Calcul de VaR et CVaR par algorithmes stochastiques

Nathan Azoulay, Alexandre Martin

Avril 2018

1 Introduction

— Implanter les algorithmes de calcul de V@R et CV@R proposés en cours sur diverses lois (exponentielles, gaussiennes, $\gamma(\alpha, \beta)$, etc). Noter que, pour les lois exponentielles ou des fonctions monotones de variables exponentielles, des formules fermées existent. On commencera par des niveaux de confiance médian (ce qui n'est pas réaliste).

— Lorsque les niveaux s'éloignent de 1/2 on introduira une méthode de réduction de variance récursive et un seuil de risque adaptatif tel que décrit dans [1] (dont le but est de réduire la variance liée à l'estimation de la CV@R). On prendra un soin particulier au réglage du pas et s'attachera à mettre en évidence numériquement des vitesses de convergence.

2 Calculs de la V@R et CV@R par un algorithme de gradient stochastique

La V@R ainsi que la CV@R sont deux mesures de risques largement utilisées par les praticiens. Estimer ces deux quantités est donc un problème essentiel en finance.

Pour L une variable aléatoire intégrable représentative d'une perte, la *Value-at-Risk* de niveau $\alpha \in]0, 1[$ est le plus petit quantile de L de niveau α, *i.e.* $\mathrm{Var}_\alpha(L) = \min\{\xi_\alpha | P(L \leq \xi_\alpha) \geq \alpha\}$. La *Conditional Value-at-Risk*, connue aussi sous le nom *Expected Shortfall*, est quant à elle définie ainsi : $\mathrm{CVar}_\alpha(L) = \mathbb{E}[L | L \geq \mathrm{Var}_\alpha(L)]$. Cette dernière étant sous-additive, contrairement à la V@R, elle favorise la diversification. Dans toute la suite, on se

place dans le contexte suivant : on suppose fixée $L = \varphi(X)$ où X est une variable aléatoire à valeurs dans $\mathbb{R}^d$ définie sur un espace de probabilité fixé, et $\varphi : \mathbb{R}^d \longrightarrow \mathbb{R}$ est une fonction mesurable telle que $\varphi(X)$ soit intégrable.

Afin d'estimer $\mathrm{Var}_\alpha(\varphi(X))$, on peut utiliser un algorithme stochastique. En effet, on sait d'après l'identité de Rockafeller-Uryasev que ces deux quantités, dans le cas où $\varphi(X)$ est sans atome, sont solutions du problème d'optimisation convexe suivant :

$$\mathrm{CVar}_\alpha(\varphi(X)) = \min_{\xi\in\mathbb{R}} V(\xi)$$

et

$$\mathrm{Var}_\alpha(\varphi(X)) = \arg\min_{\xi\in\mathbb{R}} V(\xi)$$

où $V(\xi) = \mathbb{E}[\xi + \frac{1}{1-\alpha}(\varphi(X) - \xi)_+]$.

On peut donc commencer par écrire l'algorithme de gradient stochastique suivant pour obtenir un estimateur de la V@R :

$$\xi_{n+1} = \xi_n - \gamma_{n+1}\left(1 - \frac{1}{1-\alpha}1_{\varphi(X_{n+1})\geq\xi_n}\right)$$

où les $(X_n)_{n\geq 1}$ sont des copies i.i.d de X et ξ_0 intégrable et indépendante de $(X_n)_{n\geq 1}$. La convergence p.s. de l'algorithme vers $\xi_\alpha^* = \mathrm{Var}_\alpha(\varphi(X))$ est assurée par le théorème de Robbins-Monroe sous les hypothèses suivantes :

