Projet GM2 19-20 orienté Maths/Finance. Simulation du modèle de Lundberg

PROJET : Simulation du modèle de Lundberg

Ruine en Assurance

Groupe 2 – MF01

ZIRIHI YANNICK

KOUROUMA ABOUBACAR

EL KASMI ADNANE

- **Introduction :**

La théorie de la ruine appartient aux sciences de la gestion des risques et aux mathématiques appliquées à l'assurance. Il s'agit de l'étude mathématique de modèles stochastiques et dynamiques adaptés aux réserves financières allouées à un portefeuille de contrats d'assurance de type non-vie d'une compagnie d'assurance. Assurance de type IARD (Incendie, Accidents et Risques Divers).

L'objectif est de définir un cadre permettant la bonne gestion d'un portefeuille de contrat. On imagine une compagnie d'assurance qui se lance sur un nouveau marché, le business line doit être :

- ✓ Viable, la compagnie doit être solvable a tout instant (la réserve ne doit pas tomber en dessous de 0). Elle doit être en mesure de faire face aux engagements qu'elle a pris vis à vis des assurés par voie contractuelle.
- ✓ Rentable, la tarification doit permettre à l'assureur d'engranger des bénéfices pour rémunérer les actionnaires et les employés.

L'étude des processus aléatoires s'insère dans la théorie des probabilités dont elle constitue l'un des objectifs les plus profonds. Elle soulève des problèmes mathématiques intéressants et souvent très difficiles. Ce projet présente quelques aspects des processus aléatoires utiles à l'ingénieur mathématicien du fait de leur fréquence d'occurrence dans les applications : processus de renouvellement, processus de Markov, mouvement brownien etc.

1. Livrable 2:

1.1 Activité 1. Simulation Monte-Carlo:

Dans cette activité

1. 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 :

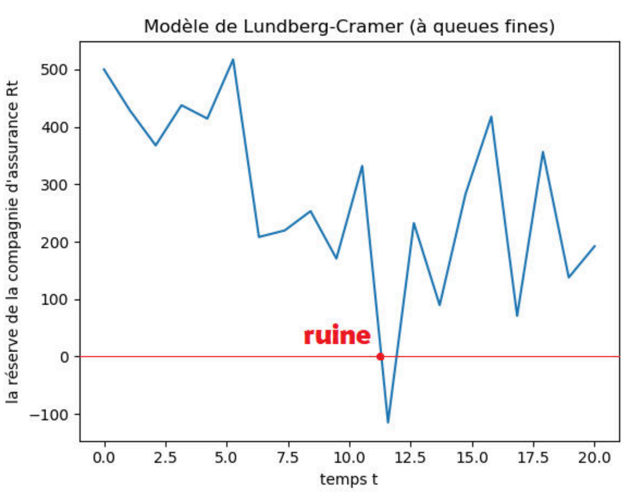
- $\mathbf{function}[Rt] = Rt\_fines(c, x, \lambda, \alpha, \beta, t)$
  - $N_t = V\_A\_Poisson\_Compose(\lambda, t)$
  - $X = 0$
  - $for\ k = 1{:}\ N_t$
    - $Z_k = V\_A\_Gamma(\alpha, \beta)$
    - $X = X + Z_k$
  - $endfor$
  - $set\ Rt = x + c.t - X$
- $\mathbf{endfunction}$

- Algorithme de simulation de R_t à queues lourde :

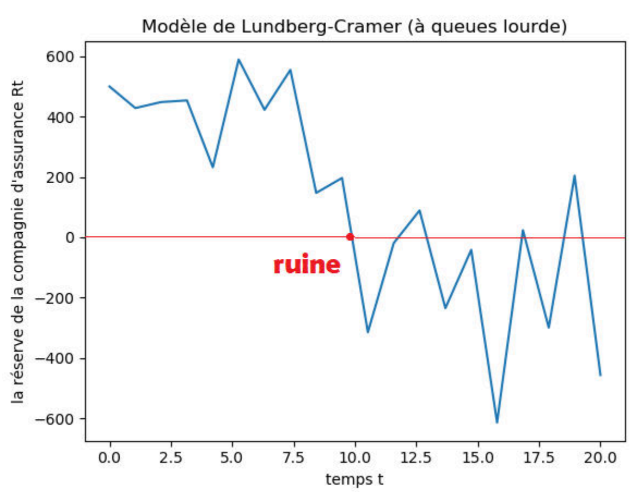
- $\mathbf{function}[Rt] = Rt\_lourde(c, x, \lambda, a, b, t)$
  - $N_t = V\_A\_Poisson\_Compose(\lambda, t)$
  - $X = 0$
  - $for\ k = 1{:}\ N_t$
    - $Z_k = V\_A\_Pareto(a, b)$
    - $X = X + Z_k$
  - $endfor$
  - $set\ Rt = x + c.t - X$
- $\mathbf{endfunction}$

