Techniques de Simulations

PRISCILLA, GOGUY
priscilla.goguy@etu.univ-lyon1.fr

MAHLÎ, REINETTE
mahli.reinette@etu.univ-lyon1.fr

13/10/2025

M1 Actuariat
ISFA

Table des matières

Première partie

Préambule

0.1 Avant-propos

Toutes les ressources utilisées seront présentes en annexes et sur le ***repository*** github ci-dessus. Dans une optique de rigueur absolue, nous tenterons de commenter chaque partie de notre code et essaierons de justifier nos méthodes de simulation.

0.2 Notations

* $\mathcal{P}(\mathbf{X})$ **:** *Soit Ω, un ensemble quelconque.*
  - *On note $\mathcal{P}(\Omega)$, l'ensemble des parties de Ω.*
* **σ-Algèbre de Ω :** *Soit Ω, un ensemble quelconque. $\mathcal{A} \subseteq \mathcal{P}(\Omega)$ est dite σ-Algèbre de Ω si :*
  - $\Omega \in \mathcal{A}$
  - $\forall A \in \mathcal{A}$, $\bar{A} \in \mathcal{A}$
  - $\forall (A_n)_{n\in\mathbb{N}} \in \mathcal{A}^{\mathbb{N}}$, $\bigcup_{n\in\mathbb{N}} A_n \in \mathcal{A}$
    - *On notera la « tribu borélienne » : $\mathbb{B}(\mathbb{R}^n)$ et λ : « mesure de Lebesgue ».*
* **μ Mesure de probabilité sur $(\Omega, \mathcal{A})$:** *Soit $(\Omega, \mathcal{A})$, un couple dit « espace probabilisable »*
  - $\mu : \mathcal{A} \longrightarrow [0, 1]$
  - $\mu(\emptyset) = 0$
  - $\forall (A_n)_{n\in\mathbb{N}} \in \mathcal{A}^{\mathbb{N}}$, *une famille disjointe,* $\mu(\bigsqcup_{n\in\mathbb{N}} A_n) = \sum_{n\in\mathbb{N}} \mu(A_n)$
    - *On notera la « probabilité historique » :* $\mathbb{P}$.
    - *On notera $(\Omega, \mathcal{A}, \mu)$: « espace probabilisé ».*
* **$\mathbf{X}$, une variable aléatoire sur $(\Omega, \mathcal{A}, \mathbb{P})$:** *Soit $(\Omega, \mathcal{A}, \mathbb{P})$: « espace probabilisé »*
  - $(\boldsymbol{E}, \mathcal{E})$ *: « espace mesurable »*
  - $\boldsymbol{X}$ **:** $(\Omega, \mathcal{A}, \mathbb{P}) \longrightarrow (\boldsymbol{E}, \mathcal{E})$
    - $\forall B \in \mathcal{E}$, $\mathbb{P}_X(B) = \mathbb{P}(\{\omega \in \Omega | X(\omega) \in \mathcal{B}\})$
* **$\mathbf{X}$, une variable aléatoire continue sur $(\Omega, \mathcal{A}, \mathbb{P})$:** *Soit $(\Omega, \mathcal{A}, \mathbb{P})$: « espace probabilisé »*
  - $(\mathbb{R}^n, \mathbb{B}(\mathbb{R}^n), \lambda)$ *: « espace mesuré »*
  - $\boldsymbol{X}$ **:** $(\Omega, \mathcal{A}, \mathbb{P}) \longrightarrow (\mathbb{R}^n, \mathbb{B}(\mathbb{R}^n), \lambda)$
  - $\mathbb{P}_X$ « λ[1]
    - $\exists! \rho : (\mathbb{R}^n, \mathbb{B}(\mathbb{R}^n), \lambda) \longrightarrow \mathbb{R}^{+}$[2], *mesurable, tel que :*

$$\mathbb{P}_X(A) = \int_A \rho(x)\, dx$$

1. $\forall A \in \mathbb{B}(\mathbb{R}^n)$, tel que $\lambda(A) = 0 \Longrightarrow \mathbb{P}_X(A) = 0$.
2. Dite : *« densité de $\boldsymbol{X}$ ».*

* **Espérance E[X], de la variable aléatoire X :** *Soit* $\boldsymbol{X}$*, une variable aléatoire sur* $(\Omega, \mathcal{A}, \mathbb{P})$
  — *On note* $\mathbf{E[X]}$*, « l'espérance de* $\boldsymbol{X}$ *».*
  — *Si* $\boldsymbol{X}$*, est discrète*[1]*, alors :*

$$\mathbf{E[X]} = \sum_{k \in X(\Omega)} k\mathbb{P}_X(\{k\})$$

  — *Si* $\boldsymbol{X}$*, est continue de densité* ρ*, alors :*

$$\mathbf{E[X]} = \int x\rho(x)\,dx$$

* **Fonction de Répartition F_X, de la variable aléatoire X :** *Soit* $\boldsymbol{X}$*, une variable aléatoire sur* $(\Omega, \mathcal{A}, \mathbb{P})$
  — *On note* $\forall x \in \mathbb{R}$, $F_X(x) = \mathbb{P}_X(]-\infty, x])$ *« la fonction de répartition de* $\boldsymbol{X}$ *».*
* **Espace $L^p(\Omega, \mathcal{A}, \mathbb{P})$:** *Soit* $(\Omega, \mathcal{A}, \mathbb{P})$
  — *On note* $L^p(\Omega, \mathcal{A}, \mathbb{P})$*, l'ensemble :*

$$\{\mathbf{X}\text{variable aléatoire sur } (\Omega, \mathcal{A}, \mathbb{P}) | \mathbf{E[}|\mathbf{X}|^p\mathbf{]} < +\infty\}$$

* **Moment d'ordre p m_p[X], de la variable aléatoire X :** $\forall\ \boldsymbol{X}, \in L^p(\Omega, \mathcal{A}, \mathbb{P})$
  — *On note* $m_p\mathbf{[X]} = \mathbf{E[X}^p\mathbf{]}$
* **Moment centré d'ordre p μ_p[X], de la variable aléatoire X :** $\forall\ \boldsymbol{X}, \in L^p(\Omega, \mathcal{A}, \mathbb{P})$
  — *On note* $\mu_p\mathbf{[X]} = \mathbf{E[(X - E[X])}^p\mathbf{]}$
* **Variantes du moment centré d'ordre 2, de la variable aléatoire X :** $\forall\ \boldsymbol{X}, \in L^2(\Omega, \mathcal{A}, \mathbb{P})$
  — *On note* $\mathbf{V}(X) = \mu_2\mathbf{[X]}$*, la : « variance de* $\boldsymbol{X}$ *»*
  — *On note* $\sigma_X = \sqrt{V(X)}$*, « l'écart-type de* $\boldsymbol{X}$ *»*
* **Covariance, entre les variables aléatoires X et Y :** $\forall(\boldsymbol{X}, \boldsymbol{Y}) \in L^2(\Omega, \mathcal{A}, \mathbb{P})^2$
  — *On note* $\mathbf{Cov}(X, Y) = \mathbf{E[(X - E[X])(Y - E[Y])]}$*, la : « covariance entre* $\boldsymbol{X}$ *et* $\boldsymbol{Y}$ *»*
  — $(L^2(\Omega, \mathcal{A}, \mathbb{P}), \boldsymbol{Cov})$*, forme un « espace euclidien »*
* **Coefficient de corrélation linéaire, entre les variables aléatoires X et Y :** $\forall(\boldsymbol{X},\ \boldsymbol{Y}) \in L^2(\Omega, \mathcal{A}, \mathbb{P})^2$
  — *On note* $\rho_{X,Y} = \dfrac{\mathbf{Cov(X,Y)}}{\sigma_X \sigma_Y}$*, la : « le coefficient de corrélation linéaire entre* $\boldsymbol{X}$ *et* $\boldsymbol{Y}$ *»*

1. $\mathbb{P}_X$ « μ, ou μ est : la « mesure de comptage ».

