

PROJET : Simulation du modèle de Lundberg

Ruine en Assurance



Groupe 2 – MF01

ZIRIHI YANNICK
KOUROUMA ABOUBACAR
EL KASMI ADNANE

*Rapport du projet Maths/Finance
2019-2020 : Simulation du modèle de
Lundberg.*

Table des matières

1. Introduction	1
2. Tâches et obligations	2
2.1 Répartition des taches	2
2.2 Exigences	2
3. Cahier des charges	2
4. Modélisation	3
4.1 Modèle de risque classique (Cramer-Lundberg)	3
4.2 Probabilité de ruine	4
4.3 Simulation MC & MC	5
5. Synthèse Livrable 1	6
5.1 Activité 1. Etudes mathématiques	6
5.2 Activité 2. Simulation Monte-Carlo	7
6. Livrable 2	14
6.1 Activité 1. Simulation Monte-Carlo	14
6.2 Activité 2. Etudes mathématiques	20
7. Travail à faire	22
7.1 Risque de ruine à l'horizon fini T	22
7.2 Risque de ruine à l'horizon infini	24
7.3 Risque de ruine à l'horizon 1 an est un évènement rare	25
7.4 Compagnie diversifiée	26
8. Ce que nous avons réussi	28
9. Problèmes rencontrés	29
10. Conclusion	30
Annexe 1	31
Annexe 2	35
Annexe 3	38
Annexe 4	39
Annexe 5	40
Annexe 6	41
Annexe 7	42
Annexe 8	43
Annexe 9	44

■ RÉSUMÉ:

L'objectif de ce travail est d'évaluer la probabilité de ruine en temps fini et infini d'une compagnie d'assurance. Afin d'identifier le modèle de risque correspondant et de calculer ces caractéristiques, nous nous appuyons sur l'approche stochastique et les résultats de la théorie de la ruine avec un ajustement des données collectées. De plus, nous réalisons une approche de simulation pour estimer la probabilité de ruine en temps fini de cette branche d'activité.

Mots clés : Assurance ; Modèles de risque ; Probabilité de ruine ; Simulation.

■ ABSTRACT:

The objective of this work is to evaluate the ruin probabilities within finite and infinite time of the Insurance. In order to identify the corresponding risk model and calculate these characteristics, we lean on the stochastic approach and the results of the ruin theory with an adjustment of the collected data. Furthermore, a simulation study is realized in order to estimate the ruin probability in finite time for this line of business.

Keywords: Insurance; Risk models; ruin probabilities; Simulation.

1. Introduction:

L'opération d'assurance a pour effet le transfert total ou partiel des conséquences financières du risque subi par l'assuré vers une société d'assurance. Les dépenses prises en charge par la société peuvent correspondre soit à des indemnités à verser à des tiers au titre de la responsabilité (civile, professionnelle, ou autre) de l'assuré, soit à la réparation des dommages subis par ce dernier. Mais qu'adviendrait-il de ces compagnies d'assurance lorsqu'elles courrent elles-mêmes un risque ?

En assurance, on qualifie de risque de ruine la probabilité que la réserve d'une compagnie d'assurance, qui est la différence entre le total des primes reçues et le total des montants de réclamations payées, devienne négative à un certain temps. À ce moment, on dit que la ruine apparaît du fait d'un mauvais calcul du taux de cotisation des assurés ou de sinistres trop importants à couvrir.

La théorie mathématique de l'assurance peut contribuer à promouvoir le développement de méthodes plus rationnelles dans la gestion des risques. Un des outils les plus puissants pour comprendre l'évolution de la richesse d'une compagnie d'assurance est la modélisation stochastique. L'équilibre à long terme des résultats de la compagnie d'assurance correspond à la notion mathématique de probabilité de ruine. Le concept de probabilité de ruine sera basé sur des modèles qui relèvent de la théorie du risque.

Nous serons amenés à considérer une compagnie d'assurance qui veut investir une certaine somme d'argent dans une branche d'assurance. Le modèle consiste à la représentation du niveau des réserves comme étant le résultat de la différence entre les recettes par primes chargées et les paiements dus aux sinistres enregistrés en tenant compte d'un capital initial. Le modèle de risque, unidimensionnel, composé d'une seule branche d'assurance, est un modèle utilisé pour décrire ce mécanisme d'arrivée

des sinistres et des montants des réclamations. Le modèle concerne l'assurance non-vie, c'est-à-dire, les assurances « dommages » ou « accidents » par opposition aux assurances vie qui présentent d'autres problèmes et relèvent d'une autre modélisation.

L'objectif de ce travail est de modéliser l'évolution de la richesse d'une compagnie d'assurance et d'évaluer sa probabilité de ruine, c'est-à-dire, la probabilité que les dommages à payer aux assurés dépassent les cotisations.

2. Tâches et obligations:

2.1 Répartition des tâches:

L'organisation et la répartition des tâches au sein de notre groupe est un enjeu clé pour la réussite du projet Simulation du modèle de Lundberg. Répartir les tâches en fonction des compétences informatiques et mathématiques de chacun constitue la base du travail en équipe. Adnane El KASMI et Aboubacar Kourouma se sont exclusivement occupés de la partie mathématique. Guiehoa Yannick Donald Zirihi et Adnane EL KASMI se sont chargés de simulation.

Afin que cela soit plus limpide, voici un tableau récapitulatif de la répartition du travail.

Elève / Tache	Mathématiques	Simulation
Aboubacar Kourouma	X	
Adnane El KASMI	X	X
Guiehoa Yannick Donald Zirihi		X

2.2 Exigences:

- ✓ Être motivé et savoir motiver.
- ✓ Chercher à s'impliquer, quel que soit son niveau.
- ✓ Aimer organiser.
- ✓ Avoir une connaissance globale du cahier des charges.
- ✓ Avoir des compétences en mathématiques et en MCMC.
- ✓ Savoir donner et recevoir du feedback, voire des critiques.

3. Cahier des charges:

Dans le cahier des charges écrit par Madame Irina Kortchemski, les réalisations exigées sont la simulation de plusieurs variables aléatoire par Monte-Carlo et la démonstration de plusieurs parties théoriques (mathématiques) afin de pourvoir calculer la probabilité de ruine par MC & MC naïf et par MC & MC en utilisant l'échantillonnage Préférentiel ou "Importante Sampling" la et la simulation du modèle de Lundberg.

4. Modélisation:

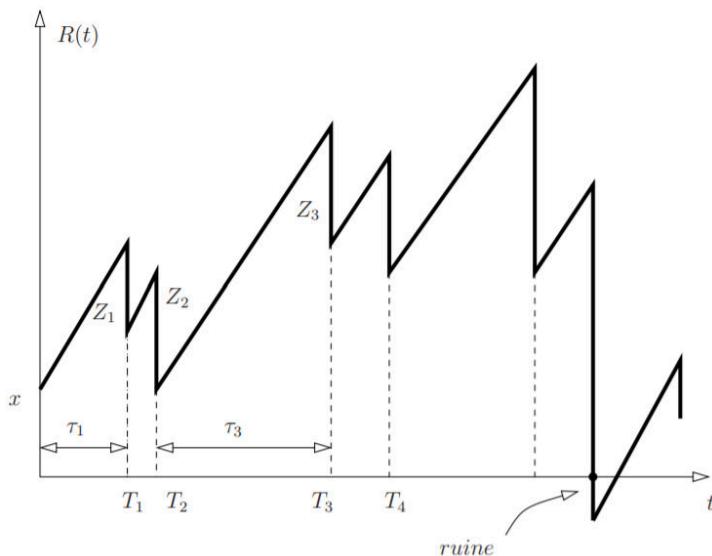
Le résultat d'une compagnie d'assurance à la fin de chaque exercice dépend de la réalisation de nombreuses activités. Dans ce travail, nous allons considérer l'activité provenant du côté purement assurance. C'est ainsi que le modèle sera réduit à la représentation du niveau des réserves comme étant le résultat de la différence entre les recettes par primes chargées et les paiements dû aux sinistres enregistrés. Cette étape concerne alors la modélisation de la réserve et de ses paramètres, à savoir, la prime, le nombre et le montant des réclamations des sinistres.

Le processus décrivant l'évolution d'un portefeuille d'assurance est défini comme suit : la société d'assurance dispose d'un capital initial x pour démarrer son activité. Un volume de primes $\Pi(t) = c \cdot t$ est perçu chaque année par la compagnie avec c représente le taux de cotisation par unité de temps. En contrepartie de ces primes, la compagnie doit dédommager des sinistres qui surviennent aléatoirement au cours du temps à des instants t_1, t_2, \dots . Les montants de ces sinistres sont Z_1, Z_2, \dots . Le résultat $X(t)$ de la branche d'activité à l'instant t est donné par :

$$R(t) = x + \pi(t) - X(t) = x + c \cdot t - \sum_{k=1}^{N_t} Z_k$$

qui représente la réserve de la branche considérée.

Nous nous intéressons à la détermination du processus $\{X(t), t \geq 0\}$.



4.1 Modèle de risque classique (Cramer-Lundberg):

Le modèle de risque de Cramer-Lundberg ou P/G, désigné aussi sous le nom du modèle de risque classique ou encore Poisson composé, a été introduit en 1903 par l'actuaire Suédois Filip Lundberg et est connu comme la base du fondement de la théorie du risque.

La notation P/G, empruntée de la théorie des files d'attentes, fournit l'information au sujet des lois des arrivées et des montants des réclamations des sinistres. La lettre G signifie général et P signifie

Poisson. Il s'ensuit que la suite $\{\tau_n\}_{n \in N}$ des arrivées des réclamations forme un processus de Poisson ce qui est équivalent à dire que les temps des inter-occurrences $T_n = \tau_n - \tau_{n-1}$ $n \geq 1$ sont de distribution exponentielle.

Ce modèle est construit selon les hypothèses suivantes :

- Le processus de comptage $\{N(t), t \geq 0\}$ du nombre de réclamations est un processus de Poisson d'intensité λ .
- La séquence $\{Z_n\}_{n \in N}$ des montants des réclamations est une suite de variables aléatoires indépendantes et identiquement distribuées de moyenne finie.
- La prime est proportionnelle au temps, c'est-à-dire, $\Pi(t) = c \cdot t$ où $c > 0$ est le taux de prime constant choisi de telle sorte que la société ait de bonnes chances de survie.
- Pour tout $t > 0$ et $n \geq 1$, on suppose que la variable aléatoire $N(t)$ et le vecteur aléatoire $\{Z_n\}_{n \in N}$ sont indépendants.

4.2 Probabilité de ruine:

→ Introduction aux modèles de ruines :

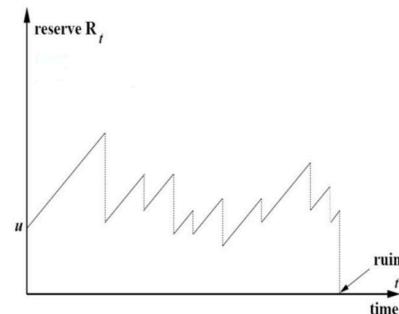
- **Ruine** = survenance d'un scénario défavorable, insolvabilité.
- **Modèle de ruine** = modélisation l'évolution de la richesse de la compagnie.
- **Probabilité de ruine** = probabilité de survenance de la ruine soit en horizon fini ou sur un horizon infini.

■ Probabilité de ruine **en temps infini** :

$$\psi(x) = P(\inf_{t \geq 0} R(t) < 0 | R(0) = x)$$

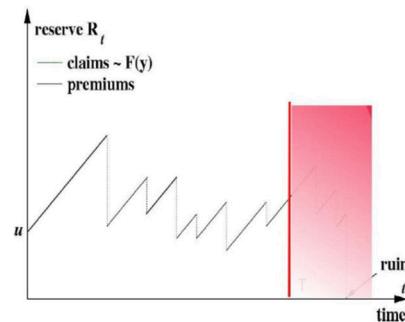
➤ L'instant de ruine :

$$T_x = \inf\{t > 0, R(t) < 0\}$$



■ Probabilité de ruine **en temps fini** :

$$\psi(x, T) = P(\inf_{0 \leq t \leq T} R(t) < 0 | R(0) = x)$$



4.3 Simulation MC & MC:

→Principe :

MC & MC : méthode de Monte Carlo par Chaine de Markov

Les différentes méthodes de simulation permettent de reproduire artificiellement des réalisations de certaines variables aléatoires qui modélisent souvent le fonctionnement des différents systèmes physiques, et qui respectent certaines lois de probabilités.

Ces réalisations permettent de faire des estimations des paramètres statistiques les plus importants des systèmes et ainsi tenter une étude plus efficace pour optimiser le fonctionnement et la fiabilité de ces systèmes.

→La simulation de Monte-Carlo :

Le principe de la simulation, au sens commun du terme, est d'utiliser un modèle, c'est-à-dire une représentation abstraite d'un système ou d'un problème, et d'étudier l'évolution de ce modèle sans faire fonctionner le système réel. La simulation de Monte-Carlo est une méthode d'estimation d'une quantité numérique qui utilise des nombres aléatoires.

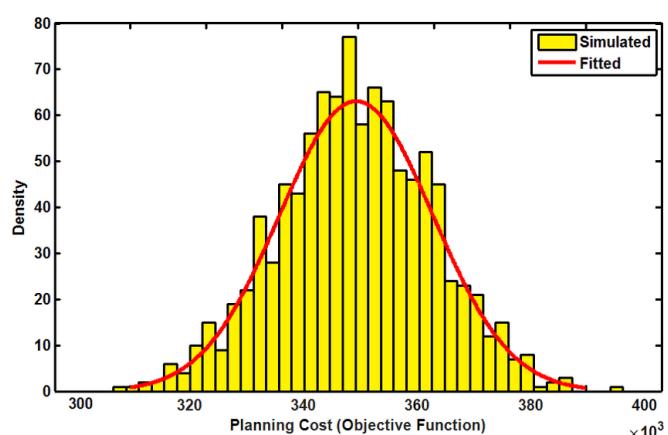
Elle présente le double avantage d'être simple d'utilisation et de pouvoir être appliquée à un très large éventail de problèmes. Elle est utilisée en finance, pour déterminer quand lever une option sur un bien financier ; en assurance pour simuler la probabilité de la ruine (ce qu'on va voir dans ce projet).

Pour cela, il faut cependant poser le problème, le modéliser de sorte que la quantité à rechercher s'exprime comme l'espérance d'une variable aléatoire X, notée $E(X)$. Une variable aléatoire est le résultat d'une expérience soumise au hasard ; son espérance est schématiquement ce que l'on s'attend à trouver en moyenne si l'on répète l'expérience un grand nombre de fois. Trouver cette modélisation est très probablement la tâche la plus complexe.

