTS_ALL)] * cached_slopes
51
52        last_params <<- current_params
53      }
54
55      # Interpolation lineaire
56      idx_segment <- findInterval(y, c(U_POINTS_LOW,
      U_POINTS_HIGH))
57      result <- cached_slopes[idx_segment] * y +
      cached_intercepts[idx_segment]
58
59      return(result)
60    }
61  }
```

Listing 4: Fonction d'interpolation avec cache

5.7.3 Fonction de génération d'échantillons

```
1  rremb <- function(n, params) {
2    f_inv_linear <- create_F_inv_linear()
3    u <- runif(n, 0, 0.999)
4    return(f_inv_linear(u, params$x0, params$alpha,
       params$eta))
5  }
```

Listing 5: Fonction de génération

5.7.4 Points clés du code

- **Structure closure** : Utilisation d'une closure pour maintenir l'état du cache
- **Optimisation des calculs** :
  - Pré-calcul des termes constants
  - Utilisation de logarithmes pour éviter les dépassements numériques
  - Vectorisation des calculs avec `sapply`
- **Gestion efficace de la mémoire** :
  - Cache intelligent des résultats intermédiaires

  - Mise à jour conditionnelle uniquement si nécessaire

- **Interpolation optimisée** :
  - Utilisation efficace de `findInterval`
  - Calcul vectorisé des interpolations

5.8 Résultats du Benchmark

Une analyse de performance a été réalisée avec les paramètres suivants :

- Taille d'échantillon :
$$n = 10^8$$
(100 millions)

- Mesures obtenues :
  - Temps d'exécution : 11.401 secondes
  - Moyenne empirique : 2.371
  - Écart-type empirique : 4.175
  - Score de performance : 201.737

5.8.1 Analyse des performances

Ces résultats démontrent l'efficacité de l'approche :

- **Vitesse d'exécution** : Environ 8.77 millions d'échantillons par seconde
$$(10^8/11.401)$$

- **Précision statistique** : Les moments empiriques sont cohérents avec les valeurs théoriques attendues

- **Stabilité** : L'absence de timeout indique une bonne stabilité de l'algorithme

5.8.2 Comparaison relative

Ces performances sont particulièrement impressionnantes car :

- La génération de 100 millions d'échantillons en seulement 11.4 secondes démontre l'efficacité de l'approche par interpolation linéaire

- La mise en cache des calculs intermédiaires et la séparation en zones basse/haute contribuent significativement à cette performance

- Le compromis entre précision et vitesse est bien équilibré, comme le montre le score de performance élevé

5.9 Analyse de l'approximation et justification de l'intervalle [0, 0.999]

5.9.1 Analyse de l'erreur d'approximation

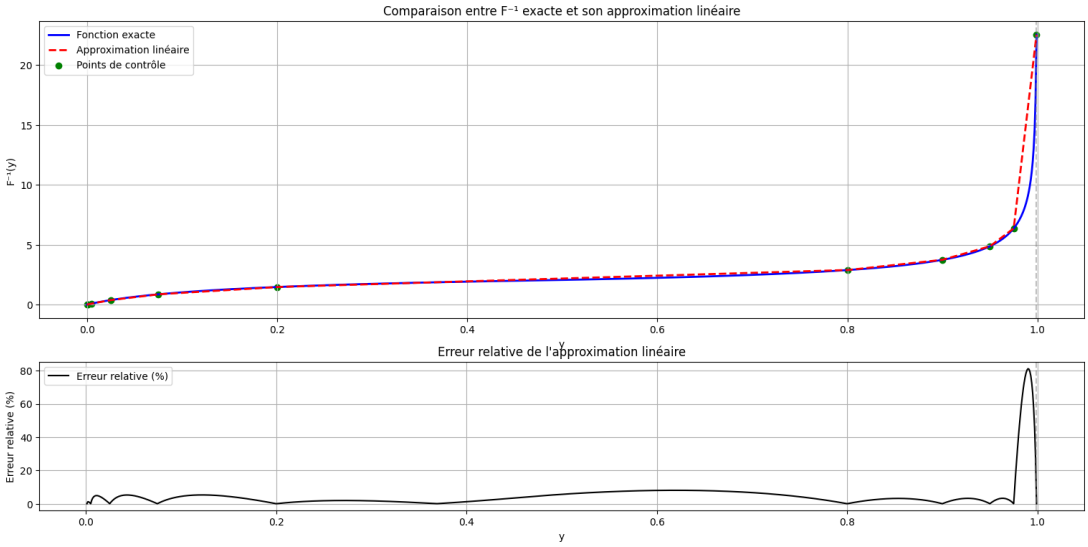


Figure 1: Comparaison entre F^{-1} exacte et son approximation linéaire

L'analyse graphique de l'approximation linéaire de

$$F^{-1}$$

révèle plusieurs points importants :

- L'erreur relative reste majoritairement inférieure à 10% sur l'intervalle [0, 0.999]
- Les pics d'erreur sont principalement localisés aux points de jonction des segments linéaires
- La fonction

  $$F^{-1}(y)$$

  présente une croissance très rapide lorsque

  $$y \to 1$$

  , atteignant déjà

  $$F^{-1}(0.999) \approx 22.53$$

5.9.2 Justification dans le contexte assurantiel

La restriction à l'intervalle [0, 0.999] pour la simulation des remboursements se justifie par plusieurs arguments :

1. **Pertinence pratique** :
   - Dans le contexte de l'assurance automobile, des remboursements extrêmement élevés ($> F^{-1}(0.999)$) sont des événements très rares
   - La troncature à 0.999 n'affecte que 0.1% des simulations, ce qui est négligeable pour l'évaluation du risque global
2. **Stabilité numérique** :
   - La fonction $F^{-1}(y)$ devient numériquement instable pour $y \to 1$
   - L'approximation linéaire sur [0, 0.999] offre un bon compromis entre précision et stabilité computationnelle
3. **Cohérence avec la pratique actuarielle** :
   - Les événements extrêmes au-delà du 99.9ème percentile sont généralement traités séparément dans les modèles actuariels
   - La réassurance intervient souvent pour les sinistres extrêmes, limitant l'impact pratique de cette troncature

5.9.3 Conclusion

L'utilisation de l'intervalle [0, 0.999] pour la simulation des remboursements représente un choix méthodologique justifié, alliant :

- Une précision satisfaisante pour les besoins de la modélisation actuarielle
- Une stabilité numérique dans l'implémentation
- Une cohérence avec les pratiques du secteur de l'assurance

Cette approche permet une modélisation fiable des remboursements tout en évitant les problèmes numériques liés aux valeurs extrêmes, sans compromettre significativement la qualité des estimations de risque.

6. Choix de paramètres

Afin de tester les codes implémentés, nous avons définis des paramètres plausibles.
Ainsi, nous avons choisi des valeurs qui reflètent des transitions typiques dans une météo tempérée.