0.3 Objets du Problème

On cherche à approximer la probabilité de ruine pour un modèle de théorie de la ruine en assurance moto. On étudie deux cas : un cas où les assurés sont indépendants, et un cas où ils sont corrélés à travers des conditions météo communes.

* **La variable aléatoire** X_{norm} **sur** $(\mathbb{R}, \mathbb{B}(\mathbb{R}))$ **:** $\forall (x_0, b, \sigma, \delta) \in \mathbb{R} \times (\mathbb{R}_+^*)^3$
  - *On note* X_{norm},
    - *la variable aléatoire continue de densité :* f_{norm} *:*

$$\forall x \in \mathbb{R},\ f_{norm}(x) = \mathbb{1}_{[0,b]}(x)\ exp(-\frac{(x-x_0)^2}{2\sigma^2})\ (1 + cos(2\pi\frac{x-x_0}{\delta})^2)$$

* **La variable aléatoire** $X_{puissance}$ **sur** $(\mathbb{R}, \mathbb{B}(\mathbb{R}))$ **:** $\forall (a, \alpha) \in \mathbb{R}_+^* \times]1 : +\infty[$
  - *On note* $X_{puissance}$,
    - *la variable aléatoire continue de densité :* $f_{puissance}$[1] *:*

$$\forall x \in \mathbb{R},\ f_{puissance}(x) = \mathbb{1}_{[a,+\infty[}(x)\ x^{-\alpha} \times \frac{\alpha - 1}{a^{1-\alpha}}$$

* **La variable aléatoire** Z **sur** $(\Omega, \mathcal{A}, \mathbb{P})$ **:** *Soit* $Z(\Omega) = \mathbb{N}$ *et* $(p_n)_{n\in\mathbb{N}} \in [0,1]^{\mathbb{N}}$ *et tel que :* $\sum_{n\in\mathbb{N}} p_n = 1$
  - *On note* Z,
    - *la variable aléatoire discrète, dont les probabilités respectives sont ainsi notées :*

$$\forall n \in \mathbb{N},\ p_n = \mathbb{P}(Z = n)$$

    - *On désignera par la suite sa probabilité de la sorte :* $\mathbb{P}_Z$

* **La variable aléatoire X sur** $(\Omega, \mathcal{A}, \mathbb{P})$ **:** *Soit :* X_{norm}, $X_{puissance}$, Z, *des variables aléatoires mutuellement indépendantes, de lois (et ou de densités) respectives :* f_{norm}, $f_{puissance}$ *et* $\mathbb{P}_Z$.
  - *On note* **X**,
    - *la variable aléatoire, définie de la sorte :*

$$\mathbf{X} = \begin{cases} X_{norm} \text{ si Z} = 0 \\ X_{puissance} \text{ si Z} = 1 \\ Z \text{ sinon} \end{cases}$$

File keyword
Run keyword
Ouvrir l'annexe
test
File.txt

1. This is my link

1 Organisation du travail

Concernant l'organisation du travail, nous avons jugé nécessaire de mettre en place un environnement permettant de partager facilement nos ébauches de code, l'avancée du rendu ainsi que tout document utile au bon déroulement du projet. Pour cela, nous avons créé une page *GitHub* dédiée.

La communication s'est faite principalement via un groupe de discussion sur les réseaux sociaux, mais aussi lors d'échanges directs en classe ou au cours de réunions informelles consacrées à l'avancement du projet.

Sur le plan technique, la première séance en classe a été consacrée à l'analyse de chaque question afin de déterminer les méthodes de simulation à utiliser et les optimisations possibles. Suite à cela, le rôle de *Priscilla* a été plutôt dirigé vers les implémentations de techniques de simulations « classiques » tandis que *Mahlî* a lui été plutôt chargé du côté optimisation du code notamment : *table de Walker*, classes...

Cela dit, de nombreuses parties du code ont été développées conjointement : chacun ayant parfois besoin des conseils, des idées ou de l'expertise de l'autre pour la mise en place de structures de données adaptées et pour garantir la cohérence globale de l'implémentation.

Enfin, nous avons choisi de travailler sur un *Notebook Python*, un format qui nous a semblé plus pratique pour organiser notre code, structurer nos sections et ajouter facilement titres et commentaires. Par souci de clarté et de rigueur, nous avons rédigé les démonstrations mathématiques associées aux différentes étapes de nos algorithmes. Conscients que ces éléments ne constituent pas le cœur de la matière, nous avons choisi de les regrouper dans une annexe, où elles restent consultables si nécessaire.

1.1 Notes d'updates

Une section que l'on supprimera lors du projet final, mais qui me permet de laisser quelques petits commentaires.
Autant je peux concevoir l'utilité de faire une section optimisée et non optimisée pour X_{norm}, autant (comme tu l'as très justement fait remarquer) je ne pense pas que cela soit pertinent pour $X_{puissance}$.
Note a moi même : enlevé les tests en fin de page 5.
Note a moi même : italique sur les commentaires.
Note a moi même : enlevé les _ de x_0 dans les codeblock
Tenter la réduction de variance sur le modèle A