Dans certains cas, la quantité à calculer est naturellement une espérance, par exemple la moyenne du nombre de clients dans une file d'attente quand les temps d'arrivée et de traitement sont aléatoires. Mais la méthode peut également être utilisée pour estimer une quantité purement déterministe, par exemple une surface ou une intégrale, en construisant artificiellement une variable aléatoire pour se ramener au calcul d'une moyenne

Par exemple, dans cette figure on remarque que le résultat (en jaune) simulé par Monte-Carlo ressemble à la courbe (en rouge) qui représente le résultat réel du problème.

- ✓ **Avantage :** Facilité de mise en œuvre.
- **Inconvénients :** Il s'agit seulement d'une approximation des variables aléatoires.



5. Synthèse Livrable 1:

5.1 Activité 1. Etudes mathématiques:

Cette activité vise à nous familiariser avec des notions mathématiques de l'assurance et avec des bases de la théorie de la ruine : le modèle de Lundberg-Cramer.

- On nous propose de montrer théoriquement que le processus de comptage $N_t = \sum_{k=1}^{\infty} 1_{\{T_k \leq t\}}$

Avec $T_k = \sum_{i=1}^{\infty} \tau_i$ suit la loi de Poisson de paramètre λt . Pour cela vous suivez les étapes :

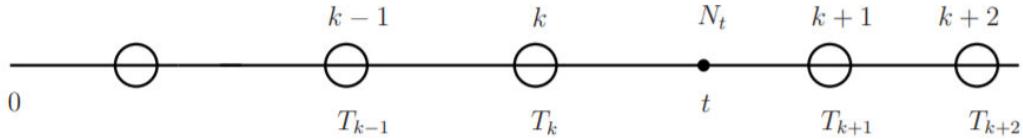
- v.a. τ_i suit la loi exponentielle : On a montré par récurrence **dans le livrable 1** que $T_k = \sum_{i=1}^{\infty} \tau_i$ suit la loi de Gamma(k, λ) de fonction de densité : $f_{T_k}(x) = \frac{\lambda}{(k-1)!} (\lambda x)^{k-1} e^{-\lambda x} 1_{x>0}$ en utilisant le théorème suivant :

Soit v.a ξ_1 possède une fonction de densité $f_{\xi_1}(x) = \lambda e^{-\lambda x} 1_{x>0}$

Et v.a ξ_2 possède une fonction de densité $f_{\xi_2}(x) = \lambda e^{-\lambda x} 1_{x>0}$

Alors : $Z = \xi_1 + \xi_2$ possède une fonction de densité $f_Z(x) = \int_{-\infty}^{+\infty} \xi_1(x-y) \xi_2(y) dy$

- Soit t un instant fixe. On a montré **dans le livrable 1** aussi que $P(N_t = k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$



En utilisant le fait que $P(N_t = k) = P(k \leq N_t < k + 1) = P(N_t \geq k) - P(N_t \geq k + 1)$ et que $P(N_t \geq k) = P(T_k \leq t)$, et en utilisant l'expression de la fonction de répartition de v.a T_k et T_{k+1} :

Soit $k \in N, t \in R_+$:

$$P(N_t \geq k) = P(T_k \leq t) = F_{T_k}(t) = \frac{\lambda^k}{(k-1)!} \int_0^t x^{k-1} e^{-\lambda x} dx = \frac{\lambda^k}{(k-1)!} \cdot I_k$$

En intégrant par parties, on obtient :

$$I_k = \left[\frac{x^k}{k} \cdot e^{-\lambda x} \right]_0^t + \int_0^t \lambda \cdot \frac{x^k}{k} e^{-\lambda x} dx = \frac{t^k}{k} \cdot e^{-\lambda t} + \frac{\lambda}{k} \cdot I_{k+1}$$

Donc : $P(N_t \geq k) = \frac{(\lambda t)^k}{k!} \cdot e^{-\lambda t} + \frac{\lambda^{k+1}}{k!} \cdot I_{k+1}$ Par conséquent, puisque le dernier terme est

égal à : $\frac{\lambda^{k+1}}{k!} \cdot I_{k+1} = P(N_t \geq k + 1)$

$$\text{Alors : } P(N_t = k) = P(N_t \geq k) - P(N_t \geq k + 1) = \frac{(\lambda t)^k}{k!} \cdot e^{-\lambda t}$$

-Calculer l'espérance :

$$E[X(t)] \text{ avec } X(t) = \sum_{k=1}^{N_t} Z_k$$

En utilisant l'espérance conditionnelle :

$$E[X(t)] = \sum_{k=0}^{\infty} E\left[\sum_{i=1}^{N_t} Z_i | N_t = k\right] \cdot P(N_t = k)$$

D'après le livrable 1, on a trouvé que :

$$E[X(t)] = \lambda t \cdot E[Z_1] = \lambda t \cdot \frac{\alpha}{\beta} \quad \text{avec} \quad E[Z_1] = \frac{\alpha}{\beta}$$

-Calculer une espérance suivante très utile pour la suite :

$$E[e^{\theta X(t)}] \text{ avec } \theta \in R^+$$

D'après le livrable 1, on a trouvé que :

$$E[e^{\theta X(t)}] = e^{\lambda t(E[e^{\theta Z_1}] - 1)} = e^{\lambda t\left(\left(\frac{\beta}{\beta-\theta}\right)^\alpha - 1\right)} \quad \text{avec} \quad E[e^{\theta Z_1}] = \left(\frac{\beta}{\beta-\theta}\right)^\alpha$$

5.2 Activité 2. Simulations de Monte-Carlo:

-Processus de comptage N_t par deux façons différentes : à partir de la définition

$$N(t) = \sum_{k=1}^{\infty} 1_{\{T_k \leq t\}} = 1_{\{\tau_1 \leq t\}} + 1_{\{\tau_1 + \tau_2 \leq t\}} + 1_{\{\tau_1 + \tau_2 + \tau_3 \leq t\}} + \dots$$

Pour $t = 1$, $\lambda = 2$, $\lambda = 5$ et $\lambda = \frac{1}{24}$; Tracer les graphes de fonction de densité de Poisson.

→ Méthode 1 : En simulant la loi de Poisson (λt) (voir livrable 1)

→ Méthode 2: En utilisant la définition de N_t

On simule la v.a Exponentielle par la méthode d'inversion de fonction de répartition qui est bijective

• Algorithme de simulation de v.a Exponentielle :

```

• function[τ] = V_A_Exponentielle(λ)
    ○ U = rand()
    ○ set τ = -1/λ . ln(1 - U)
• endfunction

```

- Algorithme de simulation de N_t :

```

• function[ $N_t$ ] = V_A_Processus_comptage( $\lambda, t, N$ )
    ○  $N_t = 0$ 
    ○  $T = 0$ 
    ○ for k = 1:N
        ○ for i = 1:k
            ○  $\tau(i) = V_A_Exponentielle(\lambda)$ 
            ○  $T = T + \tau(i)$ 
        ○ endfor
        ○ if T ≤ t
            ○  $N_t = N_t + 1$ 
        ○ endif
        ○ endfor
    • endfunction

```

Ici N représente un grand nombre pour remplacer l'infini.

- Algorithme de simulation d'une chaîne de N_t :

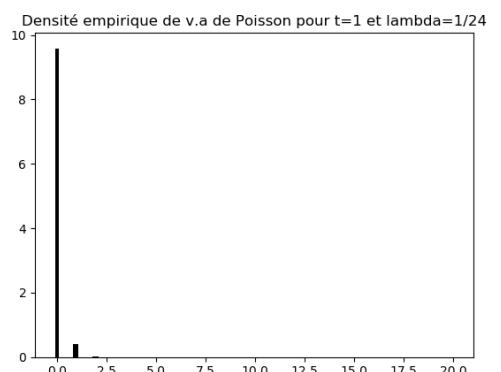
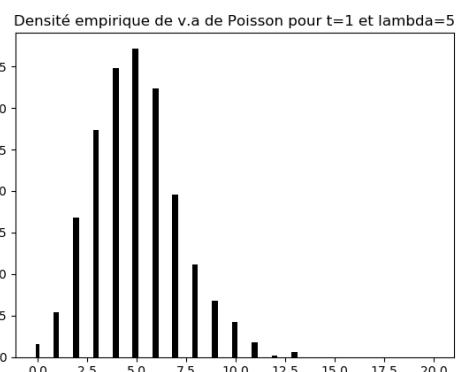
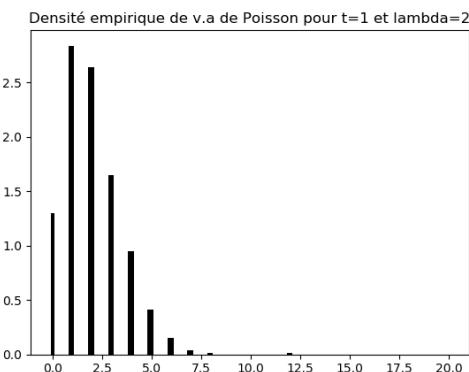
```

• function[ $N_t$ ] = Chaine_V_A_Processus_comptage( $\lambda, t, N$ )
    ○ for n = 1:  $N_{mc}$ 
        ○  $N_t(n) = V_A_Processus_comptage(\lambda, t, N)$ 
    ○ endfor
    • endfunction

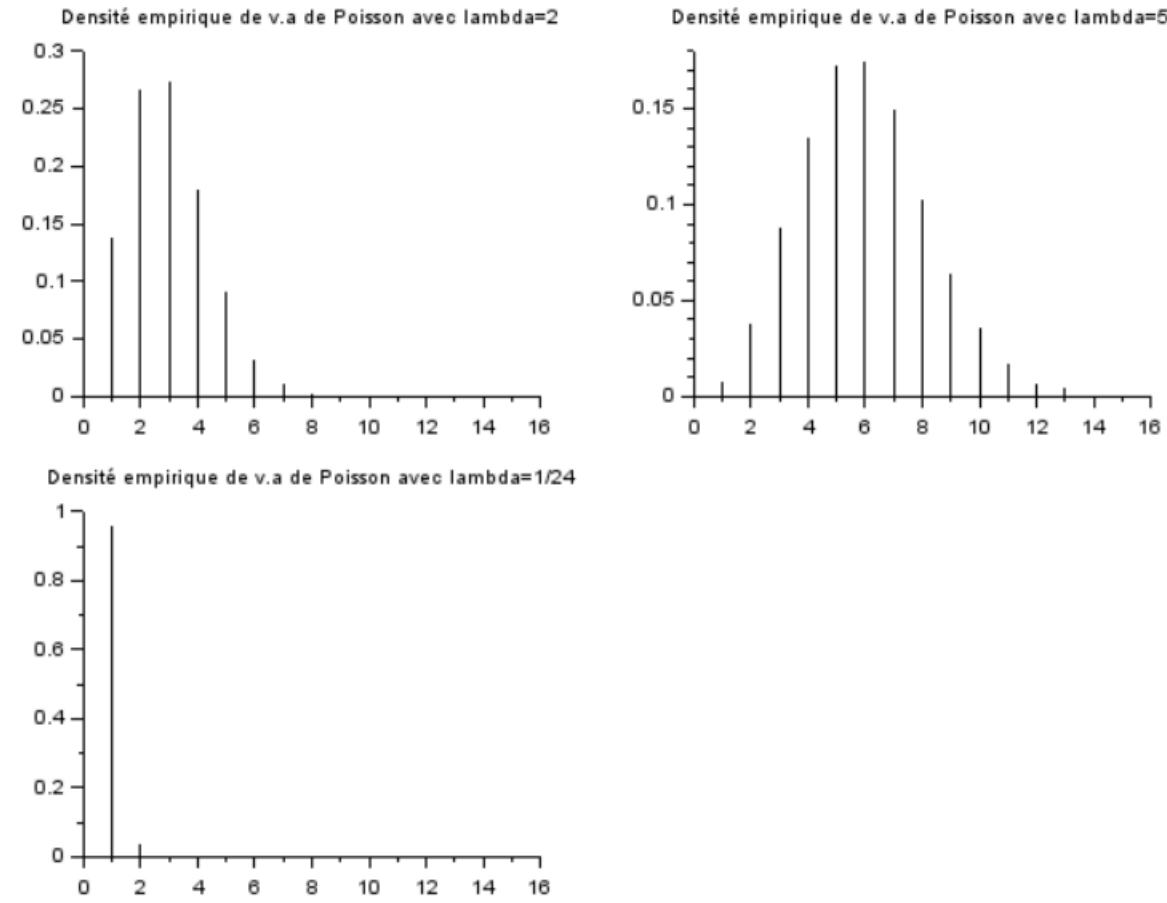
```

On peut vérifier la simulation grâce à la simulation de l'espérance empirique et la variance empirique.

→ Les graphes de fonction de densité de Poisson : Python



→ Les graphes de fonction de densité de Poisson : Scilab



(Codes Python et Scilab : voir annexe 1)

⊕ Variable aléatoire Gamma(α, β) pour les deux cas : α est un nombre entier et α est quelconque.

- Dans le cas $\alpha = n$ simulez n fois une v.a. Y_i qui suivent la loi exponentielle de paramètre β . La variable aléatoire : $X_n = Y_1 + Y_2 + Y_3 + \dots + Y_n$ suit la loi de Gamma (n, β).
- Dans le deuxième cas nous utilisons la méthode de Rejet (voir le ours MCMC).
- Nous traçons les graphes de fonction de densité de Gamma pour les différents paramètres :

$\text{Gamma}(1, \frac{1}{2}), \text{Gamma}(2, \frac{1}{2}), \text{Gamma}(3, \frac{1}{2}), \text{Gamma}(5, 1), \text{Gamma}(9, 2), \text{Gamma}(2.5, \frac{1}{2}), \text{Gamma}(3.5, \frac{1}{2})$

et comparer les graphes avec ceux de Wikipédia.

- Algorithme de simulation de loi $\text{Gamma}(\alpha = n, \beta)$: $n \in N$

```

• function[X] = V_A_Gamma(n, β)
  ○ Set Y = V_A_Exponentielle(λ)
  ○ U = rand(), i = 1
  ○ While i < n
    ○ Y = Y - 1/λ . ln(1 - U)
    ○ i = i + 1
  ○ endwhile
  ○ Set X = Y
• endfunction

```

Pour la simulation de la v.a $X \sim \text{Gamma}(\alpha, \beta)$ avec $\alpha > 0$ par la méthode de Rejet :

On sait déjà simuler $Y \sim \text{Gamma}(n, \delta)$ pour n entier, soit f_X et g_Y les fonction densité respectives.