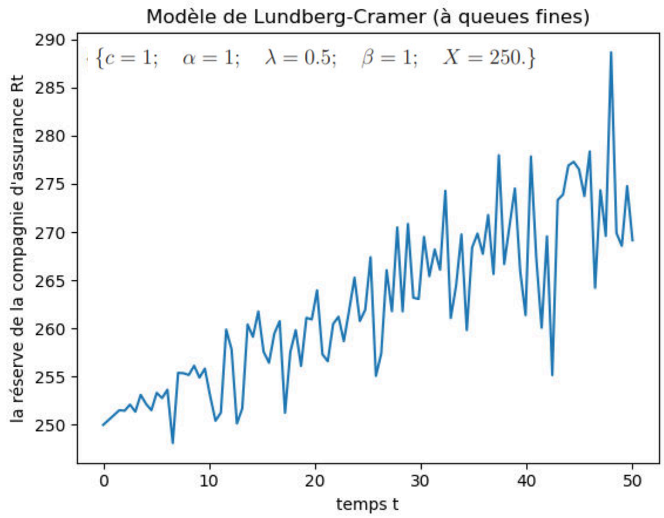
Après la simulation on obtient ces graphes :

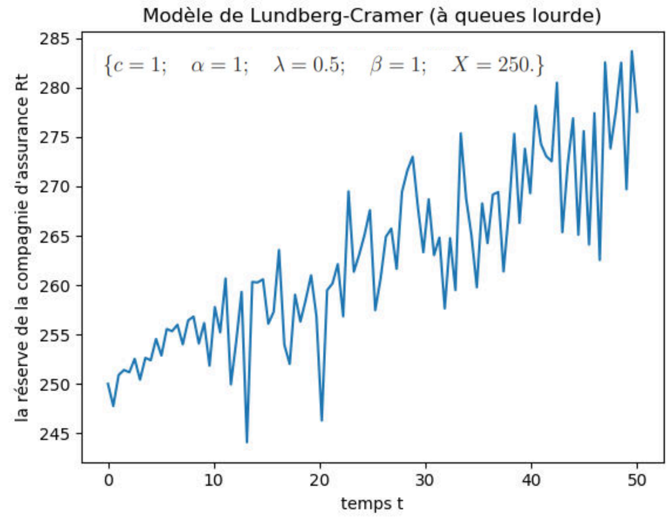
{c = 100; α = 8; λ = 6; β = 2.5; X = 500.}



{c = 100; a = 2; λ = 7; b = 10; X = 500.}







(Codes Python : voir annexe 1)

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 = 1ans (horizon fini), pour les jeux de paramètres $\{\, \boldsymbol{c = 100},\ \boldsymbol{x = 50},\ \boldsymbol{T = 1},\ \boldsymbol{\lambda = 6},\ \boldsymbol{\alpha = 8},\ \boldsymbol{\beta = 2.5} \,\}$ pour $N_{mc} = 10^6$

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

$$P(R_T < 0 \mid R(0) = x)$$

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

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 : $P(X_T > a) = E\left[1_{\{X_T > a\}}\right] = \frac{1}{N_{mc}} \sum_{n=1}^{N_{mc}} 1_{\{X_T(n) > a\}}$ avec $N_{mc} = 10^6$

• Algorithme de simulation de X_T à queues fines :