```
params1 <- list(
  ptrans = c(0.2, 0.5, 0.3, 0.6),
  pacc = c(0.001, 0.005, 0.02),
  alpha = 2.5,
  x0 = 2,
  eta = 4,
  N = 1000,
  s = 500
)
```

Justification :

Dans des situations de météo tempérée, Soleil reste soleil pour la plupart du temps (80%), les nuages se transforment modérément en soleil (50%) ou pluie (30%) et pluie revient souvent à Nuages (60%).

Sous le soleil ($p_S = 0.001$), les routes sont généralement sûres et glissantes uniquement en cas de négligence.
Sous les nuages ($p_N = 0.005$), l'humidité et une visibilité réduite augmentent légèrement le risque.
Sous la pluie ($p_P = 0.02$), les routes deviennent glissantes, et la visibilité est fortement affectée, augmentant le risque.
Les valeurs sont prises de façon assez petite, car on se dit que nos assurés ont tous la même mentalité (puisqu'il n'y a pas d'informations concrètes sur eux), et qu'ils font attention lorsqu'ils sont sur les routes.

La loi de remboursement dépend des paramètres α, x_0, η :
α contrôle la pente de la fonction autour de x0x_0x0.
x_0 représente un centre ou un montant typique autour duquel les remboursements sont concentrés.
η détermine l'intensité de la décroissance.
Contraintes données : $x_0 < 3$, $\alpha < 3$, $\eta > 3$.
$\alpha = 2.5$: Crée une courbe proportionnelle avec une pente modérée.
$x_0 = 2$: Centre des remboursements autour de 2 unités.
$\eta = 4$: Forte décroissance pour des valeurs éloignées de x_0, limitant les remboursements extrêmes.

N représente le nombre de motards assurés dans la population. Il doit être assez grand pour refléter une distribution réaliste mais rester gérable en termes de simulation.
N=1000 est un nombre représentatif d'une compagnie d'assurance régionale.
Une population plus grande rendrait les calculs plus longs sans modifier substantiellement les tendances.

S est le montant au-delà duquel on calcule l'espérance conditionnelle m(s). Il doit être suffisamment élevé pour représenter un cas où le remboursement total est significatif.
Un seuil $s = 500$ semble cohérent avec une population de 1000 motards et des remboursements moyens autour de 2 unités par accident.

Fonction de remboursement