D'après le calcul entamé dans **le livrable 1** on a pu avoir la constante de rejet $C > 1$ avec :

$$C = \sup_{x>0} \frac{f_X(x)}{g_Y(x)} = \frac{\Gamma(n).β^n}{\Gamma(α).δ^n} \cdot \left(\frac{α-n}{β-δ}\right)^{α-n} e^{-(α-n)} \quad \text{Sous les conditions : } β > δ \text{ et } n < α < n + 1.$$

D'après le théorème de rejet :

- Algorithme de simulation de loi $\text{Gamma}(\alpha, \beta)$ par la méthode de Rejet :

```

• function[X] = V_A_Gamma_Rejet(α, β)
  ○ U = rand()
  ○ Y = V_A_Gamma(n, δ)
  ○ if U ≤ f_X(Y) / C.g_Y(Y)
    ○ Set X = Y
  ○ endif
• endfunction

```

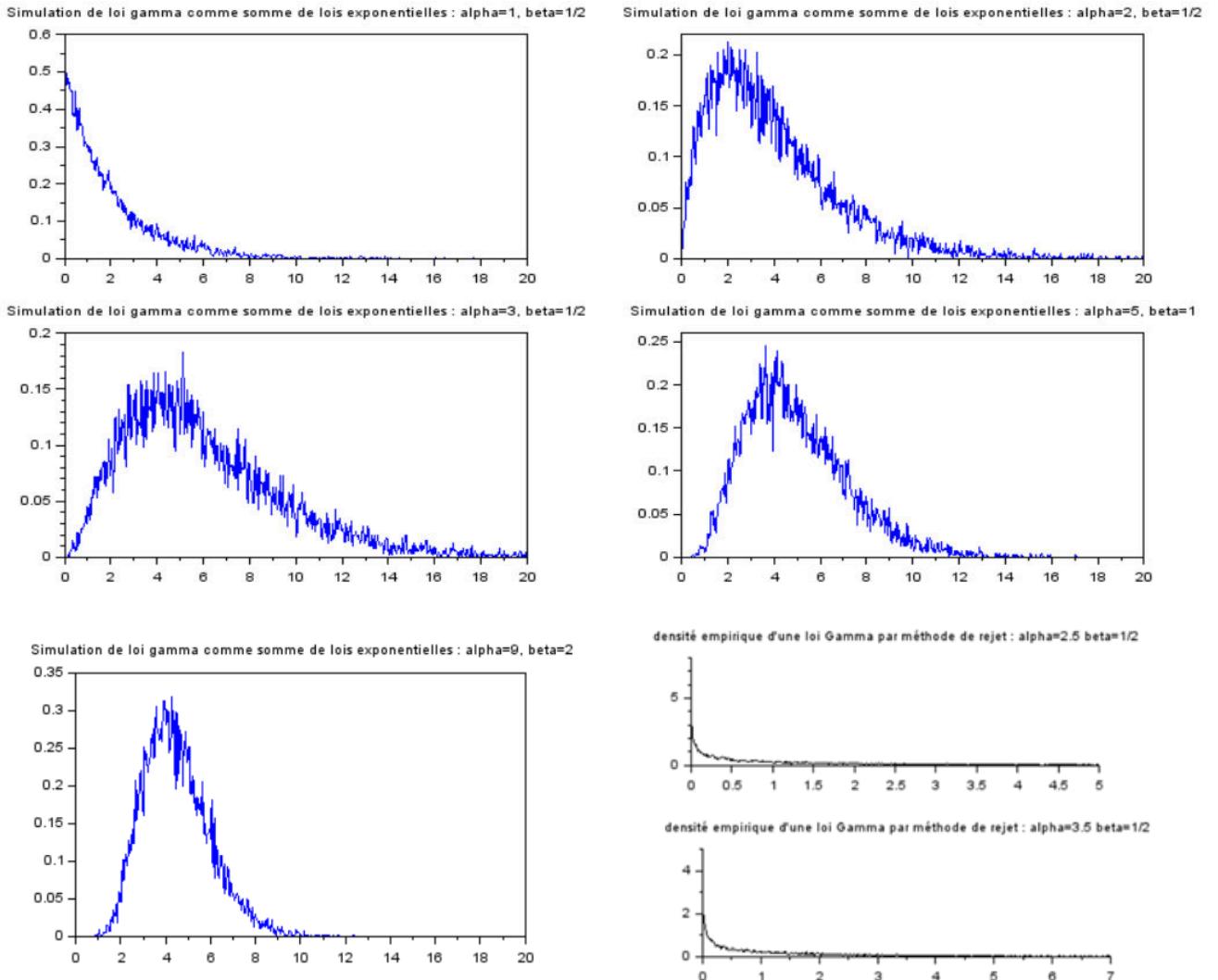
- Algorithme de fonction de densité de loi $\text{Gamma}(\alpha, \beta)$:

```

• function[f] = f_Gamma(x, α, β)
  ○ f_X(x) = β^α / Γ(α) · x^{α-1} · e^{-βx}
• endfunction

```

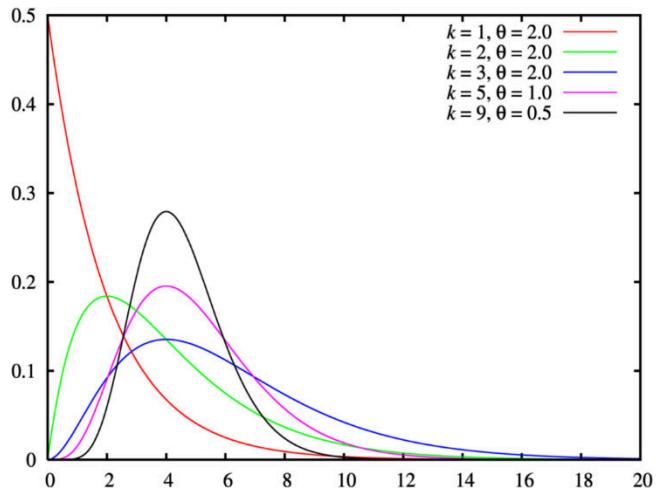
→ Les graphes de fonction de densité de Gamma : Scilab



→ Comparaison avec les graphes de Wikipédia :

On remarque que les densités empiriques simulées pour la loi Gamma ressemblent beaucoup au graphe de Wikipédia, alors on peut dire qu'on a bien réussi à simuler la loi de Gamma.

(Codes Scilab : voir annexe 2)



- Variable aléatoire Pareto. V. a. Pareto modélise des dommages causés par Covid -19.
Choisissons des jeux de données et tracez les fonction de densités.

Le principe de Pareto, aussi appelé loi de Pareto, est un phénomène empirique constaté dans certains domaines : environ 80 % des effets sont le produit de 20 % des causes.

Soit $X \sim \text{Pareto}(a, b)$ avec $a, b > 0$, sa fonction de répartition est : $\forall x > b : F_X(x) = 1 - (\frac{b}{x})^a$ bijective

Alors en utilisant la méthode d'inversion de fonction de répartition :

- Algorithme de simulation de loi Pareto(a,b) :

```

• function[X] = V_A_Pareto(a, b)
    ○ U = rand()
    ○ set X = b. (1 - u)^(-1/a)
• endfunction

```

- Algorithme de simulation d'une chaîne de v.a Pareto(a,b) :

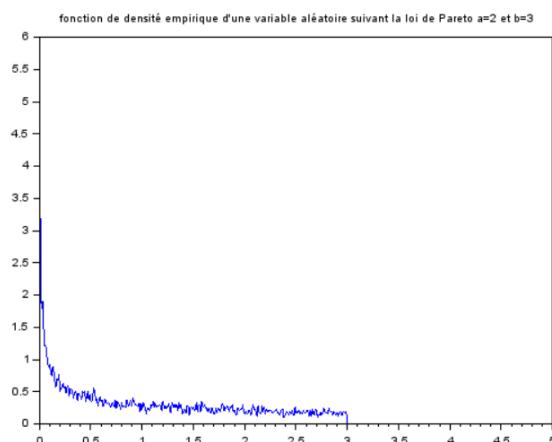
```

• function[X] = Chaine_V_A_Pareto(a, b)
    ○ for n = 1:N_mc
    ○ X(n) = V_A_Pareto(a, b)
    ○ endfor
• endfunction

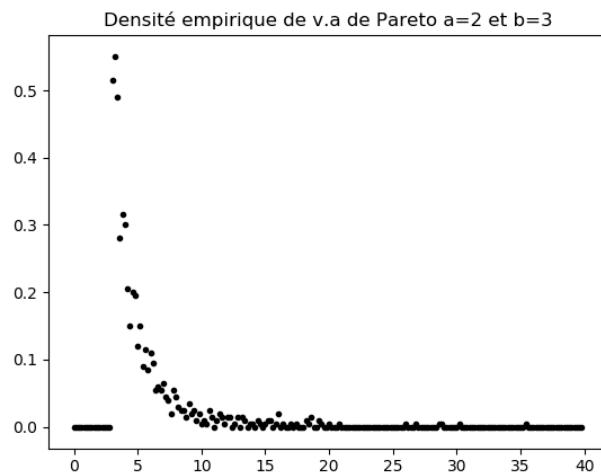
```

Soit le paramètre a de loi de Pareto : le nombre de personnes qu'un malade infecte en moyenne autour de lui ne serait pas de 2 ou 2,5 comme le suggèrent les rapports chinois relayés par l'Organisation mondiale de la santé (OMS). Soit b = 3 le taux de mort. (a=2,b=3)

→ Les graphes de fonction de densité de Pareto : Scilab



→ Les graphes de fonction de densité de Pareto : Python



(Codes Python et Scilab : voir annexe 3)

6. Livrable 2:

6.1 Activité 1. Simulation Monte-Carlo:

Dans cette activité

- Nous simulons l'évolution de la réserve de la compagnie d'assurance R_t .

Nous étudions les effets ses sinistres modélisées par v.a. Gamma (de queue fine) et de par v.a. Pareto (de queue lourde).

Réponse :

Commençons d'abord par simuler la réserve R_t à queues fines et à queues lourde :

- Algorithme de simulation de R_t à queues fines :

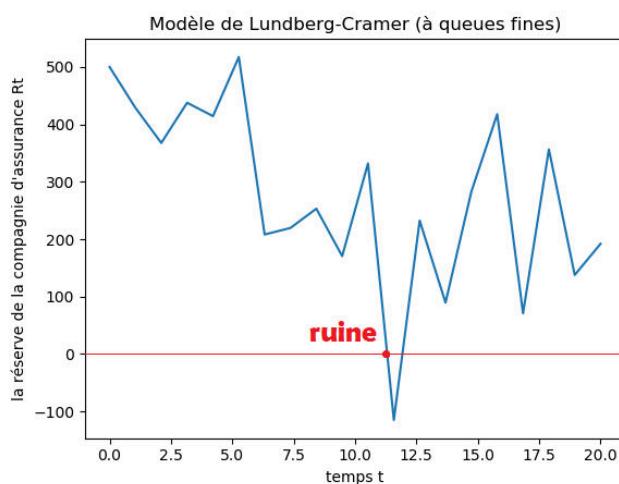
```
• function[Rt] = Rt_fines(c,x,λ,α,β,t)
    ○  $N_t = V\_A\_Poisson\_Compose(\lambda, t)$ 
    ○  $X = 0$ 
    ○ for k = 1:N_t
        ○  $Z_k = V\_A\_Gamma(\alpha, \beta)$ 
        ○  $X = X + Z_k$ 
    ○ endfor
    ○ set Rt = x + c.t - X
• endfunction
```

- Algorithme de simulation de R_t à queues lourde :

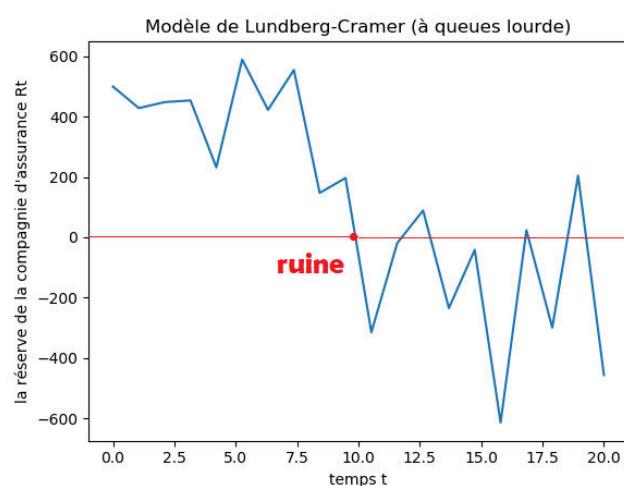
```
• function[Rt] = Rt_lourde(c,x,λ,a,b,t)
    ○  $N_t = V\_A\_Poisson\_Compose(\lambda, t)$ 
    ○  $X = 0$ 
    ○ for k = 1:N_t
        ○  $Z_k = V\_A\_Pareto(a, b)$ 
        ○  $X = X + Z_k$ 
    ○ endfor
    ○ set Rt = x + c.t - X
• endfunction
```

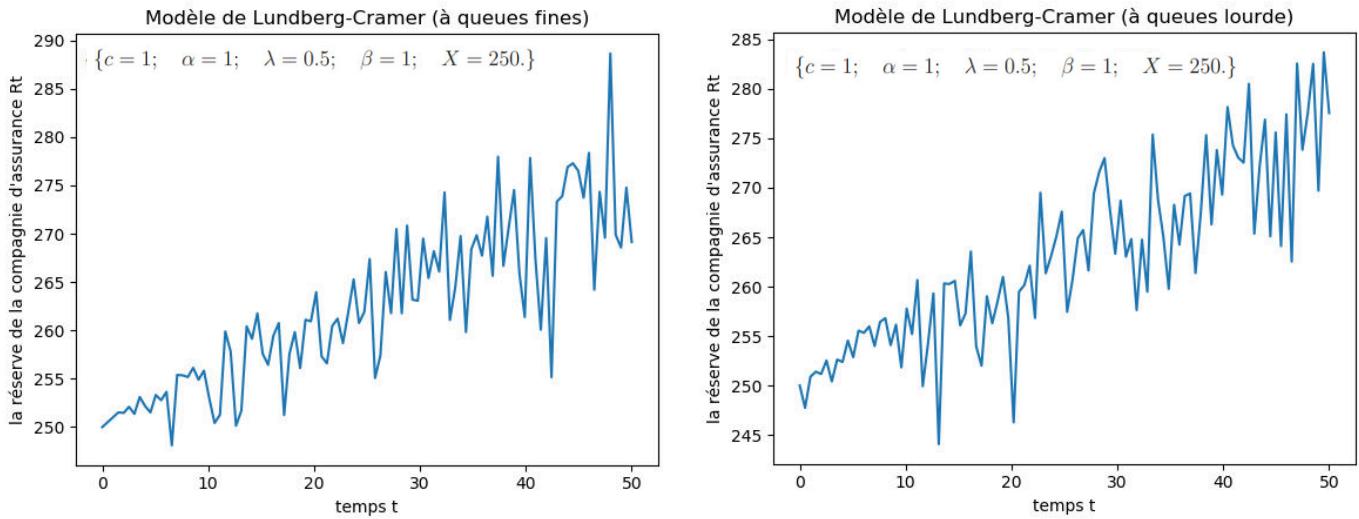
Après la simulation on obtient ces graphes :