- $\mathbf{function}[X] = \boldsymbol{V\_A\_X(c, x, \lambda, T, \alpha, \beta)}$
  - $\boldsymbol{N_T = V\_A\_Poisson\_Compose(\lambda, T)}$
  - $\boldsymbol{X = 0}$
  - $\boldsymbol{for\ k = 1 : N_T}$
    - $\boldsymbol{Z_k = V\_A\_Gamma(\alpha, \beta)}$
    - $\boldsymbol{X = X + Z_k}$
  - $\boldsymbol{endfor}$
  - $\boldsymbol{set\ X}$
- $\mathbf{endfunction}$

• Algorithme de simulation de X_T à queues lourde :

- $\mathbf{function}[X] = \boldsymbol{V\_A\_X(c, x, \lambda, T, \alpha, \beta)}$
  - $\boldsymbol{N_T = V\_A\_Poisson\_Compose(\lambda, T)}$
  - $\boldsymbol{X = 0}$
  - $\boldsymbol{for\ k = 1 : N_T}$
    - $\boldsymbol{Z_k = V\_A\_Pareto(\alpha, \beta)}$
    - $\boldsymbol{X = X + Z_k}$
  - $\boldsymbol{endfor}$
  - $\boldsymbol{set\ X}$
- $\mathbf{endfunction}$

• <u>Algorithme de simulation de Probabilité de ruine :</u>

> • $\textbf{function}[\boldsymbol{Proba}] = \boldsymbol{Proba\_Ruine}(\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{T}, \boldsymbol{\alpha}, \boldsymbol{\beta})$
>
> ○ $\boldsymbol{Counter = 0}$, $\quad \boldsymbol{a = x + c.T}$
>
> ○ $\boldsymbol{for\ n = 1 : N_{mc}}$
>
> ○ $\boldsymbol{X(n) = V\_A\_X(c, x, \lambda, T, \alpha, \beta)}$
>
> ○ $\boldsymbol{if\ X(n) > a}$
>
> ○ $\boldsymbol{Counter = Counter + 1}$
>
> ○ $\boldsymbol{endif}$
>
> ○ $\boldsymbol{endfor}$
>
> ○ $\boldsymbol{set\ Proba = \dfrac{Counter}{N_{mc}}}$
>
> • $\textbf{endfunction}$

(Codes Python : voir annexe 2)

Pour : $\{\ \boldsymbol{c = 100},\ \boldsymbol{x = 50},\ \boldsymbol{T = 1},\ \boldsymbol{\lambda = 6},\ \boldsymbol{\alpha = 8},\ \boldsymbol{\beta = 2.5}\ \}$ 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 : $\{\ \boldsymbol{c = 1},\ \boldsymbol{x = 250},\ \boldsymbol{T = 1},\ \boldsymbol{\lambda = 0.5},\ \boldsymbol{\alpha = 1},\ \boldsymbol{\beta = 1}\ \}$ 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.

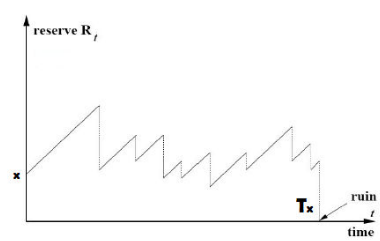
<u># Réponse :</u>

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

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

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

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 = 1ans de la question 2 Par T = T_x) :

• <u>Algorithme de simulation de T_x à queues fines :</u>

• $\mathbf{function}[t] = \boldsymbol{Tx\_fines}(\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \Delta)$ ○ $\boldsymbol{R_t} = \boldsymbol{Rt\_fines}(\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{t})$ ○ $\boldsymbol{t} = \mathbf{0}$ ○ $\boldsymbol{while}\ \boldsymbol{R_T} > \mathbf{0}$ ○ $\boldsymbol{t} = \boldsymbol{t} + \Delta$ ○ $\boldsymbol{endwhile}$ ○ $\boldsymbol{set}\ \boldsymbol{t}$ • $\mathbf{endfunction}$