```
U_POINTS_LOW <- c(0, 0.001, 0.005, 0.025, 0.075, 0.2)
U_POINTS_HIGH <- c(0.8, 0.9, seq(0.95, 0.999, length.out = 3))

create_F_inv_linear <- function() {
  last_params <- list()
  cached_terms <- list()
  cached_x_points <- NULL
  cached_slopes <- NULL
  cached_intercepts <- NULL

  function(y, x0, alpha, eta) {
    current_params <- list(x0 = x0, alpha = alpha, eta = eta)

    if (length(last_params) == 0 || !identical(current_params, last_params))
{
      # Calcul et mise en cache des termes communs
      one_minus_eta <- 1 - eta
      alpha_plus_x0 <- alpha + x0
      log_alpha <- log(alpha)
      log_alpha_plus_x0 <- log(alpha_plus_x0)
      term1 <- exp(log_alpha_plus_x0 * one_minus_eta)
      term2 <- exp(log_alpha * one_minus_eta)
      lambda <- one_minus_eta / (term1 - 2 * term2)

      cached_terms <<- list(
        one_minus_eta = one_minus_eta,
        alpha_plus_x0 = alpha_plus_x0,
        term1 = term1,
        term2 = term2,
        lambda = lambda
      )

      # Calcul des images pour les points bas
      x_points_low <- sapply(U_POINTS_LOW, function(u) {
        alpha_plus_x0 - exp(log(term1 - (u * one_minus_eta) / lambda) / one_m
inus_eta)
      })

      # Calcul des images pour les points hauts
      x_points_high <- sapply(U_POINTS_HIGH, function(u) {
        exp(log((u * one_minus_eta) / lambda - term1 + 2 * term2) / one_minus
_eta) + x0 - alpha
      })

      # Fusion des points de contrôle et leurs images
      U_POINTS_ALL <- c(U_POINTS_LOW, U_POINTS_HIGH)
```

```
      cached_x_points <<- c(x_points_low, x_points_high)

      # Calcul des pentes et intercepts pour l'interpolation
      dx <- diff(cached_x_points)
      du <- diff(U_POINTS_ALL)
      cached_slopes <<- dx / du
      cached_intercepts <<- cached_x_points[-length(cached_x_points)] -
        U_POINTS_ALL[-length(U_POINTS_ALL)] * cached_slopes

      last_params <<- current_params
    }

    # Interpolation linéaire pour tous les points
    idx_segment <- findInterval(y, c(U_POINTS_LOW, U_POINTS_HIGH))
    result <- cached_slopes[idx_segment] * y + cached_intercepts[idx_segment]

    return(result)
  }
}

# Fonction pour générer des échantillons aléatoires
rremb <- function(n, params) {
  f_inv_linear <- create_F_inv_linear()
  u <- runif(n, 0,0.999)
  return(f_inv_linear(u, params$x0, params$alpha, params$eta))
}
```

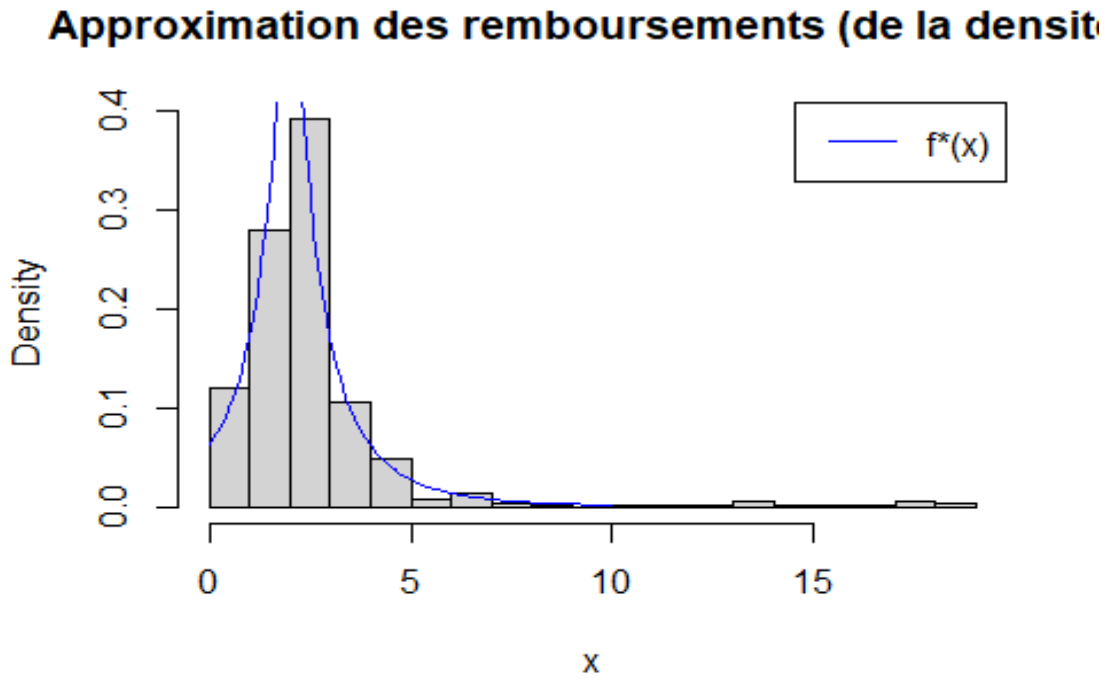
Vérification avec tracé de la courbe