{ $c = 100; \alpha = 8; \lambda = 6; \beta = 2.5; X = 500.$ }



{ $c = 100; \alpha = 2; \lambda = 7; b = 10; X = 500.$ }





(Codes Python : voir annexe 4)

2. Nous estimons la probabilité de ruine au bout d'un an (horizon fini) pour les deux jeux de paramètres.

Réponse :

On se propose de calculer la probabilité de ruine au bout d'un ans $T = 1$ ans (horizon fini), pour les jeux de paramètres $\{c = 100, x = 50, T = 1, \lambda = 6, \alpha = 8, \beta = 2.5\}$ pour $N_{mc} = 10^6$

Et $\{c = 1, x = 250, T = 1, \lambda = 0.5, \alpha = 1, \beta = 1\}$ pour $N_{mc} = 10^6$ c'est-à-dire le calcul de :

$$P(R_T < 0 | R(0) = x)$$

Donc il faut simuler : $P(X_T > a)$ avec $a = x + c \cdot T$

On simule N_{mc} réalisation de $\{X_1, X_2, \dots, X_{N_{mc}}\}$, on compte celles n_a qui sont supérieurs strictement à a , on calcule $P = \frac{n_a}{N_{mc}}$ et on a : $P(X_T > a) = E[1_{\{X_T > a\}}] = \frac{1}{N_{mc}} \sum_{n=1}^{N_{mc}} 1_{\{X_T(n) > a\}}$ avec $N_{mc} = 10^6$

• Algorithme de simulation de X_T à queues fines :

- **function**[X] = V_A_X(c, x, λ, T, α, β)
 - $N_T = V_A_Poisson_Compose(\lambda, T)$
 - $X = 0$
 - **for** k = 1:N_T
 - $Z_k = V_A_Gamma(\alpha, \beta)$
 - $X = X + Z_k$
 - **endfor**
 - **set** X
- **endfunction**

• Algorithme de simulation de X_T à queues lourde :

- **function**[X] = V_A_X(c, x, λ, T, α, β)
 - $N_T = V_A_Poisson_Compose(\lambda, T)$
 - $X = 0$
 - **for** k = 1:N_T
 - $Z_k = V_A_Pareto(\alpha, \beta)$
 - $X = X + Z_k$
 - **endfor**
 - **set** X
- **endfunction**

- Algorithme de simulation de Probabilité de ruine :

```

• function[Proba] = Proba_Ruine(c, x,  $\lambda$ , T,  $\alpha$ ,  $\beta$ )
    ○ Counter = 0 ,       $a = x + c \cdot T$ 
    ○ for n = 1:Nmc
        ○ X(n) = V_A_X(c, x,  $\lambda$ , T,  $\alpha$ ,  $\beta$ )
        ○ if X(n) > a
            ○ Counter = Counter + 1
        ○ endif
    ○ endfor
    ○ set Proba =  $\frac{\text{Counter}}{N_{mc}}$ 
• endfunction

```

(Codes Python : voir annexe 5)

Pour : { $c = 100$, $x = 50$, $T = 1$, $\lambda = 6$, $\alpha = 8$, $\beta = 2.5$ } pour $N_{mc} = 10^6$ on trouve :

```

>>> Proba_ruine_fines(100,50,1,6,8,2.5,1000000)
0.264787

>>> Proba_ruine_lourde(100,50,1,6,8,2.5,1000000)
0.0

```

Pour : { $c = 1$, $x = 250$, $T = 1$, $\lambda = 0.5$, $\alpha = 1$, $\beta = 1$ } pour $N_{mc} = 10^6$ on trouve :

```

>>> Proba_ruine_fines(1,250,1,0.5,1,1,1000000)
0.0

>>> Proba_ruine_lourde(1,250,1,0.5,1,1,1000000)
0.0

```

Ces résultats sont cohérents puisque la probabilité de ruine est parfois 10^{-30} ce qui traduit la valeur 0 dans certains calculs, mais on a réussi à avoir une probabilité de ruine 20% pour les premiers jeux de valeurs à queues fines.

3. Nous estimons la probabilité de ruine et le temps de la ruine (horizon infini) pour des deux jeux de paramètres que nous choisissons nous-même.

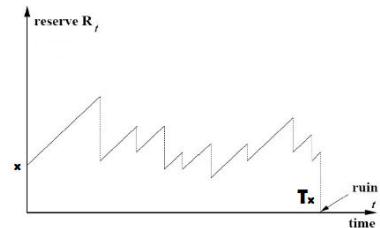
Réponse :

La probabilité de ruine ultime ou probabilité de ruine à horizon de temps infini, notée ψ , est définie par :

$$\psi(x) = P(\inf_{t \geq 0} R(t) < 0 \mid R(0) = x)$$

Et l'instant de la ruine est : $T_x = \inf\{t > 0, R(t) < 0\}$.

On représente l'évolution de la réserve $R(t)$ en fonction du temps t :



Il faut donc récupérer l'instant de la ruine T_x puis calculer la probabilité de la ruine à cet instant (remplacer $T = 1$ ans de la question 2 Par $T = T_x$) :

- Algorithme de simulation de T_x à queues fines :

```

• function[t] = Tx_fines(c,x, $\lambda$ , $\alpha$ , $\beta$ , $\Delta$ )
    ○ Rt = Rt_fines(c,x, $\lambda$ , $\alpha$ , $\beta$ ,t)
    ○ t = 0
    ○ while Rt > 0
        ○ t = t +  $\Delta$ 
    ○ endwhile
    ○ set t
    • endfunction

```

- Algorithme de simulation de T_x à queues lourde :

```

• function[t] = Tx_lourde(c,x, $\lambda$ , $\alpha$ , $\beta$ , $\Delta$ )
    ○ Rt = Rt_lourde(c,x, $\lambda$ , $\alpha$ , $\beta$ ,t)
    ○ t = 0
    ○ while Rt > 0
        ○ t = t +  $\Delta$ 
    ○ endwhile
    ○ set t
    • endfunction

```

(Codes Python : voir annexe 6)

Pour les jeux de paramètres { $c = 100$, $x = 50$, $\lambda = 6$, $\alpha = 8$, $\beta = 2.5$ } on obtient avec $\Delta = 0.01$:

```

>>> Tx=Tx_fines(50,100,6,8,2.5,0.01)
>>> Tx
0.3200000000000001

```

Et la probabilité de ruine pour l'instant T_x pour $N_{mc} = 10^6$ on trouve :

```

>>> Tx=Tx_fines(50,100,6,8,2.5,0.01)
>>> Tx
0.3200000000000001
>>> Proba_ruine_fines(50,100,Tx,6,8,2.5,1000000)
0.015125

```

4. Nous étudions la méthode de changement de l'espace de probabilité et la transformation d'Esscher.

Réponse :

On remarque que cette probabilité de ruine est très petite alors on obtient avec la simulation $P(X_T > a) = 0$ ce qui est tout à fait normal car parfois $P(X_T > a) \approx 10^{-30}$

➔ **Solution :** Pour pouvoir calculer cette probabilité par Monte-Carlo on génère dans l'espace (Ω, F_T, Q) de probabilité les sinistres avec l'intensité λ^Q beaucoup plus élevé que λ ce qui permet de compter $1_{\{X_T^Q > a\}}$. Cependant pour compenser cette valeur élevée on ajoute un coefficient de passage L_T dont la valeur est très petite :

$$P(X_T > a) = E_P[1_{\{X_T > a\}}] = E_Q \left[1_{\{X_T^Q > a\}} \cdot L_T \right] = E_Q \left[1_{\{X_T^Q > a\}} \cdot e^{-\theta \cdot X_T^Q + \Gamma(\theta)} \right]$$

Par Monte-Carlo :

$$P(X_T > a) = \frac{1}{N_{mc}} \sum_{n=1}^{N_{mc}} 1_{\{X_T^Q(n) > a\}} \cdot e^{-\theta \cdot X_T^Q(n) + \Gamma(\theta)}$$

D'après l'étude mathématique du livrable 2 on a :

$$Z_k^Q \sim Gamma(\alpha, \beta - \theta), \quad \theta = \beta - \beta \left(\frac{\beta \cdot (x + c \cdot T)}{\alpha \lambda T} \right)^{\frac{-1}{\alpha+1}}, \quad \Gamma(\theta) = \lambda T \left(\left(\frac{\beta}{\beta - \theta} \right)^\alpha - 1 \right) \text{ et } \lambda^Q = \lambda \cdot \left(\frac{\beta}{\beta - \theta} \right)^\alpha$$

5. Nous simulons une ruine rare

Réponse :

On veut simuler la probabilité de ruine d'une ruine rare pour les jeux de paramètres :

{ $c = 100$, $x = 50$, $T = 1$, $\lambda = 7$, $\alpha = 4$, $\beta = 1.5$ } Pour $N_{mc} = 10^2$, normalement si on calcule cette probabilité avec la simulation de Monte-Carlo naïf on obtient une probabilité nulle :

```
>>> Proba_ruine_fines(100,50,1,7,4,1.5,100)
0.0
```

Mais avec la transformation d'Esscher on va obtenir une valeur non nulle. Donc on va simuler la probabilité de la ruine part Importance Simpling :

• Algorithme de simulation de X_T^Q :

- **function**[X] = *V_A_X_Q*($c, x, \lambda, T, \alpha, \beta$)
 - $\theta = \beta - \beta \left(\frac{\beta \cdot (x + c \cdot T)}{\alpha \lambda T} \right)^{\frac{-1}{\alpha+1}}$
 - $\lambda^Q = \lambda \cdot \left(\frac{\beta}{\beta - \theta} \right)^\alpha$
 - $N_T = V_A_Poisson_Compose(\lambda^Q, T)$
 - $X = 0$
 - **for** $k = 1 : N_T$
 - $Z_k^Q = V_A_Gamma(\alpha, \beta - \theta)$
 - $X = X + Z_k^Q$
 - **endfor**
 - **set** X
- **endfunction**

• Algorithme de simulation de Probabilité de ruine Q:

- **function**[$Proba$] = *Proba_Ruine_Q*($c, x, \lambda, T, \alpha, \beta$)
 - $\theta = \beta - \beta \left(\frac{\beta \cdot (x + c \cdot T)}{\alpha \lambda T} \right)^{\frac{-1}{\alpha+1}}$
 - $\Gamma(\theta) = \lambda T \left(\left(\frac{\beta}{\beta - \theta} \right)^\alpha - 1 \right)$
 - **Counter** = 0, $a = x + c \cdot T$
 - **for** $n = 1 : N_{mc}$
 - $X^Q(n) = V_A_X_Q(c, x, \lambda, T, \alpha, \beta)$
 - **if** $X^Q(n) > a$
 - **Counter** = **Counter** + $e^{-\theta \cdot X^Q(n) + \Gamma(\theta)}$
 - **endif**
 - **endfor**
 - **set** $Proba = \frac{\text{Counter}}{N_{mc}}$
 - **endfunc** **on**

(Codes Python : voir annexe 7)

On applique cette méthode pour les jeux de paramètres précédents dont on a trouvé que la probabilité de ruine est nulle par la méthode Monte-Carlo naïf et on trouve :

```
>>> Proba_ruine_Q(100,50,1,7,4,1.5,100)
1.454670783095162e-22
```

Donc on a trouvé que $P(X_T > a) = 1,45 \cdot 10^{-22}$ ce qui est impossible d'obtenir avec Monte-Carlo naïf.

6. Finalement nous étudions le mécanisme de fonctionnement de compagnie diversifiée.

Réponse :

On suppose que la compagnie étudiée propose des assurances habitations et des assurances automobiles, et qu'une partie des sinistres liés aux risques naturels (Covid-19) N_t^{RN} touchent à la fois le secteur automobile et habitation. On ajoute des sinistres propres aux habitations N_t^H et des sinistres propres aux voitures N_t^A :

$$R_1(t) = x + c \cdot t - \sum_{k=1}^{N_t^{RN}} Z_k^1 - \sum_{k=1}^{N_t^H} Z_k^2 \quad \text{et} \quad R_2(t) = x + c \cdot t - \sum_{k=1}^{N_t^{RN}} Z_k^1 - \sum_{k=1}^{N_t^A} Z_k^3$$

Et

$$R_3(t) = x + c \cdot t - \sum_{k=1}^{N_t^{RN}} Z_k^1 - \sum_{k=1}^{N_t^H} Z_k^2 - \sum_{k=1}^{N_t^A} Z_k^3$$

Avec :

- ⇒ R_1 : La réserve de la branche d'assurance d'habitation liée aux risques naturels (Covid-19)
- ⇒ R_2 : La réserve de la branche du secteur automobile liée aux risques naturels (Covid-19)
- ⇒ R_3 : La réserve des deux branches liées aux risques naturels (Covid-19)

Alors on peut simuler et calculer la probabilité de la ruine de chaque branche et les effets de compensation entre branche, comme on peut déterminer le capital initial dont doit disposer la compagnie pour que sa probabilité de ruine à horizon fini soit inférieure à un seuil et aussi de représenter la distribution de la réserve sachant qu'il y a eu ruine (**on va voir cette étude dans la partie travail à faire partie : Compagnie diversifiée**).