Deuxième partie

Aspects théoriques

2 modélisation

2.1 simulation de X_{norm}

2.2 Simulation de $X_{puissance}$

$$\forall (a,\alpha) \in \mathbb{R}_+^* \times]1:+\infty[,$$

$X_{puissance}$ est une variable aléatoire continue, de densité $f_{puissance}$, tel que $\forall x \in \mathbb{R}$:

$$f_{puissance}(x) = \mathbb{1}_{[a,+\infty[}(x)\ x^{-\alpha} \times \frac{\alpha-1}{a^{1-\alpha}}$$

$$\Updownarrow$$

$$\forall t \in \mathbb{R}, F_{X_{puissance}}(t) = \mathbb{P}_{X_{puissance}}(]-\infty:t]) = \int_{-\infty}^{t} f_{puissance}(x)dx$$

$$= \int_{-\infty}^{t} \mathbb{1}_{[a,+\infty[}(x)\ x^{-\alpha} \times \frac{\alpha-1}{a^{1-\alpha}}dx$$

$$= \mathbb{1}_{[a,+\infty[}(t) \int_a^t\ x^{-\alpha} \times \frac{\alpha-1}{a^{1-\alpha}}dx = \mathbb{1}_{[a,+\infty[}(t) \frac{\alpha-1}{a^{1-\alpha}} \int_a^t\ x^{-\alpha}dx$$

$$= \mathbb{1}_{[a,+\infty[}(t) \times \frac{\overbrace{\alpha-1}^{=-(1-\alpha)}}{a^{1-\alpha}} \times \left[\frac{x^{1-\alpha}}{1-\alpha}\right]_a^t = -\mathbb{1}_{[a,+\infty[}(t) \times \left[\frac{x^{1-\alpha}}{a^{1-\alpha}}\right]_a^t$$

$$= \mathbb{1}_{[a,+\infty[}(t) \times \left[\frac{x^{1-\alpha}}{a^{1-\alpha}}\right]_t^a = \mathbb{1}_{[a,+\infty[}(t) \times \underbrace{\left[x^{1-\alpha}\right]_t^a}_{=a^{1-\alpha}-t^{1-\alpha}} \times a^{\alpha-1}$$

$$= \mathbb{1}_{[a,+\infty[}(t) \times \left(1 - \left(\frac{t}{a}\right)^{1-\alpha}\right)$$

Posons $F_{X_{puissance}}^{-}$, la fonction définit de la sorte : $\forall y \in]0,1[$:
$F_{X_{puissance}}^{-}(y) = \inf\left\{x \in \mathbb{R} \mid y \leq \mathbb{1}_{[a,+\infty[}(x) \times \left(1 - \left(\frac{x}{a}\right)^{1-\alpha}\right)\right\}$

Or $y \in]0,1[\iff 0 < y \iff \left\{x \in \mathbb{R} \mid y \leq \mathbb{1}_{[a,+\infty[}(x) \times \left(1 - \left(\frac{x}{a}\right)^{1-\alpha}\right)\right\} = \left\{x \in [a,+\infty[\mid y \leq \left(1 - \left(\frac{x}{a}\right)^{1-\alpha}\right)\right\}$

$$\inf\Big\{x \in [a,+\infty[\mid y \leq 1-\Big(\frac{x}{a}\Big)^{1-\alpha}\Big\} = \inf\Big\{x \in [a,+\infty[\mid y - 1 \leq -\Big(\frac{x}{a}\Big)^{1-\alpha}\Big\} = \sup\Big\{x \in [a,+\infty[\mid \Big(\frac{x}{a}\Big)^{1-\alpha} \leq 1-y\Big\} = \sup\Big\{x \in [a,+\infty[\mid \Big(\frac{x}{a}\Big) \leq \sqrt[1-\alpha]{1-y}\Big\} = \sup\Big\{x \in [a,+\infty[\mid x \leq a\sqrt[1-\alpha]{1-y}\Big\}$$

$$\text{Or } a\sqrt[1-\alpha]{1-y} = \underbrace{\frac{a}{\sqrt[\alpha-1]{1-y}}}_{\in]0,1[} \in [a,+\infty[\iff$$

$$F^{-}_{X_{puissance}}(y) = sup\Big\{x \in [a,+\infty[\mid x \leq a\sqrt[1-\alpha]{1-y}\Big\} = sup[a, a\sqrt[1-\alpha]{1-y}] = \boxed{a\sqrt[1-\alpha]{1-y}}$$

A l'aune de cette information nouvelle, nous pouvons établir notre modèle de la sorte :

Soit $U \sim \mathcal{U}(]0,1[)$, $F^{-}_{X_{puissance}}(U) \sim X_{puissance}$

2.3 Conditions météorologiques et chaines de Markov $(H_k)_{k\in\mathbb{N}^*}$

L'on cherche à réaliser un modèle simulant des observations journalières de nos états météorologique.
Pour ce faire (et de façon assez naturelle, nous établirons une *chaine de Markov* $(H_k)_{k\in\mathbb{N}^*}$, possédant les propriétés suivantes :

* **Processus Aléatoire** $(H_k)_{k\in\mathbb{N}^*}$ **:** *Soit* $(H_k)_{k\in\mathbb{N}^*} \in (E,\mathcal{A},\mathbb{P})^{\mathbb{N}^*}$

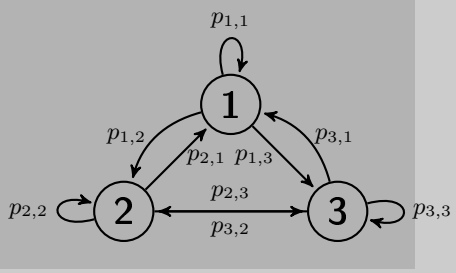
— *On note* $E = \{\text{beau temps}, \text{temps couvert}, \text{pluie}\}$, dit *« ensemble des états de* $(H_k)_{k\in\mathbb{N}^*}$ *»*.

— *On note* μ_0*, une mesure de probabilité sur* $(E,\mathcal{A})$*, dite « loi initiale de* $(H_k)_{k\in\mathbb{N}^*}$ *», tel que* $(\mu_0(i))_{i\in[|1,3|]}$*, est une permutation quelconque de* $(1,0,0)$.

— *On note* $Q \in M_3(\mathbb{R})$*, une matrice stochastique, dite « matrice de transition de* $(H_k)_{k\in\mathbb{N}^*}$ *», tel que* $\forall k \in \mathbb{N}^*$[1] :

$$\begin{bmatrix} \mathbb{P}(H_{k+1}=1|H_k=1) & \mathbb{P}(H_{k+1}=2|H_k=1) & \mathbb{P}(H_{k+1}=3|H_k=1) \\ \mathbb{P}(H_{k+1}=1|H_k=2) & \mathbb{P}(H_{k+1}=2|H_k=2) & \mathbb{P}(H_{k+1}=3|H_k=2) \\ \mathbb{P}(H_{k+1}=1|H_k=3) & \mathbb{P}(H_{k+1}=2|H_k=3) & \mathbb{P}(H_{k+1}=3|H_k=3) \end{bmatrix}$$

$$\begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,1} & p_{2,2} & p_{2,3} \\ p_{3,1} & p_{3,2} & p_{3,3} \end{bmatrix}$$



1. Pour des raisons évidentes de lisibilité nous confondrons les états *« beau temps »*, *« temps couvert »* et *« pluie »* avec les états respectifs : 1, 2, et 3.

Commentaires sur l'implémentation de la chaine de markov

Une première approche a été de partitionner notre intervalle : $[0, 1]$, de telle sorte à pouvoir simuler n'importe qu'elle variable aléatoire à support fini, à l'aide d'une loi uniforme.
Cette approche a le malheureux inconvénient d'avoir (parmi ses implémentations les moins naïves), une complexité de l'ordre de $o(nln(n))$[1].