• <u>Algorithme de simulation de T_x à queues lourde :</u>

• $\mathbf{function}[t] = \boldsymbol{Tx\_lourde}(\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \Delta)$ ○ $\boldsymbol{R_t} = \boldsymbol{Rt\_lourde}(\boldsymbol{c}, \boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{t})$ ○ $\boldsymbol{t} = \mathbf{0}$ ○ $\boldsymbol{while}\ \boldsymbol{R_T} > \mathbf{0}$ ○ $\boldsymbol{t} = \boldsymbol{t} + \Delta$ ○ $\boldsymbol{endwhile}$ ○ $\boldsymbol{set}\ \boldsymbol{t}$ • $\mathbf{endfunction}$

(Codes Python : voir annexe 3)

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

```
>>> 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.

<u># Réponse :</u>

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 normale car parfois $P(X_T > a) \approx 10^{-30}$

➔ ***Solution*** : Pour pouvoir calculer cette probabilité par Monte-Carlo on génère dans l'espace $(\Omega,\ F_T,\ Q)$ 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 :

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

Par Monte-Carlo :

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

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

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

5. Nous simulons une ruine rare

Réponse :

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

$\{\ \boldsymbol{c = 100,\ x = 50,\ T = 1,\ \lambda = 7,\ \alpha = 4,\ \beta = 1.5}\ \}$ 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**$[X] = \boldsymbol{V\_A\_X\_Q(c, x, \lambda, T, \alpha, \beta)}$
  - $\boldsymbol{\theta = \beta - \beta\left(\frac{\beta.(x + c.T)}{\alpha\lambda T}\right)^{\frac{-1}{\alpha+1}}}$
  - $\boldsymbol{\lambda^Q = \lambda.\left(\frac{\beta}{\beta - \theta}\right)^{\alpha}}$
  - $\boldsymbol{N_T = V\_A\_Poisson\_Compose(\lambda^Q, T)}$
  - $\boldsymbol{X = 0}$
  - $\boldsymbol{for\ k = 1 : N_T}$
    - $\boldsymbol{Z_k^Q = V\_A\_Gamma(\alpha, \beta - \theta)}$
    - $\boldsymbol{X = X + Z_k^Q}$
  - $\boldsymbol{endfor}$
  - $\boldsymbol{set\ X}$
- **endfunction**

(Codes Python : voir annexe 4)

• Algorithme de simulation de Probabilité de ruine Q:

- **function**$[Proba] = \boldsymbol{Proba\_Ruine\_Q(c, x, \lambda, T, \alpha, \beta)}$
  - $\boldsymbol{\theta = \beta - \beta\left(\frac{\beta.(x + c.T)}{\alpha\lambda T}\right)^{\frac{-1}{\alpha+1}}}$
  - $\boldsymbol{\Gamma(\theta) = \lambda T\left(\left(\frac{\beta}{\beta-\theta}\right)^{\alpha} - 1\right)}$
  - $\boldsymbol{Counter = 0}$, $\boldsymbol{a = x + c.T}$
  - $\boldsymbol{for\ n = 1 : N_{mc}}$
    - $\boldsymbol{X^Q(n) = V\_A\_X\_Q(c, x, \lambda, T, \alpha, \beta)}$
    - $\boldsymbol{if\ X^Q(n) > a}$
      - $\boldsymbol{Counter = Counter + e^{-\theta.X^Q(n) + \Gamma(\theta)}}$
    - $\boldsymbol{endif}$
  - $\boldsymbol{endfor}$
  - $\boldsymbol{set\ Proba = \frac{Counter}{N_{mc}}}$
- **endfunction**

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 $P(X_T > a) = 1{,}45.\,10^{-22}$ 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 :

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

Et

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

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).**

1.2 Activité 2. Etudes mathématiques:

Montrons que dans l'espace de probabilité $(\Omega,\ F_T,\ Q)$ 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 : $\lambda^Q = \lambda.\mathrm{E}\left[e^{\theta.Z_1}\right] = \lambda.(\frac{\beta}{\beta-\theta})^{\alpha}$, $\theta \epsilon R_+^*$