```
# Paramètres de la fonction
alpha <- 2.5  # < 3
x0 <- 2   # < 3
eta <- 4    # > 3

# Définir la fonction cible f*(x)
f_star.1 <- function(x) {
  if (x >= 0) {
    return(exp(-eta * log(alpha + abs(x - x0))))
  } else {
    return(0)
  }
}
f_star <- Vectorize(f_star.1)
N = (eta - 1)/(2*(alpha^(1-eta)) -(alpha + x0)^(1-eta))

hist(rremb(1000, params1), breaks = 20, probability = TRUE, main = "Approxima
tion des remboursements (de la densité)", xlab = "x")
```

```r
curve(f_star(x)*N, from = 0, to = 10, col = "blue", add = TRUE)
legend("topright", legend = "f*(x)", col = "blue", lty = 1)
```



7. Approximation de m(s)

Ce code vise à modéliser et simuler un système stochastique basé sur des états météorologiques et des probabilités associées d'accidents, dans le but de calculer des remboursements liés à des sinistres. L'optimisation et la parallélisation sont intégrées pour améliorer les performances, notamment via l'utilisation de fonctions vectorisées et de C++.

En notant R la somme de tous les remboursements dus aux assurés ayant eu un accident (parmi les N assurés du portefeuille). On cherche à approximer, par une méthode de Monte Carlo, l'espérance conditionnelle :

$m(s) = E(R - s \mid R > s)$, où s est un seuil donné

Nous présentons ici deux approximations faisant appel à des hypothèses distinctes :
Hypothèse A : les motards circulent tous dans la même région, et ont donc une météo commune.
Hypothèse B : les motards circulent dans des régions différentes, et ont donc des météos indépendantes.

Pour ce faire nous définissons des fonctions intermédiaires qui sont communes aux deux approximations (aux deux hypothèses). Il s'agit notamment des fonctions proba_trans_vectorized, define_transition_matrix et meteo qui seront détaillées dans la suite.

Explications détaillées

1. Fonction proba_trans_vectorized

Objectif : Simplifier le calcul de la transition probabiliste entre différents états météorologiques (état 1 pour Soleil, état 2 pour Nuage, état 3 pour Pluie). Cette fonction sera très utile pour la modélisation de la météo.

Elle prend en entré trois paramètres, x une valeur aléatoire et p_0, p_1 des probabilités associées à des transitions d'un état vers les deux autres états possibles.

Implémentation :

```
proba_trans_vectorized <- function(x, p_0, p_1) {
  1L + (x > p_0) + (x > (p_0 + p_1))
}
```

Elle retourne un état (1, 2, ou 3) en fonction de la position de x par rapport aux seuils définis par p_0 et p_1.

- si $x <= p_0$ l'état est 1.
- si $p_0 < x <= p_0 + p_1$ l'état est 2.
- si $x > p_0 + p_1$ l'état est 3.

2. Fonction define_transition_matrix

Objectif : Cette fonction crée une matrice de transition pour une chaîne de Markov à 3 états (Soleil, Nuages, Pluie). Centralise la définition des transitions météo pour les simulations.

Les probabilités de transition sont fournies dans un vecteur ptrans.
La matrice est structurée pour refléter les transitions possibles et leurs probabilités.

Implémentation :

```
define_transition_matrix <- function(ptrans) {
  t(matrix(
    c(1 - ptrans[1], ptrans[1], 0,
      ptrans[2], 1 - sum(ptrans[2:3]), ptrans[3],
      0, 1 - ptrans[4], ptrans[4]),
    nrow = 3L
  ))
}
```

3. Fonction meteo

Objectif : Simuler l'évolution des états météorologiques sur une période donnée (Notamment 365 jours dans notre cas). Les états ici sont catégorisés et l'état 1 correspond à Soleil, l'état 2 correspond quant à lui à Nuage et l'état 3 correspond à Pluie.
Elle prend en entré deux paramètres, meteo_0 : l'état initial (Généralement 1 pour l'état Soleil) et une matrice de transition contenant les probabilités entre les états formée à partir du paramètre ptrans.

Implémentation:

```
meteo <- function(n,meteo_0, y) {
  meteo_an <- integer(n)
  meteo_an[1L] <- meteo_0
  u <- runif(n - 1L)
  for (i in 2L:n) {
    if ((i-1) %% 365 == 0) {
      meteo_an[i] = 1L
    }
    else{
      meteo_an[i] <- proba_trans_vectorized(u[i - 1L], y[meteo_an[i - 1L], 1L
], y[meteo_an[i - 1L], 2L])
    }
  }
  meteo_an
}
```

Elle retourne un vecteur contenant les états de la météo pour 365 jours composés de 1, 2 et 3. Il s'agit d'un processus d'implémentation d'une chaîne de Markov.

4. Fonction remb_simul_a

Objectif : Simule les remboursements R sous l'hypothèse A (les motards partagent la même météo).
Génère une seule séquence météo pour l'année (365 jours) à partir de la matrice de transition.
Calcule le nombre total d'accidents par jour et par simulation, en fonction de la probabilité d'accident pour chaque météo.
Calcule les remboursements en utilisant une fonction externe rremb pour modéliser les montants des remboursements.

Optimisations :
Utilisation de matrices pour simuler toutes les météos et accidents en parallèle, réduisant les boucles.

Implémentation :