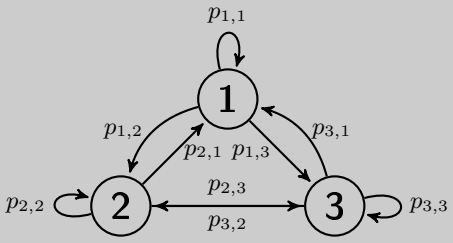
Comparativement, une autre approche dite de *« Table de Walker*[2] *»* (certes plus couteuse en temps d'initialisation), a le bon gout d'avoir une complexité de l'ordre : $o(1)$.

« La *méthode optimale* serait-elle fonction des conditions initiales ? ». La question ne manque pas pertinence, après tout notre loi a support fini ne possède que 3 états. En ce sens une *« Table de Walker »* est-elle réellement une méthode plus optimale que notre *approche naïve* ?

A cela nous avons deux objections :

— Pour $k \in \mathbb{N}$, la simulation d'un trajectoire a k étapes, d'une *chaine de Markov* à n états serait de l'ordre de $o(knln(n))$ avec notre approche naïve et de $o(k)$ avec notre *table de walker*[3].
— Simuler convenablement des conditions météorologique à l'aide d'une unique *chaine de Markov* a 3 états, nous apparait fort improbable. Il est fort à parier que l'utilisateur sera amené à faire varier le nombre d'états : n. En ce sens, notre *« modèle à Table de Walker »*, se montre beaucoup plus robuste que le précédent.

Ainsi, nous faisons donc le pari de la *robustesse* et de *l'adaptabilité* de notre modèle.



a. $n = |E|$ et E, l'ensemble des états de la variable aléatoire a support fini.

b. Vous trouverez en annexe la construction de notre *« Table de Walker »*.

c. Nous ne comptons pas l'étape d'initialisation, qui est dans le pire des cas d'ordre $o(n^2)$.

2.4 Occurrences des sinistres (modèle A)

À chaque état de notre *chaine de Markov* ($(1,2,3)$) est associé une valeur $\lambda_k \in \mathbb{R}_+^*$ (respectivement $(\lambda_1, \lambda_2, \lambda_3)$).
Notons $N \in \mathbb{N}$, la taille de notre portefeuille.
« On suppose que, pour chaque jour $k \in \mathbb{N}$, les dates des sinistres déclarés par l'assuré le jour k suivent, conditionnellement à la valeur de H_k, un processus de Poisson d'intensité λ_{H_k} ».

Ainsi, nous définissons :

* **Fonction** $\Delta(\omega) : : \mathbb{R}_+ \to \mathbb{R}_+$

$$\Delta : \begin{cases} \mathbb{R}_+ \to \mathbb{R}_+ \\ t \longmapsto \mathbb{1}_{[0,365]}(t)\lambda_{H_{\lfloor t \rfloor}} \end{cases} .$$

* **Fonction** $\mu(\omega) : : \mathbb{R}_+ \to \mathbb{R}_+$

$$\mu(t) = \int_0^t \Delta(x)dx = \int_0^t \mathbb{1}_{[0,365]}(x)\lambda_{H_{\lfloor x \rfloor}} dx$$

* **Suite de processus de poisson in-homogène mélange :** $(R_t^{(k)})_{k\in[|1,N|]}$

— $(R_t^{(k)})_{k\in[|1,N|]}$, *une suite de processus de poisson in-homogène mélange indépendant.*

— $\forall (k,t) \in [|1,N|] \times \mathbb{R}_+$, $R_t^{(k)} \sim P(\mu(t))$, *le processus de poisson in-homogène mélange indépendants représentant les temps d'occurrence des sinistres du contrat k.*

$$\forall t \in \mathbb{R}_+,\ R_t = \sum_{i=1}^{N} R_t^{(i)}$$

$$\forall t \in [0,365],\ \mu(t) = \int_0^t \Delta(x)dx = \int_0^t \mathbb{1}_{[0,365]}(x)\lambda_{H_{\lfloor x \rfloor}} dx = \int_0^t \lambda_{H_{\lfloor x \rfloor}} dx$$

$$= \int_0^1 \lambda_{H_{\lfloor x \rfloor}} dx + \int_1^2 \lambda_{H_{\lfloor x \rfloor}} dx + ... + \int_{\lfloor t \rfloor - 1}^{\lfloor t \rfloor} \lambda_{H_{\lfloor x \rfloor}} dx + \int_{\lfloor t \rfloor}^t \lambda_{H_{\lfloor x \rfloor}} dx$$

$$= \int_0^1 \lambda_{H_0} dx + \int_1^2 \lambda_{H_1} dx + ... + \int_{\lfloor t \rfloor - 1}^{\lfloor t \rfloor} \lambda_{H_{\lfloor t \rfloor - 1}} dx + \int_{\lfloor t \rfloor}^t \lambda_{H_{\lfloor t \rfloor}} dx$$

$$= \lambda_{H_0} + \lambda_{H_1} + ... + \lambda_{H_{\lfloor t \rfloor - 1}} + \int_{\lfloor t \rfloor}^t \lambda_{H_{\lfloor t \rfloor}} dx$$

$$= \sum_{i=0}^{\lfloor t \rfloor - 1} \lambda_{H_i} + \lambda_{H_{\lfloor t \rfloor}} \times (t - \lfloor t \rfloor)$$

$$\forall t \in \mathbb{R}_+,\ R_t = \sum_{i=1}^{N} \underbrace{R_t^{(i)}}_{\sim P(\mu(t))} \sim P(N \times \mu(t)) \iff$$

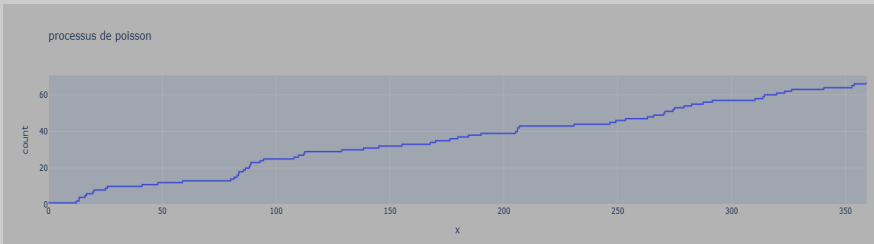
$R_t$est un processus de poisson in-homogène mélange d'intensité : $\Delta^* = N\Delta$

Il nous suffit donc de simuler une seule loi de poisson pour nos N contrats.

Posons donc : $\forall t \in \mathbb{R}$, $\Delta^*(t) = N\mathbb{1}_{[0,365]}(t)\lambda_{H_{\lfloor t \rfloor}}$.

Cette définition a le bon gout d'être aisément majorable par une formule simple : $\theta = N \times max(\lambda_1, \lambda_2, \lambda_3)$.

Montrons nous toutefois sceptiques vis a vis de notre modèle. Partir du postulat que nos assurés partage une météo commune, n'est pas une idée totalement absurde dans l'absolue, mais reste néanmoins extrêmement discutable dans les faits. Il faudrait pour cela faire l'hypothèse que nos assurés vivent sur un même territoire et que ce territoire est assez restreint pour que sa météo reste uniforme et uni-variée.
Le modèle B, fait l'hypothèse inverse (qui est tout aussi discutable). Notre modèle A repose sur une autre hypothèse : que l'état de la météo au jour n, dépends uniquement de l'état de la météo au jours $n-1$. Nous n'expliqueront pas a quel point cette hypothèse est absurde, néanmoins la valeur d'un modèle ne se limite pas a la somme de la valeur de ses axiomes (ex : les modèles de base de la micro-économie).