6.2 Activité 2. Etudes mathématiques:

Montrons que dans l'espace de probabilité (Ω, F_T, Q) sous laquelle $X_t^Q = \sum_{k=1}^{N_t^Q} Z_k^Q$ est un processus de Poisson composé de caractéristiques suivantes :

- Son intensité est : $\lambda^Q = \lambda \cdot E[e^{\theta Z_1}] = \lambda \cdot (\frac{\beta}{\beta - \theta})^\alpha, \quad \theta \in R_+^*$

- la fonction de densité de Z_k^Q : $f_G^Q(y) = f_G(y) \cdot \frac{e^{y\theta}}{E_p[e^{\theta Z_1}]} = \frac{1}{\Gamma(\alpha)} (\beta - \theta)^\alpha y^{\alpha-1} e^{-(\beta-\theta)y}$

C'est-à-dire que la loi de $f_G^Q(y)$ est la loi Gamma de paramètre $(\alpha, \beta - \theta)$.

- La dérivée Radon Nikodym : $\widehat{L}_T = \frac{dQ}{dP} = e^{\theta X_T - \Gamma(\theta)}$

- Le coefficient $\Gamma(\theta)$: $\Gamma(\theta) = \ln(E_p[e^{\theta Z_1}]) = \lambda T \left(\left(\frac{\beta}{\beta - \theta} \right)^\alpha - 1 \right)$

- Le paramètre optimale : $\theta = \beta - \beta \left(\frac{\beta \cdot (x + c \cdot T)}{\alpha \lambda T} \right)^{\frac{-1}{\alpha+1}}$

Réponse :

Soit la richesse $R(t) = x + c \cdot t - \sum_{k=1}^{N_t} Z_k = a - X(t)$ avec $a = x + c \cdot t$ et $X(t) = \sum_{k=1}^{N_t} Z_k$ le processus de Poisson composé, N_t suit la loi de Poisson d'intensité λ et Z_k suit la loi de Gamma(α, β) et sa fonction de densité $f_G(y) = \frac{1}{\Gamma(\alpha)} \beta^\alpha y^{\alpha-1} e^{-\beta y} \cdot 1_{y>0}$.

On calcule la fonction caractéristique de X_T dans l'espace probabilité (Ω, F_T, P) et de X_t^Q dans l'espace probabilité (Ω, F_T, Q) et on fait les identifications :

On a :

$$\Phi_{X(T)}(u) = E_p[e^{iuX(T)}] = \sum_{k=0}^{+\infty} E_p[e^{iu \sum_{j=1}^{N_T} Z_j} | \{N_T = k\}] \mathbb{P}(N_T = k)$$

par indépendance de N_T et $(Z_j)_j$ qui sont indépendants et identiquement distribués:

$$\begin{aligned} \Phi_{X(T)}(u) &= \sum_{k=0}^{+\infty} E_p[e^{iu \sum_{j=1}^k Z_j}] \mathbb{P}(N_T = k) = \sum_{k=0}^{+\infty} \prod_{j=1}^k E_p[e^{iu Z_j}] \mathbb{P}(N_T = k) = \sum_{k=0}^{+\infty} (E_p[e^{iu Z_j}])^k \frac{(\lambda \cdot T)^k}{k!} e^{-\lambda \cdot T} \\ &= e^{\lambda \cdot T (E_p[e^{iu Z_j}] - 1)} = e^{\lambda \cdot T (\int_{\mathbb{R}} e^{iu y} f_G(y) dy - 1)} = e^{\lambda \cdot T (\int_{\mathbb{R}} (e^{iu y} - 1) f_G(y) dy)} \end{aligned}$$

puisque: $\int_{\mathbb{R}} f_G(y) dy = 1$. Donc: $[\Phi_{X(T)}(u) = e^{\lambda \cdot T \int_{\mathbb{R}} (e^{iu y} - 1) f_G(y) dy}] \quad (*)$

D'autre part:

$$\Phi_{X(T)}^Q(u) = E_Q[e^{iu X(T)}] = E_p[e^{iu X(T)} \frac{dQ}{dP}] = E_p[e^{iu X(T)} \hat{\mathbb{L}}_T] = E_p[e^{iu X(T)} \cdot e^{\theta \cdot X(T) - \Gamma(\theta)}]$$

or: il est évident par construction que $E_p[\hat{\mathbb{L}}_T] = 1$ où $\hat{\mathbb{L}}_T = e^{\theta \cdot X_T - \Gamma(\theta)}$

donc: $E_p[\hat{\mathbb{L}}_T] = E_p[e^{\theta \cdot X_T - \Gamma(\theta)}] = E_p[e^{\theta \cdot X_T}] \cdot e^{-\Gamma(\theta)} \Rightarrow e^{\Gamma(\theta)} = E[e^{\theta \cdot X(T)}]$

Donc: $\boxed{\Gamma(\theta) = \ln(\mathbb{E}[e^{\theta \cdot X_{(T)}}])}$ d'après la question (4) du livrable (1), on a:

$$\mathbb{E}_p[e^{\theta \cdot X_{(t)}}] = e^{\lambda \cdot t(\mathbb{E}[e^{\theta \cdot Z_1}] - 1)} = e^{\lambda \cdot t((\frac{\beta}{\beta-\theta})^\alpha - 1)} \text{ et par suite:}$$

$$\boxed{\Gamma(\theta) = \lambda \cdot T((\frac{\beta}{\beta-\theta})^\alpha - 1)} \text{ pour } t=T.$$

$$\text{Et donc: } \Phi_{X_{(T)}^Q}(u) = \mathbb{E}_p[e^{(iu+\theta)X_{(T)}}] e^{\lambda \cdot T(\mathbb{E}[e^{\theta \cdot Z_1}] - 1)}$$

$$\text{d'après la question (4) du livrable (1): } \mathbb{E}_p[e^{(iu+\theta)X_{(t)}}] = e^{\lambda \cdot t(\mathbb{E}[e^{(iu+\theta)Z_1}] - 1)} \text{ donc pour } t=T$$

$$\Phi_{X_{(T)}^Q}(u) = e^{\lambda \cdot T(\mathbb{E}_p[e^{(iu+\theta)Z_1}] - 1)} \cdot e^{\lambda \cdot T(\mathbb{E}_p[e^{\theta \cdot Z_1}] - 1)} = e^{\lambda \cdot T(\mathbb{E}_p[e^{(iu+\theta)Z_1}] - \mathbb{E}_p[e^{\theta \cdot Z_1}])}$$

par linéarité de l'espérance:

$$\Phi_{X_{(T)}^Q}(u) = e^{\lambda \cdot T(\mathbb{E}_p[e^{(iu+\theta)Z_1}] - e^{\theta \cdot Z_1})} = e^{\lambda \cdot T(\mathbb{E}_p[e^{\theta \cdot Z_1}(e^{iuZ_1} - 1)])} = e^{\lambda \cdot T \int_{\mathbb{R}} (e^{iu y} - 1) e^{\theta \cdot y} f_G(y) dy}$$

$$\text{Donc: } \boxed{\Phi_{X_{(T)}^Q}(u) = e^{\lambda \cdot T \int_{\mathbb{R}} (e^{iu y} - 1) e^{\theta \cdot y} f_G(y) dy}} \quad] \quad (**)$$

Par identification de (*) et (**) on trouve:

$$\boxed{\lambda^Q = \lambda \cdot \int_{\mathbb{R}} e^{\theta \cdot y} f_G(y) dy = \lambda \mathbb{E}_p[e^{\theta \cdot Z_1}]}$$

$$\text{et: } \boxed{f_G^Q(y) = \frac{e^{\theta \cdot y} f_G(y)}{\int_{\mathbb{R}} e^{\theta \cdot y} f_G(y) dy} = \frac{e^{\theta \cdot y}}{\mathbb{E}_p[e^{\theta \cdot Z_1}]} \cdot f_G(y)}$$

$$\text{or } \mathbb{E}_p[e^{\theta \cdot Z_1}] = \int_{\mathbb{R}} e^{\theta \cdot y} f_G(y) dy = \int_{\mathbb{R}} e^{\theta \cdot y} \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} e^{-\beta \cdot y} \mathbf{1}_{[0, +\infty[}(y) = \int_{\mathbb{R}} \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} e^{-(\beta-\theta)y} dy$$

$$\text{par un changement de variable } u = (\beta - \theta)y \Rightarrow du = (\beta - \theta)dy \Rightarrow dy = \frac{du}{(\beta-\theta)}$$

$$\mathbb{E}_p[e^{\theta \cdot Z_1}] = \int_0^{+\infty} (\frac{\beta}{\beta-\theta})^\alpha \frac{u^\alpha}{\Gamma(\alpha)} e^{-uy} y^{\alpha-1} dy = (\frac{\beta}{\beta-\theta})^\alpha \frac{\Gamma(\alpha)}{\Gamma(\alpha)} = (\frac{\beta}{\beta-\theta})^\alpha$$

$$\text{Donc: } \forall \theta \in \mathbb{R}_*: \boxed{\lambda^Q = \lambda(\frac{\beta}{\beta-\theta})^\alpha} \text{ et } \boxed{f_G^Q(y) = \frac{1}{\Gamma(\alpha)} (\beta - \theta)^\alpha y^{\alpha-1} e^{-(\beta-\theta)y} \mathbf{1}_{[0, +\infty[}(y)}$$

D'après ce qui précède on a trouvé que: $\Gamma(\theta) = \lambda \cdot T((\frac{\beta}{\beta-\theta})^\alpha - 1)$

On sait que θ est la racine de $\Gamma'(\theta) = a$ avec $a = x + cT$

$$\text{On a: } \Gamma(\theta)' = (\lambda \cdot T((\frac{\beta}{\beta-\theta})^\alpha - 1))' = \frac{\alpha \cdot \lambda \cdot T \beta^\alpha}{(\beta-\theta)^{\alpha+1}}$$

$$\text{donc: } \Gamma(\theta)' = a \Leftrightarrow \frac{\alpha \cdot \lambda \cdot T \beta^\alpha}{(\beta-\theta)^{\alpha+1}} = x + cT \Leftrightarrow \frac{\alpha \cdot \lambda \cdot T \beta^\alpha}{x+cT} = (\beta - \theta)^{\alpha+1}$$

$$\Leftrightarrow \beta - \theta = (\frac{\alpha \cdot \lambda \cdot T \beta^\alpha}{x+cT})^{\frac{1}{\alpha+1}} = \beta (\frac{\alpha \cdot \lambda \cdot T}{\beta(x+cT)})^{\frac{1}{\alpha+1}}$$

$$\Leftrightarrow \theta = \beta - \beta \cdot (\frac{\alpha \cdot \lambda \cdot T}{\beta(x+cT)})^{\frac{1}{\alpha+1}}$$

$$\text{Donc: } \boxed{\theta = \beta - \beta \cdot (\frac{\alpha \cdot \lambda \cdot T}{\beta(x+cT)})^{\frac{1}{\alpha+1}}}$$

7. Travail à faire:

7.1 Risque de ruine à l'horizon fini

L'objectif de cette partie est de calculer la probabilité de ruine $P(R_t < 0)$ pour différentes valeurs des paramètres du modèle $\{x, c, \lambda, \alpha, \beta\}$ et pour $T = 1$ an

1. Estimons la probabilité de ruine au bout d'un an pour les deux jeux de paramètres suivants :

Pour $\{x = 50, c = 14, \lambda = \frac{1}{24}, \alpha = 3, \beta = 0.04\}$ et $\{x = 50, c = 14, \lambda = \frac{1}{24}, \alpha = 3, \beta = 0.04\}$

On trouve avec $N_{mc} = 10^3$:

```
>>> Proba_ruine_Q(14,50,1,2,5,0.75,1000)
0.0
>>> Proba_ruine_Q(14,250,1,2,5,0.75,1000)
0.0
```

Car la valeur de λ et β sont très petites, alors si on augmente les valeurs de λ, β et α :

Pour $\{x = 50, c = 14, \lambda = 2, \alpha = 5, \beta = 0.75\}$ on trouve avec $N_{mc} = 10^3$:

```
>>> Proba_ruine_Q(14,50,1,2,5,0.75,1000)
3.441841998980371e-06
```

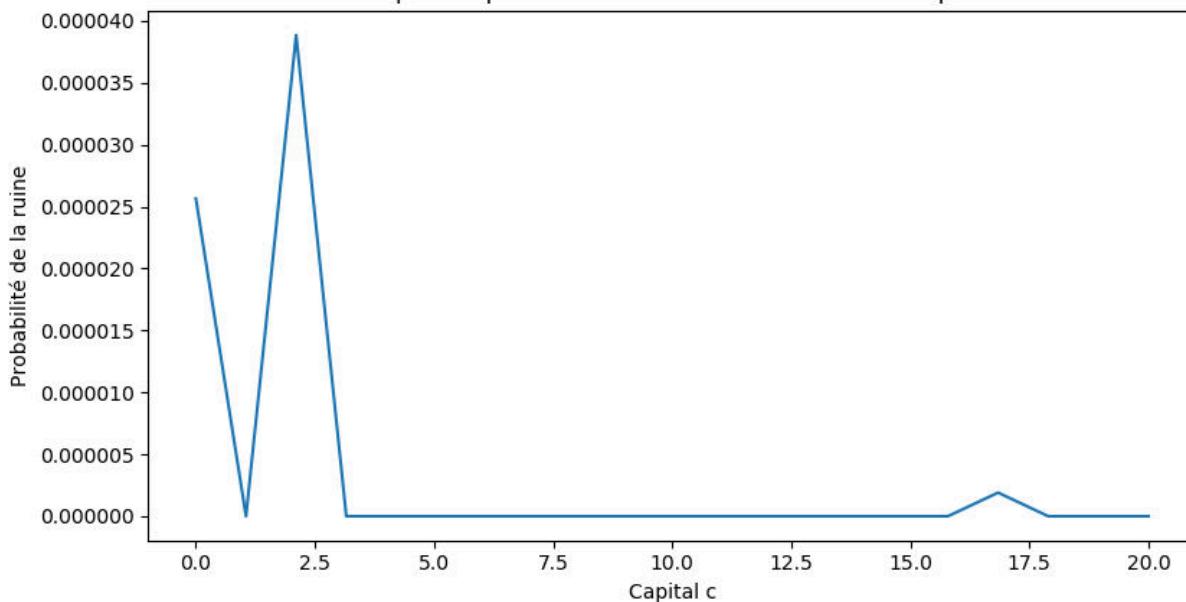
Alors $P(R_t < 0) = 3,44 \cdot 10^{-6}$

2. Construire le graphe qui affiche l'évolution de la probabilité de ruine en fonction du capital de départ.

Pour $\{x = 50, T = 1, \lambda = 2, \alpha = 5, \beta = 0.75\}$ on trouve avec $N_{mc} = 10^3$:

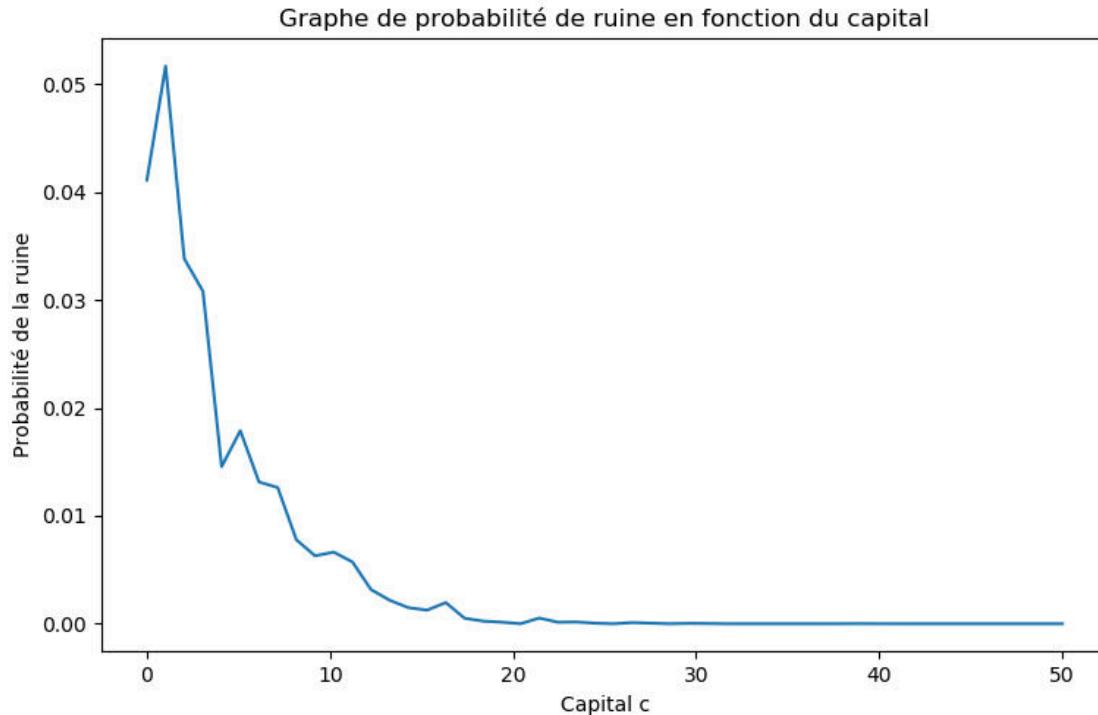
```
>>> grapheProba_ruine_Q(50,1,2,5,0.75,1000)
```

Graphe de probabilité de ruine en fonction du capital



Pour $\{x = 20, T = 1, \lambda = 2, \alpha = 5, \beta = 0.75\}$ on trouve avec $N_{mc} = 10^3$:

```
>>> grapheProba_ruine_Q(20,1,2,5,0.75,1000)
```

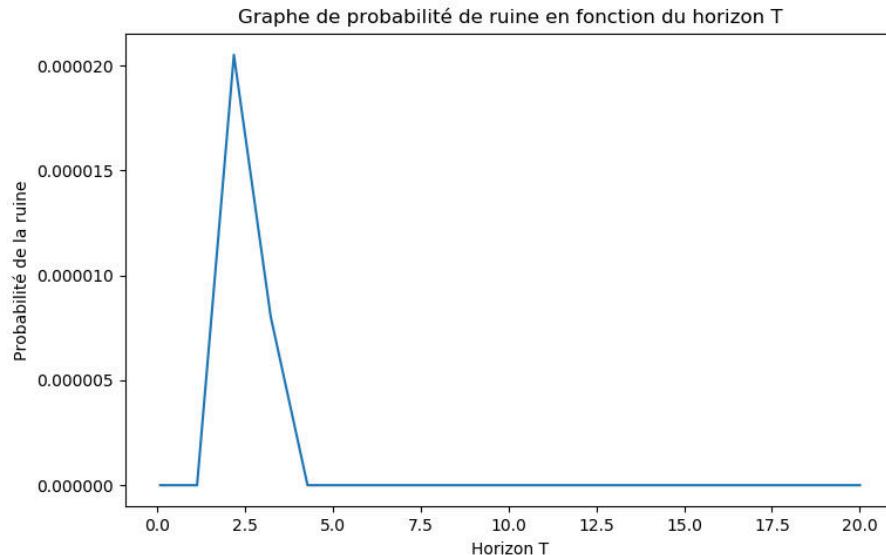


(Codes Python : voir annexe 8)

3. Construire le graphe qui affiche l'évolution de la probabilité de ruine en fonction du l'horizon T

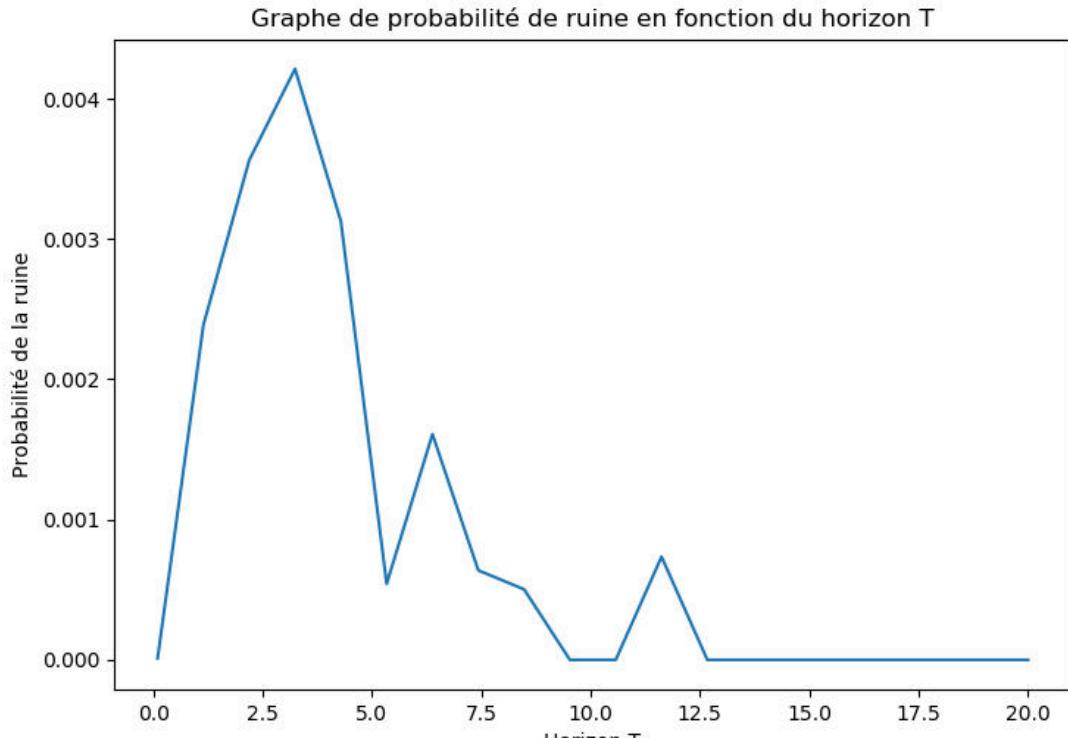
Pour $\{x = 50, c = 14, \lambda = 2, \alpha = 5, \beta = 0.75\}$ on trouve avec $N_{mc} = 10^3$:

```
>>> grapheProba_ruine_Q_T(14,50,2,5,0.75,1000)
```



Pour $\{x = 20, c = 14, \lambda = 2, \alpha = 5, \beta = 0.75\}$ on trouve avec $N_{mc} = 10^3$:

```
>>> grapheProba_ruine_Q_T(14,20,2,5,0.75,1000)
```



(Codes Python : voir annexe 9)

7.2 Risque de ruine à l'horizon infini :

Pour $\{x = 50, c = 14, \lambda = \frac{1}{24}, \alpha = 3, \beta = 0.04\}$ et $\{x = 50, c = 14, \lambda = \frac{1}{24}, \alpha = 3, \beta = 0.04\}$

On n'a pas pu trouver le temps de ruine car la valeur de λ et β sont très petites, alors si on augmente les valeurs de λ, β et α :

Pour $\{x = 50, c = 14, \lambda = 2, \alpha = 5, \beta = 1.5\}$ et $\{x = 20, c = 14, \lambda = 2, \alpha = 5, \beta = 1.5\}$

On trouve avec $\Delta = 0.1$ ans :

```
>>> Tx_fines(50,14,2,5,1.5,0.1)
2.6000000000000001
```

```
>>> Tx_fines(20,14,2,5,1.5,0.1)
1.5000000000000002
```

On trouve $T_{x=50} = 2.6$ ans et $T_{x=20} = 1.5$ ans

7.3 Risque de ruine à l'horizon 1 an est un évènement rare :

1. Simuler un évènement rare $P \approx 10^{-17}$ à l'aide de transformation d'Esscher :

Pour $\{x = 50, c = 100, \lambda = 6, \alpha = 8, \beta = 1.75\}$ on trouve avec $N_{mc} = 10^2$:

$$P = 3,12 \cdot 10^{-17} \approx 10^{-17}$$

```
>>> Proba_ruine_Q(100,50,1,6,8,1.75,100)
3.123793544942571e-17
```

2. Estimer la probabilité de ruine au bout d'un an pour les deux jeux de paramètres suivants :

Pour $\{x = 250, c = 14, \lambda = 2, \alpha = 2.5, \beta = 0.5\}$ et $\{x = 250, c = 1, \lambda = 0.5, \alpha = 1, \beta = 1\}$

On obtient une probabilité de ruine nulle car la valeur de λ et β sont très petites, alors si on augmente les valeurs de λ, β et α :

Pour $\{x = 50, c = 14, \lambda = 2, \alpha = 5, \beta = 0.75\}$ on trouve avec $N_{mc} = 10^3$:

```
>>> Proba_ruine_Q(14,50,1,2,5,0.75,1000)
3.441841998980371e-06
```

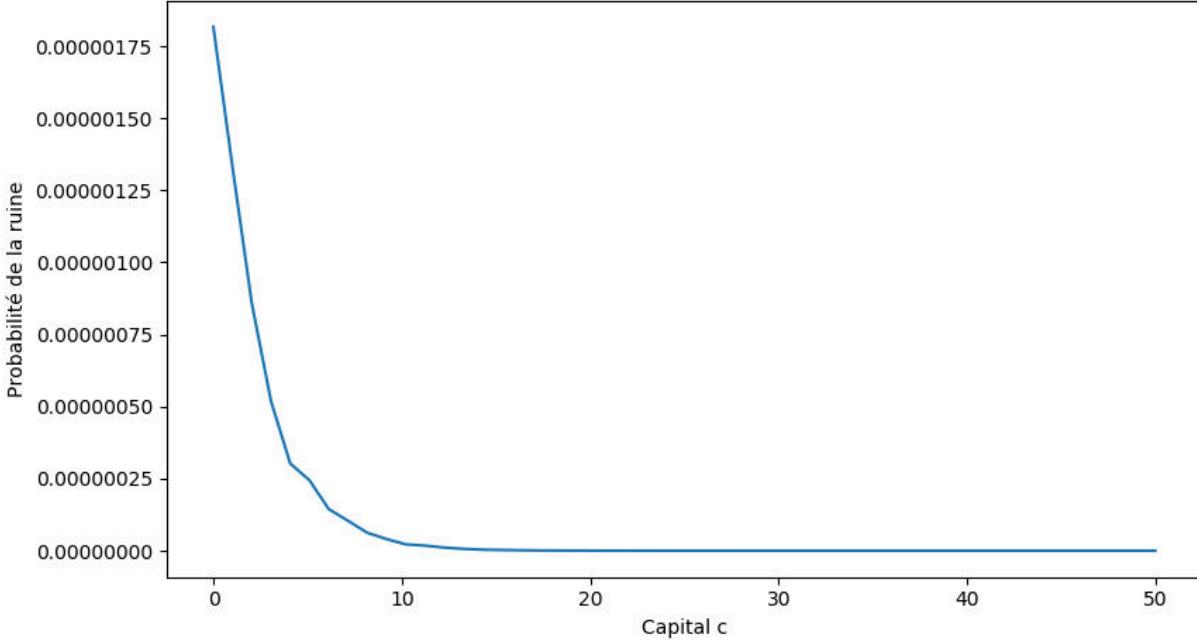
Alors $P(R_t < 0) = 3,44 \cdot 10^{-6}$

3. Déterminer le capital initial que la compagnie qui doit en avoir pour que sa probabilité de ruine à l'horizon annuel soit inférieure à $10^{-4}, 10^{-6}$:

Pour $\{x = 50, T = 1, \lambda = 4, \alpha = 4, \beta = 1.5\}$ on trouve avec $N_{mc} = 10^3$:

```
>>> grapheProba_ruine_Q(50,1,4,4,1.5,1000)
```

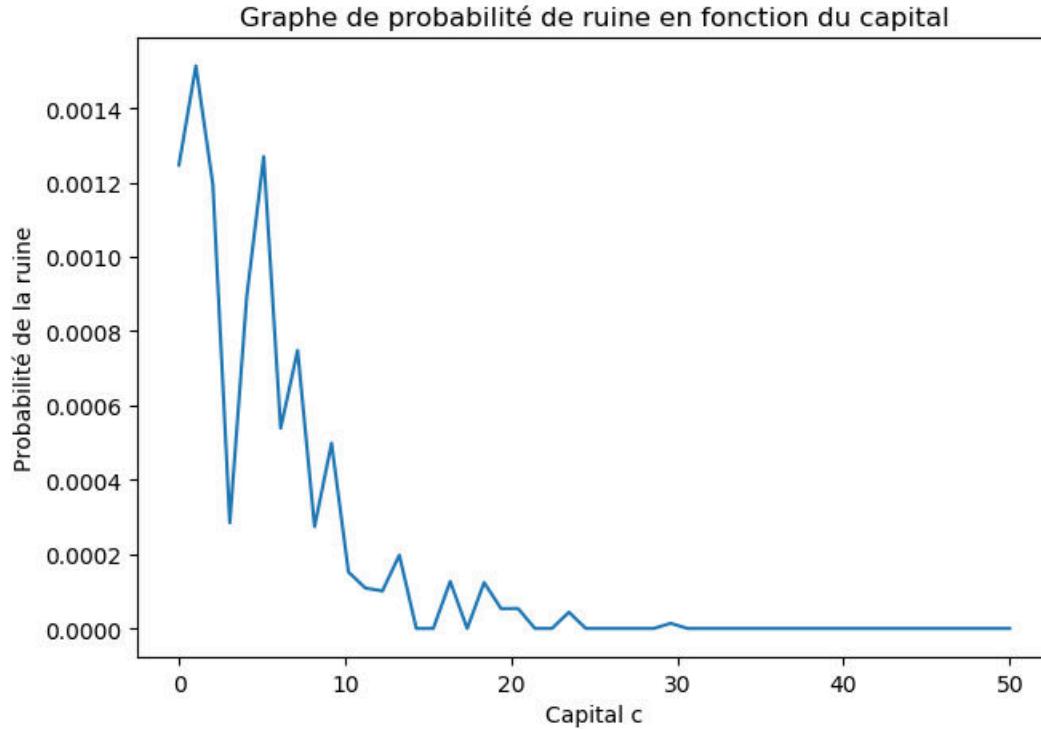
Graphe de probabilité de ruine en fonction du capital