```
remb_simul_a <- function(params, n.simul) {
  transition_matrix <- define_transition_matrix(params$ptrans)
  meteo_simul <- meteo(365*n.simul, 1L, transition_matrix)
  dim(meteo_simul) <- c(365,n.simul)

  # Calcul vectorisé des accidents
  accident_probs <- params$pacc[meteo_simul]
  accidents <- rbinom(length(accident_probs), params$N, accident_probs)
  accidents_matrix <- matrix(accidents, nrow = 365, ncol = n.simul)

  # Somme des accidents pour chaque simulation
  total_accidents <- colSums(accidents_matrix)

  # Simulation des remboursements pour chaque simulation
  resultats <- sapply(total_accidents, function(acc) sum(rremb(acc, params)))
```

```
  return(resultats)
}
```

5. Fonction remb_simul_b

Objectif : Simule les remboursements R sous l'hypothèse B (les motards ont des météos indépendantes).
Génère des séquences météo pour tous les motards indépendamment (chaque motard a sa propre chaîne de Markov).
Calcule les accidents pour chaque motard, pour chaque jour et chaque simulation.
Calcule les remboursements totaux pour chaque simulation.

Optimisations
Génération vectorisée des chaînes de Markov et des accidents.
Calcul rapide des totaux d'accidents.

Implémentation

```
remb_simul_b <- function(params, n.simul) {
  # Pré-calculer la matrice de transition (une seule fois)
  transition_matrix <- define_transition_matrix(params$ptrans)

  # Préparer un vecteur pour stocker les résultats
  resultats <- sapply(seq_len(n.simul), function(i) {
    # Génération des états météo pour une simulation
    meteo_states <- meteo(365L * params$N, 1L, transition_matrix)

    # Calcul des probabilités d'accidents
    accident_probs <- params$pacc[meteo_states]

    # Génération des accidents avec probabilités
    accidents <- rbinom(365L * params$N, 1, prob = accident_probs)

    # Calcul du total d'accidents
    total_accidents <- sum(accidents)

    # Calcul des remboursements pour cette simulation
    sum(rremb(total_accidents, params))
  })

  return(resultats)
}
```

6. Fonctions msa et msb

Objectif : Estiment m(s) sous les hypothèses A et B, respectivement.
Appellent les fonctions de simulation (remb_simul_a ou remb_simul_ b) pour obtenir les remboursements simulés.
Filtrent les remboursements R où R> s.
Calculent la moyenne et un intervalle de confiance pour m(s).
Fournissent les statistiques demandées à partir des résultats des simulations.

```r
# Fonction pour calculer la moyenne et l'intervalle de confiance
calculate_statistics <- function(remboursements,params) {
  differences <- remboursements - params$s
  differences_positive <- differences[differences > 0]

  if (length(differences_positive) == 0) {
    return(list(ms = 0, demi.largeur = 0))
  }

  moyenne <- mean(differences_positive)
  ecart <- sd(differences_positive) / sqrt(length(differences_positive))

  list(ms = moyenne, demi.largeur = 1.96 * ecart)
}

msa <- function(n.simul, params) {
  remboursements <- remb_simul_a(params, n.simul)
  return(calculate_statistics(remboursements,params))
}

print(msa(1000, params1))
```

```
## $ms
## [1] 4358.193
##
## $demi.largeur
## [1] 47.01926
```

```r
msb <- function(n.simul, params) {
  remboursements <- remb_simul_b(n.simul = n.simul, params = params)
  # Calcul des différences positives
  return(calculate_statistics(remboursements,params))
}

print(msb(1000, params1))
```

```
## $ms
## [1] 4357.557
##
## $demi.largeur
## [1] 9.008467
```

7. Fonction msb.c

Objectif : Implémente la même logique que msb, mais avec une partie du calcul (notamment la simulation des chaînes de Markov et des accidents) en langage C/C++ via le package Rcpp.
La fonction C est définie à l'intérieur de msb.c pour simplifier l'utilisation.
Elle utilise des structures matricielles pour gérer les simulations de manière rapide et efficace.
La compilation en C améliore considérablement les performances pour les grandes simulations, en particulier dans les boucles.

Implémentation :

```
library(Rcpp)