2.5 Probabilité de Ruine Annuelle (modèle A)

* **Modèle de Réserves :** $\forall (N, u, c) \in \mathbb{N} \times \mathbb{R}^2$, $(T_i)_{i \in \mathbb{N}^*}$, *une suite de variables aléatoires identiquement distribuées et croissantes.*

— *On note N, la taille de notre portefeuille.*

— *On note u, l'investissement initial et individuel par assurés.*

— *On note c, le taux de prime par assuré et par unité de temps.*

— *On note $(T_i)_{i \in \mathbb{N}^*}$, la suite des dates de sinistres déclarés (de tous les assurés). La suite est par ailleurs ordonnée.*

— $\forall t \in \mathbb{R}^+$:

$$R_t = Nu + Nct - \sum_{i=1}^{+\infty} \mathbb{1}_{[0,t]}(T_i)\ X_i$$

* **Partition A de $[0, 365]$:** $\forall k \in \mathbb{N}$.

— $n = inf\{k \in \mathbb{N}^* | T_k \in [0, 365]\}$

$$A_k = \begin{cases} [0, T_1[\text{ , si k = 0} \\ [T_k, T_{k+1}[\text{ , si } k \in [|1, n-1|] \\ [T_n, 365] \text{ , si } k = n \\ \emptyset \text{ , sinon} \end{cases}$$

— $(A_k)_{k \in \mathbb{N}}$ *forme une partition de* $[0, 365]$.

Soit : $\forall \omega \in \Omega$, $R_t(\omega)$, est bien définie sur $[0, 365]$. On a donc :

$$\min_{t \in [0,365]} R_t(\omega) = \min_{k \in [|0,n|]} (\min_{t \in A_k} R_t(\omega))$$

Cette partition $((A_k)_{k \in \mathbb{N}})$, nous permet d'exprimer tout minimum local $(\min_{t \in A_k} R_t(\omega))$, sous une forme explicite (plus ou moins simple) :

$$\forall (k, t) \in \mathbb{N} \times A_k,\ R_t(\omega) = Nu + Nct - \sum_{i=1}^{+\infty} \mathbb{1}_{[0,t]}(T_i(\omega))\ X_i(\omega)$$

$$= Nu + Nct - \sum_{i=1}^{k} X_i(\omega) \iff$$

$$\min_{t \in A_k} R_t(\omega) = \begin{cases} Nu \text{ , si k = 0} \\ Nu + NcT_k(\omega) - \sum_{i=1}^{k} X_i(\omega) \text{ , si } k \in [|1, n|] \end{cases}$$

, car par soucis de réalisme l'on considère que $c \in \mathbb{R}_+^*$.

Nous avons donc le résultat suivant :

$$\min_{t\in[0,365]} R_t(\omega) = Nu + \min_{k\in[|1,n|]}(NcT_k(\omega) - \sum_{i=1}^{k} X_i(\omega))$$

$$= Nu + \min_{k\in[|1,n|]}(\sum_{i=1}^{k}(\frac{NcT_k(\omega)}{k} - X_i(\omega)))$$

Ce résultat nous permet de caractériser l'événement suivant :

$$\left(\min_{t\in[0,365]} R_t < 0\right) = \left(\exists k \in [|1,n|] \text{ , tel que : } \sum_{i=1}^{k}(\frac{NcT_k}{k} - X_i) < -Nu\right)$$

Il nous suffirait donc de procédé pas à pas.

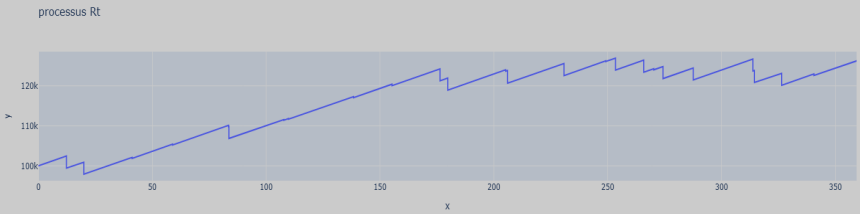
Soit $(V_j)_{j\in\mathbb{N}}$, une suite de v.a.i.i.d., tel que $\forall j \in \mathbb{N}$,

$$V_j \sim \mathbb{B}(\mathbb{P}\left(\exists k \in [|1,n|] \text{ , tel que : } \sum_{i=1}^{k}(\frac{NcT_k}{k} - X_i) < -Nu\right))$$

Par loi des grands nombres nous avons :

$$\frac{1}{p}\sum_{j=1}^{p} V_j \xrightarrow[p\to+\infty]{p.s.} \mathbb{P}\left(\min_{t\in[0,365]} R_t < 0\right)$$

Le fait que nous ayons deux sommes (d'une part $\sum_{i=1}^{k}(\frac{NcT_k}{k} - X_i)$ et d'une autre $\sum_{j=1}^{p} V_j$), rend le problème parallélisable.



Réduction de variance

2.6 Occurrences des sinistres (modèle B)

2.7 Probabilité de Ruine Annuelle (modèle B)

Troisième partie

Aspects Programmation

3 Commentaires sur les techniques de programmation

3.1 Dépendances

```
1  import random as rd
2  import numpy as np
3  import matplotlib.pyplot as plt # graphics
4  import plotly.express as px # graphics
5  import threading # parallélisation
6  import bisect # insertion dans une liste triée
```

3.2 X_{norm}

f_{norm}, est une fonction complexe et au vue de sa forme, il parait fort peu probable de trouver une forme explicite à son intégrale (sur un intervalle quelconque).

Une méthode similaire à celle de *Box-Muller*, a brièvement été évoquée, sans succès.

« La méthode du rejet semble donc s'imposer d'elle même ? ». Il s'agit encore d'une question d'arbitrage. Faut-il faire le choix de la meilleure des complexités, en approchant la fonction de répartition de X_{norm}, par des formules explicites incorrectes, ou préférer à la complexité une juste représentation de sa loi ?

Nous avons de prime abord opté pour l'implémentation d'une méthode de rejet dont le support serait la loi uniforme $U([0,b])$[1]. De plus, la majoration suivante, apparait de façon quasi-immédiate : $f_{norm} \leq 2$.

a. La densité f_{norm} étant nulle en dehors de l'intervalle $[0,b]$, il est naturel d'utiliser comme loi de proposition une uniforme sur ce même intervalle.