— $\sum \gamma_n = \infty$
— $\sum \gamma_n^2 < \infty$

Pour estimer CVar_α, l'idée est d'adapter un algorithme de Monte Carlo classique. Si l'on connaît déjà ξ_α^*, alors on a :

$$\mathrm{CVar}_\alpha = V(\xi_\alpha^*) = \mathbb{E}[v(\xi_\alpha^*, X)]$$

avec $v(\xi, x) = \xi + \frac{1}{1-\alpha}(\varphi(x) - \xi)_+$ comme vu plus haut. On aurait donc un estimateur naturel donné par :

$$C_n = \frac{1}{n}\sum_{k=0}^{n-1} v(\xi_\alpha^*, X_{k+1})$$

Plutôt que de d'abord estimer ξ_α^* par une première exécution de l'algorithme vu précédemment, on peut directement injecter son approximation ξ_k à la volée pour obtenir :

$$C_n = \frac{1}{n}\sum_{k=0}^{n-1} v(\xi_k, X_{k+1})$$

ce qui se réecrit de façon récursive en :

$$C_{n+1} = C_n - \frac{1}{n+1}(C_n - v(\xi_n, X_{n+1}))$$

avec $C_0 = 0$. On peut souhaiter remplacer le facteur $\frac{1}{n+1}$ par γ_{n+1}. C'est ce que nous ferons dans notre implémentation. On peut alors montrer que l'algorithme converge p.s. vers $C_\alpha^* = \mathrm{CVar}_\alpha$.

La dernière étape est d'obtenir une vitesse de convergence pour cet algorithme, via un théorème de type central limite. Un tel théorème est fortement conditionné par le choix de γ_n. Choisissant $\gamma_n = \frac{1}{n^a}$ avec $\frac{1}{2} < a < 1$ et en remplaçant (ξ_n, C_n) par leurs moyennes de Cesàro $\bar{Z}_n = (\bar{\xi_n}, \bar{C_n})$, et sous l'hypothèse supplémentaire que $\varphi(X)$ possède une densité $f_{\varphi(X)}$, continue et strictement positive en ξ_α^*, on peut montrer à l'aide du principe de moyennisation de Ruppert et Polyak le résultat de convergence en loi suivant :

$$\sqrt{n}(\bar{Z}_n - (\xi_\alpha^*, C_\alpha^*)) \underset{d}{\to} \mathcal{N}(0, \Sigma)$$

où on a noté Σ la matrice de covariance asymptotique, dont les composantes diagonales sont données par $\Sigma_{1,1} = \frac{1}{f_X^2(\xi_\alpha^*)}\mathrm{Var}(1_{\varphi(X)>\xi_\alpha^*})$ et $\Sigma_{2,2} = \frac{1}{(1-\alpha)^2}\mathrm{Var}((\varphi(X) - \xi_\alpha^*)_+)$

Lorsque $\alpha \approx 1$, il devient difficile d'estimer ces deux quantités avec les algorithmes précédents. En effet, la convergence de l'algorithme stochastique devient chaotique, les évènements $\{\varphi(X_n + 1) \geq \xi_n\}$ devenant très rares. De plus, $\varphi(X)$ peut potentiellement être difficile à simuler ; on souhaite donc réduire au maximum le nombre d'échantillons nécessaires. La prochaine section décrit une méthode de réduction de la variance asymptotique appelée *importance sampling*.

3 Importance Sampling

On cherche à réduire la variance asymptotique de l'algorithme précédent. Dans l'algorithme précédent, on calculait des espérances du type :

$$\mathbb{E}[F(X)]$$

Si l'on suppose que X a une densité p par rapport à la mesure de Lebesgue, on note l'identité suivante pour $\theta \in \mathbb{R}^d$:

$$\mathbb{E}[F(X)] = \mathbb{E}\left[F(X+\theta)\frac{p(X+\theta)}{p(X)}\right]$$

Un bon candidat comme réducteur de variance est donc le vecteur aléatoire $F(X+\theta)\frac{p(X+\theta)}{p(X)}$ où θ minimise la norme quadratique :

$$Q(\theta) = \mathbb{E}\left[\left(F(X+\theta)\frac{p(X+\theta)}{p(X)}\right)^2\right] = \mathbb{E}\left[F^2(X)\frac{p(X)}{p(X-\theta)}\right]$$