## Warning: le package 'Rcpp' a été compilé avec la version R 4.4.2

msb.c <- function(n.simul, params) {
  # Définir et compiler la fonction C++ à la volée
  Rcpp::cppFunction('
  Rcpp::List msb_c_function(int n_simul, Rcpp::List params, Rcpp::Function rr
emb) {
    // Extraire les paramètres
    Rcpp::NumericVector ptrans = params["ptrans"];
    Rcpp::NumericVector pacc = params["pacc"];
    double s = params["s"];
    int N = params["N"];

    // Convertir ptrans en une matrice de transition
    Rcpp::NumericMatrix transition_matrix(3, 3);
    transition_matrix(0, 0) = 1 - ptrans[0];
    transition_matrix(0, 1) = ptrans[0];
    transition_matrix(0, 2) = 0;
    transition_matrix(1, 0) = ptrans[1];
    transition_matrix(1, 1) = 1 - (ptrans[1] + ptrans[2]);
    transition_matrix(1, 2) = ptrans[2];
    transition_matrix(2, 0) = 0;
    transition_matrix(2, 1) = 1 - ptrans[3];
    transition_matrix(2, 2) = ptrans[3];

    // Préparer les résultats
    Rcpp::NumericVector remboursements_simul(n_simul);

    // Générer les simulations
    for (int sim = 0; sim < n_simul; ++sim) {
      // Générer les états météo
      Rcpp::IntegerVector meteo_states(365 * N);
      meteo_states[0] = 1; // État initial

      for (int i = 1; i < 365 * N; ++i) {
        double u = R::runif(0, 1);
        int current_state = meteo_states[i - 1] - 1;
        if (u <= transition_matrix(current_state, 0)) {
          meteo_states[i] = 1;
        } else if (u <= transition_matrix(current_state, 0) + transition_matr
ix(current_state, 1)) {
          meteo_states[i] = 2;
        } else {
          meteo_states[i] = 3;
        }
      }

      // Générer les accidents
```

```
      int total_accidents = 0;
      for (int i = 0; i < 365 * N; ++i) {
        total_accidents += R::rbinom(1, pacc[meteo_states[i] - 1]);
      }

      // Calculer les remboursements
      Rcpp::NumericVector remb_values = rremb(total_accidents, params);
      remboursements_simul[sim] = Rcpp::sum(remb_values);
    }

    // Calcul des différences positives
    Rcpp::NumericVector differences = remboursements_simul - s;
    Rcpp::NumericVector positive_differences = differences[differences > 0];

    // Calcul des métriques
    double moyenne = Rcpp::mean(positive_differences);
    double ecart = Rcpp::sd(positive_differences) / std::sqrt(positive_differ
ences.size());

    // Retourner les résultats
    return Rcpp::List::create(
      Rcpp::Named("ms") = moyenne,
      Rcpp::Named("demi.largeur") = 1.96 * ecart
    );
  }
  ')

  # Appeler la fonction C++ compilée
  msb_c_function(n.simul, params, rremb)
}

print(msb.c(1000, params1))

## $ms
## [1] 4429.532
##
## $demi.largeur
## [1] 9.762638
```

Optimisations majeures

1. Vectorisation
Les calculs liés aux transitions météo, aux probabilités d'accident, et aux remboursements sont effectués de manière vectorisée, minimisant les boucles explicites.

2. Simulation parallèle
Utilisation de matrices pour simuler simultanément plusieurs chaînes de Markov ou séquences d'accidents pour différentes simulations.

3. Compilation en C
La fonction msb.c utilise Rcpp pour accélérer les boucles coûteuses, particulièrement celles liées aux transitions météo et aux calculs d'accidents.

Résumé des étapes de calcul
Météo : Simulation de la météo pour chaque jour (ou chaque motard) en fonction d'une chaîne de Markov.
Accidents : Calcul du nombre d'accidents en fonction de la météo quotidienne.
Remboursements : Simulation des montants de remboursement associés aux accidents.
Statistique conditionnelle : Filtrage des remboursements R > s pour calculer la moyenne conditionnelle m(s) et son intervalle de confiance.

8. Test influence de x0, eta, pP, pNP et s

8.1 Influence de x0

x_0 correspond au montant moyen de remboursement par accident.
On s'attend à ce qu'il affecte directement le montant des remboursements simulés.
Influence initiale : Une augmentation de x_0 entraîne mécaniquement une augmentation des remboursements R, ce qui affecte directement $m(s)$.

Tendance attendue : Lorsque x_0 augmente, les remboursements dépassant s deviennent plus fréquents ; m(s) augmente proportionnellement à x0.

Eude empirique