Ainsi pour $c = 1$ on a :
`>>> Proba_ruine_Q(1,50,1,4,4,1.5,1000)` c'est-à-dire $P \approx 10^{-6}$
`1.26278376258069e-06`

Pour $\{x = 50, T = 1, \lambda = 4, \alpha = 5, \beta = 0.75\}$ on trouve avec $N_{mc} = 10^3$:

`>>> grapheProba_ruine_Q(50,1,4,5,0.75,1000)`



Ainsi pour $c = 8$ on a :
`>>> Proba_ruine_Q(8,50,1,4,5,0.75,1000)` c'est-à-dire $P \approx 10^{-4}$
`0.00011665468719913458`

7.4 Compagnie diversifiée :

On suppose que la compagnie étudiée propose des assurances habitations et des assurances automobiles, et qu'une partie des sinistres liés aux risques naturels (Covid-19) N_t^{RN} touchent à la fois le secteur automobile et habitation. On ajoute des sinistres propres aux habitations N_t^H et des sinistres propres aux voitures N_t^A :

$$R_1(t) = x + c.t - \sum_{k=1}^{N_t^{RN}} Z_k^1 - \sum_{k=1}^{N_t^H} Z_k^2 \quad \text{et} \quad R_2(t) = x + c.t - \sum_{k=1}^{N_t^{RN}} Z_k^1 - \sum_{k=1}^{N_t^A} Z_k^3$$

Et

$$R_3(t) = x + c \cdot t - \sum_{k=1}^{N_t^{RN}} Z_k^1 - \sum_{k=1}^{N_t^H} Z_k^2 - \sum_{k=1}^{N_t^A} Z_k^3$$

Avec :

- ⇒ R_1 : La réserve de la branche d'assurance d'habitation liée aux risques naturels (Covid-19)
- ⇒ R_2 : La réserve de la branche du secteur automobile liée aux risques naturels (Covid-19)
- ⇒ R_3 : La réserve des deux branches liées aux risques naturels (Covid-19)

8. Ce que nous avons réussi :

Dans cette partie, nous allons énumérer les différentes fonctionnalités que nous avons réussi à implémenter dans notre projet. La liste est la suivante :

- ✓ Se familiariser avec des notions mathématiques de l'assurance et ave des bases de la théorie de la ruine : le modèle de Lundberg-Cramer.
- ✓ Montrer théoriquement que le processus de comptage N_t suit la loi de Poisson de paramètre λt .
- ✓ Calculer l'espérance $E[X(t)]$ avec $X(t) = \sum_{k=1}^{N_t} Z_k$
- ✓ Calculer une espérance très utile : la fonction génératrice des moments de $X(t)$:
$$M_{X(t)}(\theta) = E[e^{\theta X(t)}]$$
- ✓ Simuler par Monte-Carlo le processus de comptage N_t par deux façons différentes et de tracer les graphes de fonction de densité de Poisson.
- ✓ Simuler par Monte-Carlo la loi $Gamma(\alpha = n, \beta)$ avec $n \in N$ et simuler par la méthode de rejet la loi $Gamma(\alpha, \beta)$ avec $\alpha > 0$ puis tracez les graphes de fonction de densité de Gamma pour les différents paramètres et les comparer avec le graphe de Wikipédia.
- ✓ Simuler par Monte-Carlo la loi Pareto(a,b) et tracez les graphes de fonction de densité de Pareto pour les différents paramètres afin de modéliser des dommages causés par COVID-19
- ✓ Estimer la probabilité de ruine au bout d'un an (horizon fini) pour des deux jeux de paramètres par la méthode Monte-Carlo naïf et par échantillonnage préférentiel.
- ✓ Estimer la probabilité de ruine et le temps de la ruine (horizon infini) pour des deux jeux de paramètres par la méthode Monte-Carlo naïf et par échantillonnage préférentiel.
- ✓ Etudier la méthode de changement de l'espace de probabilité et la transformation d'Esscher.
- ✓ Simuler une ruine rare à l'aide de transformation d'Esscher.
- ✓ Etudier le mécanisme de fonctionnement de compagnie diversifiée.
- ✓ Construire le graphe qui affiche l'évolution de la probabilité de ruine en fonction du capital de départ.
- ✓ Construire le graphe qui affiche l'évolution de la probabilité de ruine en fonction du l'horizon T.

9. Problèmes rencontrés :

Dans cette partie, nous allons voir les différents problèmes que nous avons rencontrés dans notre projet. La liste est la suivante :

- Choisir d'autres jeux de paramètres pour calculer la probabilité de la ruine et tracer le graphe qui affiche l'évolution de la probabilité de ruine en fonction du capital de départ et le graphe qui affiche l'évolution de la probabilité de ruine en fonction de l'horizon T. En effet, les jeux de paramètres qui sont dans le sujet du projet ne permettent pas d'avoir des résultats cohérents puisqu'ils sont très petits alors on a eu des difficultés pour tomber sur d'autres jeux de paramètres qui nous donnent le résultat attendu.
- Compagnie diversifiée : on n'a pas pu traiter toute la partie compagnie diversifiée car on a mis beaucoup de temps dans les autres parties : les autres parties forgent la base de notre étude alors : d'abord on a réalisé les tâches les plus importantes afin d'avoir un travail parfaitement encadré, en suite on a réalisé les autres tâches secondaires et comme vous savez, ceci est un projet donc c'est pas évident de tout faire dans un intervalle de temps réduit, mais l'important est que les tâches principales soient parfaitement effectuées.

10. Conclusion

Pour conclure, en plus de répondre aux réalisations, notre travail possède plusieurs fonctionnalités de simulation proposées dans le cahier des charges et d'autres de notre initiative. Par conséquent nous avons modélisé une compagnie d'assurance grâce à la simulation du modèle de Lundberg, nous avons calculé la probabilité de ruine de cette branche d'assurance en se basant sur la simulation des résultats de la théorie du risque. La modélisation de la réserve nous a permis d'identifier le modèle de risque correspondant à la branche qui est le modèle de Lundberg, en supposant l'indépendance entre le nombre et le montant des réclamations et la constance de la prime.

La simulation des résultats de la théorie classique de la ruine au modèle de Lundberg nous a permis d'obtenir une expression exacte de la probabilité de ruine à temps infini. Quant à la probabilité de ruine à temps fini, nous avons présenté des résultats pour ce modèle particulier. Notons que pour la probabilité de ruine à horizon fini, la simulation ne demeure pas compliquée, même si les expressions de cette probabilité sont complexes. Il est alors intéressant d'utiliser des méthodes numériques ou des techniques de simulation.

Le cours de **MC & MC** nous a bien aidé pour bien traiter ce projet et nous sommes satisfaits du travail que nous avons effectué. Nous avons anticipé de nombreux problèmes, mais le module de **Travail en équipe** nous a bien guidé ce qui a permis de trouver rapidement les solutions. Ce projet est particulièrement motivant pour choisir la filière **Ingénierie financière** car l'étude peut se traduire dans le monde réel et le cahier des charges est détaillé, ce qui nous fait sentir utiles et efficaces. De plus, ce projet a été pour nous un bon aperçu de ce que serait **un travail de groupe en entreprise** (savoir faire participer chaque membre, quel que soit son niveau, et intégrer les travaux de chacun au rendu final), et a permis de nous améliorer rapidement.

Annexe 1

Python :

```
1      #-----#
2      # 1. Processus de comptage Nt par deux façons différentes #
3      #-----#
4
5 # importer les bibliothèques #
6
7 from math import *
8 from numpy import *
9 from random import *
10 from matplotlib.pyplot import *
11
12 #-----#
13 # Simulation d'une v.a. Exponentielle:(Par inversion de fonction de Répartition) #
14 #-----#
15
16 def V_A_Exponentielle(λ):
17     U=random()
18     X=-(1/λ)*log((1-U))
19     return X
20
21 def Chaine_V_A_Exponentielle(λ, Nmc):
22     l=[]
23     for i in range(0,Nmc):
24         l.append(V_A_Exponentielle(λ))
25     return l
26
27 #-----#
28 # Verification de simulation #
29 #-----#
30
31 def E_empirique_Exponentielle(λ,Nmc):
32     X=Chaine_V_A_Exponentielle(λ, Nmc)
33     E=(1/Nmc)*(sum(X[i] for i in range(0,Nmc)))
34     return E
35
36 def Var_empirique_Exponentielle(λ,Nmc):
37     X=Chaine_V_A_Exponentielle(λ, Nmc)
38     E=(1/Nmc)*(sum(X[i] for i in range(0,Nmc)))
39     D=(1/Nmc)*(sum(pow(X[i],2) for i in range(0,Nmc)))
40     return(D-pow(E,2))
41
42 #-----#
43 # Simulation du processus de comptage Nt ( Methode 1: Par définition de Nt ) #
44 #-----#
45
46 def V_A_Processus_Comptage(λ,t,N):          #ici N represente un grand nombre pour modeliser l'infini (N>>100)#
47     z=[ ]
48     Nt=0
49     for k in range(1,N+1):
50         for i in range(1,k+1):
51             z.append(V_A_Exponentielle(λ))
52             if (sum(z[j] for j in range(0,k)))<=t:    # T(k)<=t#
53                 Nt=Nt+1
54     return Nt
```

```

55
56 def Chaine_V_A_Processus_Comptage(λ,t,N,Nmc):      #ici N represente un grand nombre pour modeliser l'infini (N>>100)#
57     l=[]
58     for i in range(0,Nmc):
59         l.append(V_A_Processus_Comptage(λ,t,N))
60     return l
61
62 #-----#
63 # Verification de simulation #
64 #-----#
65
66 def E_empirique_Processus_Comptage(λ,t,N,Nmc):      #ici N represente un grand nombre pour modeliser l'infini (N>>100)#
67     X=Chaine_V_A_Processus_Comptage(λ,t,N,Nmc)
68     E=(1/Nmc)*(sum(X[i] for i in range(0,Nmc)))
69     return E
70
71 def Var_empirique_Processus_Comptage(λ,t,N,Nmc):    #ici N represente un grand nombre pour modeliser l'infini (N>>100)#
72     X=Chaine_V_A_Processus_Comptage(λ,t,N,Nmc)
73     E=(1/Nmc)*(sum(X[i] for i in range(0,Nmc)))
74     D=(1/Nmc)*(sum(pow(X[i],2) for i in range(0,Nmc)))
75     return(D-pow(E,2))
76
77 #-----#
78 # Simulation du processus de comptage Nt ( Methode 2: Par définition de loi de Poisson composée ) #
79 #-----#
80
81 def V_A_Poisson_Composee(μ,t):
82     X=np.random.poisson(μ*t)      # Une méthode de simulation directe de v.a de Poisson avec Numpy
83     return X
84
85 def Chaine_V_A_Poisson_Composee(μ,t,Nmc):
86     l=[]
87     for i in range(0,Nmc):
88         l.append(V_A_Poisson_Composee(μ,t))
89     return l
90
91 #-----#
92 # Verification de simulation #
93 #-----#
94
95 def E_empirique_Poisson_Composee(μ,t,Nmc):
96     X=Chaine_V_A_Poisson_Composee(μ,t,Nmc)
97     E=(1/Nmc)*(sum(X[i] for i in range(0,Nmc)))
98     return E
99
100 def Var_empirique_Poisson_Composee(μ,t,Nmc):
101     X=Chaine_V_A_Poisson_Composee(μ,t,Nmc)
102     E=(1/Nmc)*(sum(X[i] for i in range(0,Nmc)))
103     D=(1/Nmc)*(sum(pow(X[i],2) for i in range(0,Nmc)))
104     return(D-pow(E,2))
105
106 #-----#
107 # Densité empirique de v.a de Poisson composée  #
108 #-----#
109
110 def f_x(X,a,delta):
111     N_x=200
112     x=array([])
113     proba=np.array([])
114     Nmc=len(X)
115
116     for i in range(N_x):
117
118         x.append(x,a+delta*(i))
119         counter=0
120
121         for k in range(Nmc):
122             if (x[i]<=X[k]<=x[i]+delta):
123                 counter +=1
124             proba.append(proba,counter/(Nmc*delta))
125     return(x, proba)

```

```

126
127
128
129 def graphe_Poisson(mu,t,Nmc):
130     X=array([])
131     for i in range(Nmc):
132         X.append(X,V_A_Poisson_Compousee(mu,t))
133     y=f_x(X,0,0.1)[1]
134     z=f_x(X,0,0.1)[0]
135     bar(z,y,0.2,color='black')
136     title('Densité empirique de v.a de Poisson pour t= et lambda=')
137     # Il faut choisir la valeur de lambda
138     show()

```

Scilab :

```

1 Nmc = 10000;
2
3 // Simulation loi poisson composée
4
5 function[X] = Poisson(lambda)
6     n=0;
7     proba=exp(-lambda);
8     F=proba;
9     U=rand();
10    while U>F
11        proba=(lambda/(n+1))^proba;
12        F=F+proba;
13        n=n+1;
14    end
15    X=n;
16 endfunction
17
18 // Construction d'une chaîne de valeur suivant une loi de Poisson
19 function[X]=Chaine_de_valeur_poisson(lambda)
20     for n=1:Nmc
21         X(n)=Poisson(lambda);
22     end
23 endfunction
24
25 // Construction de la fonction de densité empirique
26 function[Densite] = Densite_poisson(X)
27     a = 0;
28     b = 15; // on découpe en Nmc intervalles de 1
29     delta = 1;
30     for i=1:b+1
31         x(i) = a + delta*(i-1);
32         c=0; // initialisation du compteur
33         for n=1:Nmc
34             if x(i) <= X(n) & X(n) < x(i) + delta
35                 c = c + 1;
36             end
37         end
38         Proba(i)=c/Nmc;
39         Densite(i)=Proba(i)/1; // notre VA est discrète
40     end
41 endfunction