Algorithm 1 Algorithme de rejet

```
1: i ← True
2: while i do
3:     U ←∼ U([0, b])
4:     Y ←∼ U([0, 2])
5:     if Y ≤ f_norm(U) then return U
6:     end if
7: end while
```

Première approche :

```
def f_norm(x : float, x_0 : float, b : float, sigma : float, delta : float) :
    ind = (x >= 0) & (x <= b)
    return ind*(np.exp(-((x - x0)**2) / (2 * sigma**2)) * (1 + np.cos(2*np.pi*(x - x0)/delta)**2))
```

```
def echantillon_f_norm(x_0 : float, b : float, sigma : float, delta : float, n : int) :
    res = []
    while len(res) < n :
        X = np.random.uniform(0, b) # f_ norm est nulle sur [b,inf]
        Y = np.random.uniform(0, M)
        if Y <= f_norm(X,x0,b,sigma,delta) :
            res.append(X) # condition d'acceptation
    return res
```

Deuxième approche :

Conscients que cette méthode n'est sans doute pas la plus efficace au vue de la forme de f_{norm} nous avons donc opté pour une seconde méthode.
La loi $\mathbb{P}_{norm}$ étant très proche d'une loi normale, nous avons pensé à faire une méthode de rejet vis à vis de cette dernière.

Nous avons donc majoré le rapport $\frac{f_{norm}}{g}$, avec g la densité d'une loi normale.
Cela garantit une meilleure efficacité du rejet : la probabilité d'acceptation augmente, et donc le nombre moyen d'itérations[1] pour accepter un point diminue.

a. Qui fut précédemment de l'ordre de $2b$.

Algorithm 2 Algorithme de rejet

```
i ← True
while i do
    U ←∼ loi de densité g
    Y ←∼ U([0, M])
    if Y × g(U) ≤ f(U) then return U
    end if
end while
```

Nouvelle approche :

```
def g(x : float, x_0 : float, sigma : float) :
    ind = (x >= 0) & (x <= b)
    return (1 / (np.sqrt(2*np.pi)*sigma)) * np.exp(-((x - x0)**2) / (2 * sigma**2))
```

```
def echantillon_f_norm_opt(x_0 : float, b : float, sigma : float, delta :
float, n : int) :
    M = 2*np.sqrt(2*np.pi)*sigma
    res = []
    while len(res) < n :
        X = np.random.normal(x0, sigma)
        Y = np.random.uniform(0, M)
        if Y*g(X,x0,sigma) <= f_norm(X,x0,b,sigma,delta) :
            res.append(X) # condition d'acceptation
    return res
```

3.3 $X_{puissance}$

Cette partie sera relativement succincte. Les causes de cette concision sont doubles :

— Premièrement car tous les calculs ont déjà été traités dans la partie ***Modélisation***.

— Deuxièmement car le code qui y est associé est lui même relativement succint.

La complexité est ici en temps constant et nous voyons mal comment optimiser ce code sans passer par un *interfacage c++*.

Le code dans toute sa beauté fonctionnelle :

```
def run_loi_de_puissance(a : float, alpha : float) :
    return a*(1 - rd.random())**(1/(1-alpha))
```

3.4 Z

Z suit une loi de probabilité qui nous est inconnue, il n'existe donc aucune *implémentation optimale* de cette dernière (que ce soit en terme de code et ou de complexité).

Il parait assez évident, qu'une approche par *« inversion de la fonction de répartition »*, serait complètement hors propos.

A terme, nous avons finalement opter pour une méthode de rejet (bien que la *« table de walker »* nous ait un instant effleurer l'esprit).

Par la suite nous restructurons notre code pour le rendre moins sensible a la casse.

La programmation orientée objet nous offre un paradigme bien plus adaptable et robuste au changement de paramètres.

Elle structure notre code et lui offre une architecture bien plus *arborescente*.

3.5 X

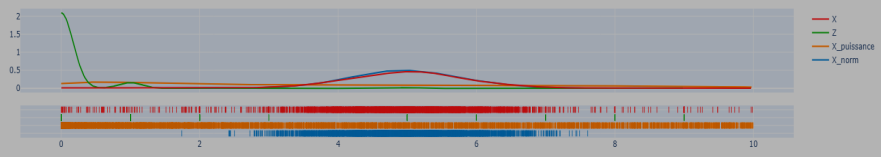
```
1   class Settings() :
2       def __init__(self, x_0=np.random.uniform(0,1),
    b=np.random.uniform(0.1,1), sigma=np.random.uniform(0.1,1),
    delta=np.random.uniform(0.1,1), a=np.random.uniform(0.1,1), al-
    pha=np.random.uniform(1,10), u=np.random.uniform(100,200),
    c=np.random.uniform(0.1,1), N=rd.randint(1,10000),
    monte_carlo_limit=10000, lambda_01=np.random.uniform(0.1,1),
    lambda_02=np.random.uniform(0.1,1), lambda_03=np.random.uniform(0.1,1)) :
3           """
4           Contient toutes les variables initiales de nos modeles
5           """
6           self.colors = "bg" : 0
7           self.parameters = { "x 0" : x_0,
8                   "b" : b,
9                   "sigma" : sigma,
10                  "delta" : delta,
11                  "a" : a,
12                  "alpha" : alpha,
13                  "u" : u,
14                  "c" : c,
15                  "N" : N,
16                  "limite machine de monte carlo limit" :
    monte_carlo_limit,
17                  "lambda 01" : lambda_01,
18                  "lambda 02" : lambda_02,
19                  "lambda 03" : lambda_03 }
20      def __repr__(self) :
21          return "Parametres du modèle : " + "\n" + "\n".join(["
    - " + str(elt) + " : " + str(self.parameters[elt]) for elt in
    self.parameters.keys()])
22      def __str__(self) :
23          return "Parametres du modèle : " + "\n" + "\n".join(["
    - " + str(elt) + " : " + str(self.parameters[elt]) for elt in
    self.parameters.keys()])
```

Bien que le choix de stocker nos paramètres ait brièvement été aborder, les nombreuses ouvertures et fermetures de fichiers successives auraient un impact néfaste et difficilement quantifiable sur la qualité de notre code.