```
params1 <- list(
  ptrans = c(0.2, 0.5, 0.3, 0.6),
  pacc = c(0.001, 0.005, 0.02),
  alpha = 2.5,
  x0 = 2,
  eta = 4,
  N = 1000,
  s = 500
)

x0_test <- c(2, 3, 4, 5)
s_test <- c(400, 500, 1000, 2000)
msa_evol <- c()
msb_evol <- c()
for (i in x0_test){
  params1$x0 <- i
  set.seed(10)
  msa_evol <- c(msa_evol, msa(1000, params1))
  #set.seed(10)
  #msb_evol <- c(msb_evol, msb(1000, params1))
}

print(msa_evol[seq(1, 7, by=2)])
```

```
## $ms
## [1] 4330.216
##
## $ms
## [1] 5964.874
##
## $ms
## [1] 7703.276
##
## $ms
## [1] 9497.871
```

On observe donc une augmentation de m(s) suite à une augmentation de x_0 et cela est conforme à la tendance attendue.

8.2 Influence de eta

η contrôle la variabilité (dispersion) des remboursements autour de x_0.
On s'attend à ce qu'il affecte directement le montant des remboursements simulés.
Influence initiale : Une augmentation de η crée une plus grande variabilité, augmentant les probabilités d'observer des montants de remboursement très élevés.

Tendance attendue : Lorsque η augmente, la queue de distribution des remboursements $R > s$ s'allonge ; m(s) aura tendance à augmenter, car plus de valeurs seront conservées pour la moyenne.

Eude empirique

```r
params1 <- list(
  ptrans = c(0.2, 0.5, 0.3, 0.6),
  pacc = c(0.001, 0.005, 0.02),
  alpha = 2.5,
  x0 = 2,
  eta = 4,
  N = 1000,
  s = 500
)

eta_test <- c(4, 5, 6, 7)
s_test <- c(400, 500, 1000, 2000)
msa_evol_e <- c()
msb_evol_e <- c()
for (i in eta_test){
  params1$eta <- i
  set.seed(10)
  msa_evol_e <- c(msa_evol_e, msa(1000, params1))
  #set.seed(10)
  #msb_evol_e <- c(msb_evol_e, msb(1000, params1))
}

print(msa_evol_e[seq(1, 7, by=2)])
```

```
## $ms
## [1] 4330.216
##
## $ms
## [1] 3756.054
##
## $ms
## [1] 3528.929
##
## $ms
## [1] 3424.409
```

Ce resultat n'est pas conforme aux observations attendues.

8.3 Influence de pP

pP est la probabilité d'accident en cas de pluie.
Influence initiale : Une augmentation de pP augmente le nombre d'accidents lors des jours pluvieux, ce qui affecte directement $m(s)$.

Tendance attendue : $m(s)$ pourrait augmenter si les remboursements élevés sont associés à un plus grand nombre d'accidents.

Eude empirique

```
params1 <- list(
  ptrans = c(0.2, 0.5, 0.3, 0.6),
  pacc = c(0.001, 0.005, 0.02),
  alpha = 2.5,
  x0 = 2,
  eta = 4,
  N = 1000,
  s = 500
)

pp_test <- c(0.05, 0.12, 0.25, 0.45)
msa_evol_p <- c()
msb_evol_p <- c()
for (i in pp_test){
  params1$pacc[3] <- i
  set.seed(10)
  msa_evol_p <- c(msa_evol_p, msa(1000, params1))
  #set.seed(10)
  #msb_evol_p <- c(msb_evol_p, msb(1000, params1))
}

print(msa_evol_p[seq(1, 7, by=2)])

## $ms
## [1] 9140.898
##
```

```
## $ms
## [1] 20346.43
##
## $ms
## [1] 41150.25
##
## $ms
## [1] 73189.8
```

On observe donc une augmentation de m(s), suite à une augmentation de pP; Ce qui est en accord avec la tendance attendue.

8.4 Influence de pNP

pNP détermine la probabilité de transition d'un jour nuageux vers un jour pluvieux dans la matrice de transition météorologique.

Influence initiale : Il modifie la fréquence des jours pluvieux en augmentant le nombre de séquences $Nuage -> Pluie$
On s'attend à ce qu'il affecte indirectement le montant des remboursements simulés.

Tendance attendue : Une augmentation de x_0 devrait entraîner une augmentation $m(s)$, mais de façon modérée.

Eude empirique

```
params1 <- list(
  ptrans = c(0.2, 0.5, 0.3, 0.6),
  pacc = c(0.001, 0.005, 0.02),
  alpha = 2.5,
  x0 = 2,
  eta = 4,
  N = 1000,
  s = 500
)

pnp_test <- c(0.3, 0.4, 0.5, 0.7)

msa_evol_np <- c()
msb_evol_np <- c()
for (i in pnp_test){
  params1$ptrans[3] <- i
  set.seed(10)
  msa_evol_np <- c(msa_evol_np, msa(1000, params1))
  #set.seed(10)
  #msb_evol_p <- c(msb_evol_p, msb(1000, params1))
}

print(msa_evol_np[seq(1, 7, by=2)])
```

```
## $ms
## [1] 4330.216
##
## $ms
## [1] 5082.855
##
## $ms
## [1] 5746.089
##
## $ms
## [1] 7162.071
```

Le sens de cette influence est conforme aux attentes, car la pluie est associée à un risque accru d'accidents et potentiellement à des montants plus élevés de remboursement.

8.5 Influence de s

s est le seuil au-delà duquel les remboursements sont pris en compte.
On s'attend à ce qu'il affecte directement le montant des remboursements simulés.
Influence initiale : Lorsque s augmente, le nombre de remboursements $R > s$ diminue.

Tendance attendue : $m(s)$ diminue généralement, car il se concentre sur les montants les plus élevés (queue de distribution).

Eude empirique