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

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 : $\widehat{L_T} = \frac{dQ}{dP} = e^{\theta.X_T-\boldsymbol{\Gamma}(\theta)}$

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

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

Réponse :

Soit la richesse $R(t) = x + c.t - \sum_{k=1}^{N_t} Z_k = a - X(t)$ avec $a = x + c.t$ 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 $f_G(y) = \frac{1}{\boldsymbol{\Gamma}(\alpha)}\beta^{\alpha}y^{\alpha-1}e^{-\beta y}.1_{y>0}$.

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

On a :

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

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

$$\Phi_{X_{(T)}}(u) = \sum_{k=0}^{+\infty} \mathbb{E}_p[e^{iu\sum_{j=1}^{k} Z_j}]\mathbb{P}(N_T = k) = \sum_{k=0}^{+\infty}\prod_{j=1}^{k} \mathbb{E}_p[e^{iuZ_j}]\mathbb{P}(N_T = k) = \sum_{k=0}^{+\infty} (\mathbb{E}_p[e^{iuZ_j}])^k \frac{(\lambda.T)^k}{k!}e^{-\lambda.T}$$

$$= e^{\lambda.T(\mathbb{E}_p[e^{iuZ_j}]-1)} = e^{\lambda.T(\int_{\mathbb{R}} e^{iuy} f_G(y)dy-1)} = e^{\lambda.T(\int_{\mathbb{R}}(e^{iuy}-1)f_G(y)dy)}$$

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

D'autre part:

$$\Phi_{X_{(T)}^{\mathbb{Q}}}(u) = \mathbb{E}_{\mathbb{Q}}[e^{iuX_{(T)}^{\mathbb{Q}}}] = \mathbb{E}_p[e^{iuX_{(T)}}\frac{d\mathbb{Q}}{d\mathbb{P}}] = \mathbb{E}_p[e^{iuX_{(T)}}\hat{\mathbb{L}}_T] = \mathbb{E}_p[e^{iuX_{(T)}}.e^{\theta.X_{(T)}-\Gamma(\theta)}]$$

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

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

Donc: $\boxed{\Gamma(\theta) = ln(\mathbb{E}[e^{\theta.X_{(T)}}])}$ d'après la question (4) du livrable (1), on a:
$\mathbb{E}_p[e^{\theta.X_{(t)}}] = e^{\lambda.t(\mathbb{E}[e^{\theta.Z_1}]-1)} = e^{\lambda.t((\frac{\beta}{\beta-\theta})^\alpha-1)}$ et par suite:
$\boxed{\Gamma(\theta) = \lambda.T((\frac{\beta}{\beta-\theta})^\alpha - 1)}$ pour t=T.
Et donc: $\Phi_{X^{\mathbb{Q}}_{(T)}}(u) = \mathbb{E}_p[e^{(iu+\theta)X_{(T)}}]e^{\lambda.T(\mathbb{E}[e^{\theta.Z_1}]-1)}$
d'après la question (4) du livrable (1): $\mathbb{E}_p[e^{(iu+\theta)X_{(t)}}] = e^{\lambda.t(\mathbb{E}[e^{(iu+\theta)Z_1}]-1)}$ donc pour t=T

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

par linéarité de l'espérence:

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

Donc: $[\Phi_{X^{\mathbb{Q}}_{(T)}}(u) = e^{\lambda.T\int_{\mathbb{R}}(e^{iuy}-1)e^{\theta.y}f_G(y)dy}\quad]$ (**)
Par identification de (*) et (**) on trouve:

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

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

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

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

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

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

D'après ce qui précède on a trouvé que: $\Gamma(\theta) = \lambda.T((\frac{\beta}{\beta-\theta})^\alpha - 1)$
On sait que θ est la racine de $\Gamma'(\theta)$=a avec a=x+cT

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

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

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

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

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

Annexe 1

Code Python :