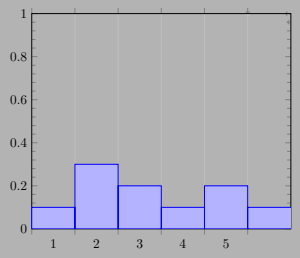
```
class Sinistre() :
    def __init__(self, settings=ProjectSettings, p=get_a_random_distribution(n=10)) :
        self.settings = settings
        self.p = p
    def f_norm(self, x : float) :
        """f_norm est la loi selon laquelle on doit simuler pour obtenir Pnorm"""
        ind = (x >= 0) & (x <= self.settings.parameters["b"])
        return ind*(np.exp(-((x - self.settings.parameters["x 0"])**2) / (2 * self.settings.parameters["sigma"]**2)) * (1+ np.cos(2*np.pi*(x - self.settings.parameters["x 0"])/self.settings.parameters["delta"])**2))
    def g(self, x : float) :
        return (1 / (np.sqrt(2*np.pi)*self.settings.parameters["sigma"])) * np.exp(-((x - self.settings.parameters["x 0"])**2) / (2 * self.settings.parameters["sigma"]**2))
    def f_norm_run(self) :
        while True :
            X = np.random.normal(self.settings.parameters["x 0"], self.settings.parameters["sigma"])
            Y = np.random.uniform(0, 2*np.sqrt(2*np.pi)*self.settings.parameters["sigma"])
            if Y*self.g(x=X) <= self.f_norm(x=X) :
                return X #condition d'acceptation, boucle brisée
    def loi_de_puissance_run(self) :
        return self.settings.parameters["a"]*(1 - rd.random())**(1/(1-self.settings.parameters["alpha"]))
    def loi_z_run(self) :
        M = np.max(self.p)
        k = len(self.p)
        while True :
            I = np.random.randint(0, k)
            U = np.random.uniform(0, M)
            if U <= self.p[I] :
                return I # condition d'acceptation
    def loi_X_run(self) :
        z = self.loi_z_run()
            return (z == 0)*self.f_norm_run() + (z == 1)*self.loi_de_puissance_run() + (z > 1)*z
    def empiricLikelihood(self, n=4000) :
        hist_data = [[self.f_norm_run() for i in range(n)],[self.loi_de_puissance_run() for i in range(n)],[self.loi_z_run() for i in range(n)],[self.loi_X_run() for i in range(n)]]
        group_labels = ["X_norm", "X_puissance","Z","X"]
        fig = ff.create_distplot(hist_data, group_labels, show_hist=False)
        fig.show()
```

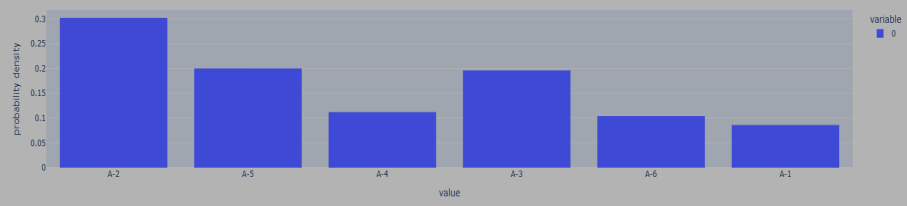
3.6 Vérifications d'usage

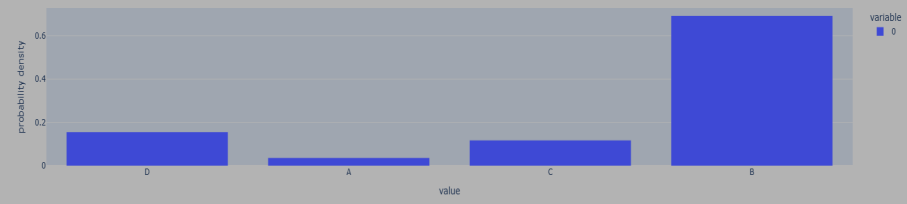
Nos *densités empiriques*, sont visiblement conformes (ou du moins vraisemblables) aux densités f_{norm}, $f_{puissance}$ et f_Z (ou $\mathbb{P}_Z$ conditionnellement a la nature de Z).



Qui plus est, nous tenons (dans un soucis de concision) a notifier que les vérifications d'usage pour nos *« tables de walker »* et autres *chaines de markov* (dont les codes ne seront pas détaillés ci-dessous au vu de leurs complexités) sont libre access sur les ***notebook***.







4 Les différents modèles

4.1 Modèle A

Première approche :

De prime abord, nous avons envisager de simuler étape par étapes notre sinistralité journalière, partant du postulat que l'agglomération de ces dernières formerait a terme notre sinistralité annuelle.
En cette idée réside le cœur de notre implémentation première.

```
def simuler_processus_poisson(lambd, horizon=1.0) :
    T = []
    tn = 0
    while True :
        tn += np.random.exponential(1/lambd)
        if tn > horizon :
            break;
        T.append(tn)
    return np.array(T)
```

La ***fonction*** simuler_sinistres_avec_temps renvoie une liste de tuples : (T_i : temps occurrence, X_i : montant sinistre associé).

La ***fonction*** simuler_processus_poisson, simule un processus de Poisson homogène sur $[0, horizon]$, puis renvoie la liste des temps d'occurrences T_n.

```
def simuler_sinistres_avec_temps(H, lambd, p, x0, b, sigma, delta, a,
alpha) :
    liste_T_X = [] # liste des couples (temps, montant)
    for jour in range(len(H)) :
        etat = H[jour] # on récupère la météo du jour
        lambda_j = lambd[etat - 1] # lambda associé à cette météo
        temps = simuler_processus_poisson(lambda_j)
        montants = echantillon_X_rejet(p, x0, b, sigma, delta, a,
alpha, len(temps)) # Simuler un montant pour chaque temps
        for i in range(len(temps)) :
            t=temps[i]
            liste_T_X.append((jour + t, montants[i]))
    return liste_T_X
```

Seconde approche :

Cette partie bénéficiera grandement des formules exposés en 3^{e} ***partie***, sur notre capacité a procéder étapes par étapes, afin d'optimiser notre code.
En premier lieux, nous générons nos possibles dates d'occurrences par une *simili* méthode de rejet, puis nous générons la trajectoire de la *chaine de markov* portant en elle l'information nécessaire a la bonne formation de nos critères de rejet.
Il n'est ainsi pas nécessaire de générer toute la trajectoire de la météo journalière, si l'information que celle-ci nous apporte ne nous est pas indispensable.

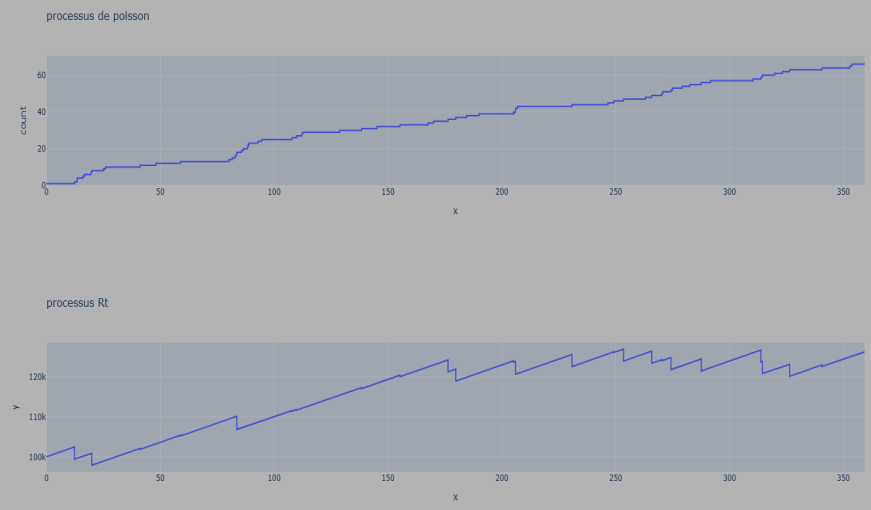
Algorithm 3 Algorithme de rejet

$\theta \leftarrow N \times max(\lambda_1, \lambda_2, \lambda_3)$
$k \leftarrow 0$
$N_{sim} \leftarrow\sim P(\theta)$
    Simuler $(T_1, ..., T_N)$ suivant les points d'un processus de Poisson homogène d'intensité θ dans $[0, 356]$
    Trier le vecteur $(T_1, ..., T_N)$
for $i = 0$ **to** N_{sim} **do**
    $U \leftarrow\sim U([0, 1[)$
  if La $\lfloor T_i \rfloor$-ème étape de $(H_k)_k$ a été simulée **then**
    if $\theta * U \leq \Delta(T_i)$ **then**
      $k \leftarrow k + 1$
      $T_k^{'} \leftarrow T_i$
    end if
  else
    Simuler la trajectoire jusqu'à la $\lfloor T_i \rfloor$-ème étape
    if $\theta * U \leq \Delta(T_i)$ **then**
      $k \leftarrow k + 1$
      $T_k^{'} \leftarrow T_i$
    end if
  end if
end for
    return $(T_1^{'}, ..., T_k^{'})$