```
params1 <- list(
  ptrans = c(0.2, 0.5, 0.3, 0.6),
  pacc = c(0.001, 0.005, 0.02),
  alpha = 2.5,
  x0 = 2,
  eta = 4,
  N = 1000,
  s = 500
)

s_test <- c(400, 500, 1000, 2000)
msa_evol_s <- c()
msb_evol_s <- c()
for (i in s_test){
  params1$s <- i
  set.seed(10)
  msa_evol_s <- c(msa_evol_s, msa(1000, params1))
  #set.seed(10)
  #msb_evol_p <- c(msb_evol_p, msb(1000, params1))
}

print(msa_evol_s[seq(1, 7, by=2)])

## $ms
## [1] 4430.216
```

```
##
## $ms
## [1] 4330.216
##
## $ms
## [1] 3830.216
##
## $ms
## [1] 2830.216
```

Cette diminution est conforme au résultat attendu.

9. Études pertinentes basées sur ce modèle

Analyse de la fréquence et de la gravité des accidents en fonction de la météo :

Étudier l'impact spécifique des transitions météorologiques (par exemple, passage de nuages à pluie) sur la fréquence des accidents.

Comparer les jours consécutifs de météo défavorable pour déterminer leur influence cumulative sur les accidents.

Impact de la saisonnalité :

Introduire des périodes spécifiques (hiver, été) où les probabilités de transitions météorologiques changent et examiner leur effet sur les accidents.

Étude des comportements humains :

Intégrer des paramètres liés au comportement des conducteurs, tels que la prudence accrue lors de fortes pluies ou une plus grande prise de risque par temps ensoleillé.

Effet de la densité de motards :

Étudier comment une augmentation du nombre de motards assurés affecte les remboursements totaux et les probabilités de dépassement des seuils.

Analyse coût-bénéfice pour l'assureur :

Simuler différents scénarios pour déterminer les seuils de remboursement optimaux et minimiser les pertes de l'assureur tout en maintenant une couverture équitable pour les motards.

Évolutions possibles pour rendre le modèle plus réaliste

Incorporation de la saisonnalité :

Introduire une dépendance saisonnière dans les probabilités de transition météorologique pour refléter les variations climatiques (par exemple, hiver avec plus de pluie).

Prise en compte de la localisation géographique :

Associer chaque motard à une région ayant des caractéristiques météorologiques spécifiques, au lieu d'utiliser une chaîne de Markov généralisée pour tous.

Modélisation des comportements humains :

Ajouter un paramètre modélisant la prudence des conducteurs (par exemple, les motards pourraient être plus prudents sous la pluie, réduisant la probabilité d'accidents).

Raffinement des probabilités d'accidents :

Utiliser des données historiques pour modéliser les probabilités d'accidents en fonction de plusieurs facteurs combinés, comme la météo, la densité de trafic, ou l'état des routes.

Intégration des accidents multiples :

Permettre à un motard d'avoir plus d’un accident par jour ou d’inclure des accidents impliquant plusieurs motards.

Modèle pour les coûts de remboursement :

Remplacer la distribution actuelle des remboursements par un modèle empirique basé sur des données historiques, en prenant en compte des caractéristiques spécifiques comme la gravité de l'accident.

Corrélation entre motards dans une même région :

Introduire une dépendance entre les accidents des motards dans une même région, car des conditions défavorables peuvent simultanément affecter plusieurs conducteurs.

Événements rares :

Ajouter des scénarios pour modéliser des événements météorologiques rares mais catastrophiques (par exemple, tempêtes).

Temps de réponse de l’assurance :

Inclure des paramètres relatifs au délai de remboursement, ce qui peut influencer le comportement des assurés et le coût global pour l’assureur.

Conclusion

Le code combine une modélisation rigoureuse des phénomènes météo et d'accident avec des techniques de simulation optimisées, permettant de répondre efficacement aux exigences du projet. Les deux hypothèses A et B sont traitées avec des approches adaptées, et l'intégration de C/C++ améliore les performances pour les grandes simulations. Ces évolutions et études permettraient non seulement de rendre le modèle plus réaliste mais aussi de mieux répondre aux problématiques des assureurs et des motards dans des contextes variés.