```
#-------------------------------------------------------#
# Simulation Rt à queues fines : Zk -> Gamma(α,β)       #
#-------------- ----------------------------------------#


def Rt_fines(x,c,λ,α,β,t):

    Nt=V_A_Poisson_Composee(λ,t)
    Z=[]
    for k in range(0,Nt):
        Z.append(V_A_Gamma(α,β))
    return x+c*t-sum(Z)


def grapheRt_fines(x,c,λ,α,β):

    t=np.linspace(0.,50.,100)
    y=[Rt_fines(x,c,λ,α,β,i) for i in t]
    plot(t,y)
    xlabel("temps t")
    ylabel("la réserve de la compagnie d'assurance Rt")
    title("Modèle de Lundberg-Cramer (à queues fines)")
    show()
```

```
#-------------------------------------------------------#
# Simulation Rt à queues loude : Zk -> Pareto(a,b)      #
#-------------- ----------------------------------------#

def Rt_lourde(x,c,λ,a,b,t):

    Nt=V_A_Poisson_Composee(λ,t)
    Z=[]
    for k in range(0,Nt):
        Z.append(V_A_Pareto(a,b))
    return x+c*t-sum(Z)


def grapheRt_lourde(x,c,λ,a,b):

    t=np.linspace(0.,50.,100)
    y=[Rt_lourde(x,c,λ,a,b,i) for i in t]
    plot(t,y)
    xlabel("temps t")
    ylabel("la réserve de la compagnie d'assurance Rt")
    title("Modèle de Lundberg-Cramer (à queues lourde)")
    show()
```

Annexe 2

Code Python :

```
#-------------------------------------------------------------------------------------#
# Simulation XT et Probabilité de ruine (à queues fines) : Zk -> Gamma(α,β)          #
#-------------- ----------------------------------------------------------------------#

def V_A_X_fines(λ,T,α,β):
    NT=V_A_Poisson_Composee(λ,T)
    Z=[]
    for k in range(0,NT):
        Z.append(V_A_Gamma(α,β))
    return sum(Z)

def Proba_ruine_fines(c,x,T,λ,α,β,Nmc):
    a=x+c*T
    counter=0
    for n in range(1,Nmc+1):
        X=V_A_X_fines(λ,T,α,β)
        if X>a:
            counter=counter+1
    return counter/Nmc

#-------------------------------------------------------------------------------------#
# Simulation XT et Probabilité de ruine (à queues lourde) : Zk -> Pareto(a,b)        #
#-------------- ----------------------------------------------------------------------#

def V_A_X_lourde(λ,T,a,b):
    NT=V_A_Poisson_Composee(λ,T)
    Z=[]
    for k in range(0,NT):
        Z.append(V_A_Pareto(a,b))
    return sum(Z)

def Proba_ruine_lourde(c,x,T,λ,a,b,Nmc):
    a=x+c*T
    counter=0
    for n in range(1,Nmc+1):
        X=V_A_X_lourde(λ,T,a,b)
        if X>a:
            counter=counter+1
    return counter/Nmc
```

Annexe 3

Code Python :

```python
#---------------------------------------------------------------------------------#
# Estimation de l'instant de la ruine Tx (à queues fines) : Zk -> Gamma(α,β)      #
#-------------- ------------------------------------------------------------------#

def Tx_fines(x,c,λ,α,β,delta):
    t=0
    while Rt_fines(x,c,λ,α,β,t)>0:
        t=t+delta
    return t

#---------------------------------------------------------------------------------#
# Estimation de l'instant de la ruine Tx (à queues fines) : Zk -> Pareto(a,b)     #
#-------------- ------------------------------------------------------------------#

def Tx_lourde(x,c,λ,a,b,delta):
    t=0
    while Rt_lourde(x,c,λ,a,b,t)>0:
        t=t+delta
    return t
```

Annexe 4

Code Python :

```python
#-------------------------------------------------------------#
# Simulation Importance Sampling : Zk_Q -> Gamma(α,β-θ)      #
#-------------- ----------------------------------------------#


def V_A_X_Q(c,x,λ,T,α,β):
    θ=β-β*pow((β*(x+c*T))/(α*λ*T),-1/(α+1))
    λQ=λ*pow(β/(β-θ),α)
    NT=V_A_Poisson_Composee(λQ,T)
    Z=[]
    for k in range(0,NT):
        Z.append(V_A_Gamma(α,β-θ))
    return sum(Z)


def Proba_ruine_Q(c,x,T,λ,α,β,Nmc):
    θ=β-β*pow((β*(x+c*T))/(α*λ*T),-1/(α+1))
    Γθ=λ*T*(pow(β/(β-θ),α)-1)
    a=x+c*T
    counter=0
    for n in range(1,Nmc+1):
        X=V_A_X_Q(c,x,λ,T,α,β)
        if X>a:
            counter=counter+exp(-θ*X+Γθ)
    return counter/Nmc
```