L'implémentation ci-dessus est quelque peu inspirée d'une méthode de rejet. Par soucis d'honnêteté intellectuelle il nous faut reconnaitre la sensibilité de ce dernier aux valeurs importantes prises par $(\lambda_1, \lambda_2, \lambda_3)$.

```
1       def run_ruine(self, rounds=365, draw=True) :
2           N_sim = np.random.poisson(lam=self.theta*rounds) # Nous essayer d'optimiser le code de façon a que l'on ait besoin de simuler toute la trajectoire de la chaine de markov, si et seulement si l'information qu'elle apport nous ait nécessaire.
3           for k in range(N_sim) :
4               va_transitoire = np.random.uniform(0,rounds)
5               va_transitoire_aux = int(va_transitoire)
6               if (va_transitoire_aux in self.trajectory) : # recherche en temps constant
7                   if (np.random.uniform(0,self.theta) <= self.settings.parameters["N"]*self.trajectory[va_transitoire_aux]) :
8                       bisect.insort(self.T,va_transitoire) # o(lnn)
9                   else :
10                      for i in range(va_transitoire_aux - self.round + 2) : # Nombre de tours qu'il nous reste pour que va_transitioire_aux soit dans le dictionnaire
11                          self.state, self.round, self.trajectory[self.round] = self.transitions_walker_table[self.indexa[self.state]].run(), self.round + 1, self.state
12                          if (np.random.uniform(0,self.theta) <= self.settings.parameters["N"]*self.trajectory[va_transitoire_aux]) :
                                bisect.insort(self.T,va_transitoire) # o(lnn)
13          if draw :
14              X = [self.sinister.loi_X_run() for k in range(len(self.T))]
15              X[0] = 0
16              drawP(T=self.T, X=X)
17              drawRt(T=self.T, X=X, N=self.settings.parameters["N"], u=self.settings.parameters["u"], c=self.settings.parameters["c"])
```

4.2 Vérifications d'usage



4.3 Probabilité de ruine

Première approche :

```
def modele_A_meteo_commune(N_assures, u, c, horizon, M, H0, lambd, p, x0, b, sigma, delta, a, alpha) :
    H = simuler_chaine_markov(M, H0, horizon)
    T_all = []
    X_all = []
    for i in range(N_assures) :
        sinistres_i = simuler_sinistres_avec_temps(H, lambd, p, x0, b, sigma, delta, a, alpha)
        if len(sinistres_i) > 0 :
            T_i = np.array([t for (t, x) in sinistres_i])
            X_i = np.array([x for (t, x) in sinistres_i])
            T_all.append(T_i)
            X_all.append(X_i)
    # S'il n'y a aucun sinistre chez tous les assurés
    if len(T_all) == 0 :
        return np.array([]), np.array([]), None
    # On fusionne tous les sinistres dans une seule chronologie
    T_global = np.concatenate(T_all)
    X_global = np.concatenate(X_all)
    indices_tri = np.argsort(T_global) # retourne une permutation d'indices pour avoir les temps triés par ordre croissant
    T_global = T_global[indices_tri]
    X_global = X_global[indices_tri]
    # Calcul du processus de réserve agrégée
    Rt = N_assures * u + N_assures * c * T_global - np.cumsum(X_global)
    # Détection de la ruine
    indices_ruine = np.where(Rt < 0)[0]
    t_ruine = None
    if len(indices_ruine) > 0 :
        t_ruine = T_global[indices_ruine[0]]
    return T_global, Rt, t_ruine
```

Seconde approche :

```
def run(self) :
    self.run_ruine(draw=False)
    d_sum = 0
    x_sum = 0
    for k in range(1,len(self.T)) :
        d_sum = d_sum + self.settings.parameters["N"]*self.settings.parameters["c"]*self.T[k]
        x_sum = x_sum + self.sinister.loi_X_run()
        if (d_sum/k - x_sum <= -self.settings.parameters["N"]*self.settings.parameters["u"]) :
            global globsum_moda
            globsum_moda = globsum_moda + 1
            break;
```

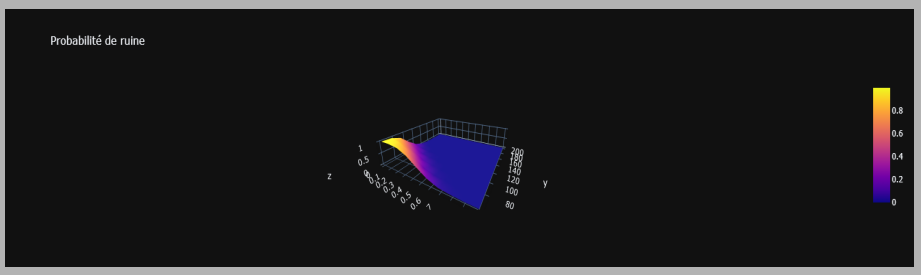


FIGURE 1 – Évolution de la probabilité de ruine en fonction de u et de c

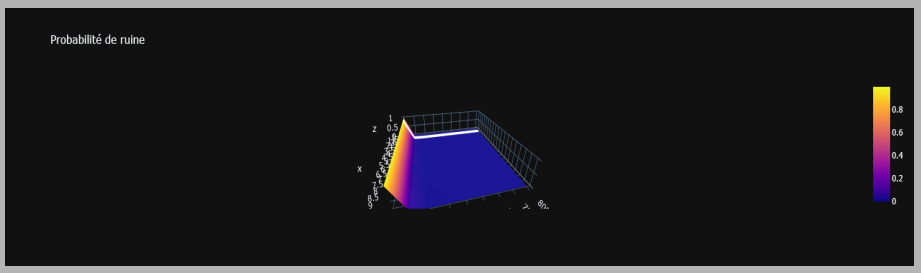


FIGURE 2 – Évolution de la probabilité de ruine en fonction de a et de α

4.4 Modèle B

4.5 Complexités algorithmiques

Quatrième partie
Conclusion

5 Commentaires sur la pertinence du modèle

6 Évolutions possibles du modèle

Interfaçage avec d'autres langages peut-être plus performants :
R, C++
NB : vérifier que R soit plus performant que python.

7 Commentaires

8 Annexes et contacts

Annexes et contacts