```

```

42 X1 = Chaine_de_valeur_poisson(2);
43 X2 = Chaine_de_valeur_poisson(5);
44 X3 = Chaine_de_valeur_poisson(1/24);
45
46 D1=Densite_poisson(X1);
47 D2=Densite_poisson(X2);
48 D3=Densite_poisson(X3);
49
50 x = linspace(0,10,1)
51
52
53 subplot(221);
54 plot2d3(D1);
55 xtitle("Densité empirique de v.a.de Poisson avec lambda=2")
56 subplot(222);
57 plot2d3(D2);
58 xtitle("Densité empirique de v.a.de Poisson avec lambda=5")
59 subplot(223);
60 plot2d3(D3);
61 xtitle("Densité empirique de v.a.de Poisson avec lambda=1/24")

1 //..Processus de comptage
2
3 clear ;
4
5 Nmc = 10000;
6
7
8 //..On compte le nombre de sinistre entre l'instant 0 et la date t.:
9
10 function[X] = Expo(lambda)
11 .... X=-log(rand())/lambda;
12 endfunction
13
14 //..Cette fonction calcul le nombre de sinistre pendant une durée t
15 function[N] = Comptage(t,lambda)
16 .... c=0;
17 .... T=0;
18 .... while T<=t
19 .... .... tau = Expo(lambda);
20 .... .... T = T+tau;
21 .... .... c=c+1;
22 .... .... N=c
23 .... end
24 endfunction
25
26 N1 = Comptage(1,2);
27 N2 = Comptage(1,5);
28 N3 = Comptage(1,1/24);
29
30 disp("Affichage du nombre de sinistre pour différents jeux de paramètres")
31 disp("Pour t=1 et lambda=2 alors ... Nt = "+string(N1));
32 disp("Pour t=1 et lambda=5 alors ... Nt = "+string(N2));
33 disp("Pour t=1 et lambda=1/24 alors Nt = "+string(N3));

```

Annexe 2

Scilab :

```

1 Nmc = 10000;
2
3 //simulation·loi·de·Gamma·comme·somme·de·n·variables·exponentielles
4
5 //simulation·comme·n=B·variables·aléatoires·qui·suivent·une·loi·exponentielle
6 //Cette·méthode·ne·marche·que·pour·des·beta·entier.
7 function[X] = Gamma(alpha,B)
8     Y = -log(rand())/B;
9     i = 1;
10    while i<alpha
11        Y = Y - log(rand())/B;
12        i = i + 1;
13    end
14    X = Y;
15 endfunction
16
17 //construction·d'une·chaine·de·valeur·suivant·une·loi·de·Gamma·comme·somme·de·n·variables·exponentielles
18 function[X]=Chaine_de_valeur_gamma(alpha,B)
19     for n=1:Nmc
20         X(n)=Gamma(alpha,B);
21     end
22 endfunction
23
24
25 //Cette·fonction·permet·de·construire·la·fonction·de·densité·de·la·fonction·Gamma
26 function[x,D] = Densite_gamma(X)
27     a = 0;
28     b = 20;
29     delta = 20*b/Nmc;
30     for i=1:Nmc/20
31         x(i) = a + delta*(i-1);
32         c=0; //initialisation·du·compteur
33         for n=1:Nmc
34             if x(i) <= X(n) & X(n)< x(i) + delta
35                 c = c + 1;
36             end
37         end
38         Proba(i)=c/Nmc;
39         D(i)=Proba(i)/delta; //notre·VA·est·continue
40     end
41 endfunction
42
43 subplot(221)
44 Y1 = Chaine_de_valeur_gamma(1,1/2);
45 [x1,D1] = Densite_gamma(Y1);
46 plot(x1,D1);
47 xtitle("Simulation·de·loi·gamma·comme·somme·de·lois·exponentielles·;·alpha=1,·beta=1/2");

```

```

1 Nmc=10000;
2
3 //densité de la loi Gamma
1 function[y] = f_Gamma(a,b,x)
2 ... y=(b^a)/gamma(a)*exp(-b*x)*(x^(a-1));
3 endfunction
7
8 //Parametre de rejet C>1 il faut que b>d et n<a<n+1
1 function[c] = C(a,b,n,d)
2 ... c=(gamma(n)/gamma(a))*((b^a)/(d^n))*(((a-n)/(b-d))^(a-n))*exp(-(a-n))
3 endfunction
12
13
14
1 function[X] = V_A_Gamma(a,b)
2 ... U=rand();
3 ... Y=-log(1-U)/b;
4 ... i=1;
5 ... while i<a
6 ... ... Y=Y+-log(1-U)/b;
7 ... ... i=i+1;
8 ... end
9 ... X=Y;
10 endfunction
25
1 function[X] = Rajet_Gamma(a,b,n,d)
2 ... U=rand();
3 ... Y=V_A_Gamma(n,d);
4 ... if U<=(f_Gamma(a,b,Y)/(C(a,b,n,d)*f_Gamma(n,d,Y)))
5 ... ... X=Y;
6 ... end
7 endfunction
1 function[x,D] = Densite_gamma(X)
2 ... a=0;
3 ... b=20;
4 ... delta=20*b/Nmc;
5 ... for i=1:Nmc/20
6 ... ... x(i)=a+delta*(i-1);
7 ... ... c=0;
8 ... ... for n=1:Nmc
9 ... ... ... if x(i)<=X(n) & X(n)<x(i)+delta
10 ... ... ... c=c+1;
11 ... ... end
12 ... ... end
13 ... ... Proba(i)=c/Nmc;
14 ... ... D(i)=Proba(i)/delta;
15 ... ... end
16 endfunction

```

```

1 function[X]=Chaine_de_valeur_gamma(a,b,n,d)
2     for n=1:Nmc
3         X(n)=Rejet_Gamma(a,b,n,d);
4     end
5 endfunction
57
58 subplot(421);
59 Y1 = Chaine_de_valeur_gamma(5/2,1/2,2,1/4);
60 [x1,D1] = Densite_gamma(Y1,1.0);
61 plot2d(x1,D1);
62 xtitle("densité empirique d'une loi Gamma par méthode de rejet :: alpha=5/2 beta=1/2");

```

Annexe 3

Python :

```

164 #-----#
165 # Simulation de loi Pareto(a,b) avec a,b>0          #
166 #-----#
167
168 def pareto(a,b):
169     U=random()
170     X=b/(U**(1/a))                                #
171     return X
172
173 def graphe_pareto(a,b,Nmc):
174     X=np.array([])
175     for i in range(Nmc):
176         X=np.append(X,pareto(a,b))
177     y=f_x(X,0,0.1)[1]
178     z=f_x(X,0,0.1)[0]
179     xlim=(0.5)
180     ylim=(0,3)
181     scatter(z,y,color='black',marker='.')
182     title('Densité empirique de v.a de Pareto a= et b=') # Il faut choisir la valeur de a et beta
183     show()
184

```

Scilab :

```

1 Nmc = 10000;
2 //Modélisation des dommages causés par le COVID-19 par une variable aléatoire de Pareto
3
4 //On utilise la méthode d'inversion de la fonction de répartition
5
6 function[X] = Pareto(k,xm)
7 ... X = xm*rand()^k;
8 endfunction
9
10 function[X] = Chaine_de_valeur_pareto(k,xm)
11 ... for i=1:Nmc
12 ... ... X(i)=Pareto(k,xm);
13 ... end
14 endfunction
15
16 function[Densite] = Densite_pareto(X)
17 ... a = 0;
18 ... b = 5;
19 ... delta = 20*b/Nmc;
20 ... for i=1:Nmc/20
21 ... ... x(i) = a + delta*(i-1);
22 ... ... c=0; //initialisation du compteur
23 ... ... for n=1:Nmc
24 ... ... ... if x(i) <= X(n) & X(n)< x(i) + delta
25 ... ... ... ... c = c + 1;
26 ... ... end
27 ... end
28 ... Proba(i)=c/Nmc;
29 ... Densite(i)=Proba(i)/delta; //notre VA est continue
30 ... end
31 ... plot(x,Densite);
32 ... xtitle("fonction de densité empirique d'une variable aléatoire suivant la loi de Pareto a=2 et b=3");
33 endfunction
34
35 disp("La loi de Pareto peut permettre de modéliser le coût d'un sinistre grave");
36
37 W = Chaine_de_valeur_pareto(2,3);
38 D=Densite_pareto(W);

```

Annexe 4

Code Python :

```
87 #-----#
88 # Simulation Rt à queues fines : Zk -> Gamma(α,β) #
89 #-----#
90
91
92 def Rt_fines(x,c,λ,α,β,t):
93
94     Nt=V_A_Poisson_Composee(λ,t)
95     Z=[]
96     for k in range(0,Nt):
97         | Z.append(V_A_Gamma(α,β))
98     return x+c*t-sum(Z)
99
100
101 def grapheRt_fines(x,c,λ,α,β):
102
103     t=np.linspace(0.,50.,100)
104     y=[Rt_fines(x,c,λ,α,β,i) for i in t]
105     plot(t,y)
106     xlabel("temps t")
107     ylabel("la réserve de la compagnie d'assurance Rt")
108     title("Modèle de Lundberg-Cramer (à queues fines)")
109     show()
110
111
112
113 #-----#
114 # Simulation Rt à queues lourde : Zk -> Pareto(a,b) #
115 #-----#
116
117 def Rt_lourde(x,c,λ,a,b,t):
118
119     Nt=V_A_Poisson_Composee(λ,t)
120     Z=[]
121     for k in range(0,Nt):
122         | Z.append(V_A_Pareto(a,b))
123     return x+c*t-sum(Z)
124
125
126 def grapheRt_lourde(x,c,λ,a,b):
127
128     t=np.linspace(0.,50.,100)
129     y=[Rt_lourde(x,c,λ,a,b,i) for i in t]
130     plot(t,y)
131     xlabel("temps t")
132     ylabel("la réserve de la compagnie d'assurance Rt")
133     title("Modèle de Lundberg-Cramer (à queues lourde)")
134     show()
135
```

Annexe 5

Code Python :

```
137 #-----#
138 # Simulation XT et Probabilité de ruine (à queues fines) : Zk -> Gamma(α,β) #
139 #-----#
140
141 def V_A_X_fines(λ,T,α,β):
142     NT=V_A_Poisson_Composee(λ,T)
143     Z=[]
144     for k in range(0,NT):
145         Z.append(V_A_Gamma(α,β))
146     return sum(Z)
147
148 def Proba_ruine_fines(c,x,T,λ,α,β,Nmc):
149     a=x+c*T
150     counter=0
151     for n in range(1,Nmc+1):
152         X=V_A_X_fines(λ,T,α,β)
153         if X>a:
154             counter=counter+1
155     return counter/Nmc
156
157 #-----#
158 # Simulation XT et Probabilité de ruine (à queues lourde) : Zk -> Pareto(a,b) #
159 #-----#
160
161 def V_A_X_lourde(λ,T,a,b):
162     NT=V_A_Poisson_Composee(λ,T)
163     Z=[]
164     for k in range(0,NT):
165         Z.append(V_A_Pareto(a,b))
166     return sum(Z)
167
168 def Proba_ruine_lourde(c,x,T,λ,a,b,Nmc):
169     a=x+c*T
170     counter=0
171     for n in range(1,Nmc+1):
172         X=V_A_X_lourde(λ,T,a,b)
173         if X>a:
174             counter=counter+1
175     return counter/Nmc
176
```

Annexe 6

Code Python :

```
178 #-----#
179 # Estimation de l'instant de la ruine Tx (à queues fines) : Zk -> Gamma(α,β) #
180 #-----#
181
182 def Tx_fines(x,c,λ,α,β,delta):
183     t=0
184     while Rt_fines(x,c,λ,α,β,t)>0:
185         t=t+delta
186     return t
187
188 #-----#
189 # Estimation de l'instant de la ruine Tx (à queues fines) : Zk -> Pareto(a,b) #
190 #-----#
191
192 def Tx_lourde(x,c,λ,a,b,delta):
193     t=0
194     while Rt_lourde(x,c,λ,a,b,t)>0:
195         t=t+delta
196     return t
197
```

Annexe 7

Code Python :

```
199 #-----#
200 # Simulation Importance Sampling : Zk_Q -> Gamma(α,β-θ)    #
201 #-----#
202
203
204 def V_A_X_Q(c,x,λ,T,α,β):
205     θ=β-β*pow((β*(x+c*T))/(α*λ*T), -1/(α+1))
206     λQ=λ*pow(β/(β-θ),α)
207     NT=V_A_Poisson_Composee(λQ,T)
208     Z=[]
209     for k in range(0,NT):
210         Z.append(V_A_Gamma(α,β-θ))
211     return sum(Z)
212
213
214 def Proba_ruine_Q(c,x,T,λ,α,β,Nmc):
215     θ=β-β*pow((β*(x+c*T))/(α*λ*T), -1/(α+1))
216     Γθ=λ*T*(pow(β/(β-θ),α)-1)
217     a=x+c*T
218     counter=0
219     for n in range(1,Nmc+1):
220         X=V_A_X_Q(c,x,λ,T,α,β)
221         if X>a:
222             counter=counter+exp(-θ*X+Γθ)
223     return counter/Nmc
```

Annexe 8

Code Python :

```
226 |     | #----- Travail à faire -----#
227
228
229 #-#-----#
230 # Graphe de Probabilité de ruine en fonction du Capital c  #
231 #-----#
232
233
234 def grapheProba_ruine_Q(x,T,λ,α,β,Nmc):
235     c=np.linspace(0.,50.,50)
236     y=[Proba_ruine_Q(i,x,T,λ,α,β,Nmc) for i in c]
237     plot(c,y)
238     xlabel("Capital c")
239     ylabel("Probabilité de la ruine")
240     title("Graphe de probabilité de ruine en fonction du capital")
241     show()
242
```

Annexe 9

Code Python :

```
244 #-----#
245 # Graphe de Probabilité de ruine en fonction du horizon T #
246 #-----#
247
248
249 def grapheProba_ruine_Q_T(c,x,λ,α,β,Nmc):
250     t=np.linspace(0.1,20.,20)
251     y=[Proba_ruine_Q(c,x,i,λ,α,β,Nmc) for i in t]
252     plot(t,y)
253     xlabel("Horizon T")
254     ylabel("Probabilité de la ruine")
255     title("Graphe de probabilité de ruine en fonction du horizon T")
256     show()
257
```