On ajoute alors quelques hypothèses supplémentaires. On suppose qu'il existe $b \in [1,2]$ et $\rho > 0$ tels que :

— p est log-concave et $\lim_{|x|\to+\infty} p(x) = 0$

— $\frac{|\nabla p(x)|}{p(x)} = O(|x|^{b-1}), \quad |x| \to +\infty$

— $\log(p(x)) + \rho|x|^b$ est convexe

avec de plus les hypothèses suivantes sur F :

— $\forall A > 0, \mathbb{E}\left[F(X)^2 e^{A|X|^{b-1}}\right] < +\infty$

— il existe $a > 0$ et $C > 0$ tels que $\forall x \in \mathbb{R}^d, |F(x)| \leq Ce^{a|x|}$

Ces hypothèses permettent d'affirmer d'une part que Q est finie et possède un minimum, et d'autre part de pouvoir différentier Q, son gradient étant donné par :

$$\nabla Q(\theta) = \mathbb{E}\left[F^2(X-\theta)\frac{p^2(X-\theta)}{p(X)p(X-2\theta)}\frac{\nabla p(X-2\theta)}{p(X-2\theta)}\right]$$

La dernière hypothèse (contrôle exponentiel sur F, comme décrit dans la section 3 de [2]) permet alors d'affirmer que l'algorithme de gradient stochastique défini par :

$$\theta_{n+1} = \theta_n - \gamma_{n+1} e^{-2a(|\theta|^2+1)-2\rho|\theta|^b} K(\theta_n, X_{n+1})$$

avec

$$K(\theta, x) = F^2(x-\theta)\frac{p^2(x-\theta)}{p(x)p(x-2\theta)}\frac{\nabla p(x-2\theta)}{p(x-2\theta)}$$

converge p.s. vers une variable aléatoire θ^* prenant ses valeurs dans l'ensemble des points qui minimisent Q, et ce encore une fois grâce au théorème de Robbins-Monroe.

On peut maintenant mettre ça en pratique sur l'algorithme de la première partie. Les espérances qui nous intéressaient dans cette partie étaient données par $\mathbb{E}[1_{\varphi(X)\geq\xi_\alpha^*}]$ et $\mathbb{E}[(\varphi(X)-\xi_\alpha^*)_+]$. Vu le paragraphe précédent, on cherche alors à minimiser les fonctionnelles suivantes pour réduire les variances :

$$Q_1(\theta) = \mathbb{E}\left[1_{\varphi(X)\geq\xi_\alpha^*}\frac{p(X)}{p(X-\theta)}\right]$$

$$Q_2(\mu) = \mathbb{E}\left[(\varphi(X)-\xi_\alpha^*)_+^2\frac{p(X)}{p(X-\mu)}\right]$$

et on remplacera donc les espérances précédentes par $\mathbb{E}\left[1_{\varphi(X+\theta^*)\geq\xi_\alpha^*}\frac{p(X+\theta^*)}{p(X)}\right]$ et $\mathbb{E}\left[(\varphi(X+\mu^*)-\xi_\alpha^*)_+\frac{p(X+\mu^*)}{p(X)}\right]$. Évidemment comme au début on ne connaît pas ξ_α^*, θ^* et μ^*, on va les remplacer par leurs estimations à la volée. On obtient finalement l'algorithme suivant :

$$\begin{aligned}
\xi_{n+1} &= \xi_n - \gamma_{n+1}L_1(\xi_n,\theta_n,X_{n+1})\\
C_{n+1} &= C_n - \gamma_{n+1}L_2(\xi_n,C_n,\mu_n,X_{n+1})\\
\theta_{n+1} &= \theta_n - \gamma_{n+1}L_3(\xi_n,\theta_n,X_{n+1})\\
\mu_{n+1} &= \mu_n - \gamma_{n+1}L_4(\xi_n,\mu_n,X_{n+1})
\end{aligned}$$

avec

$$\begin{aligned}
L_1(\xi,\theta,x) &= e^{-\rho|\theta|^b}\left(1-\frac{1}{1-\alpha}1_{\varphi(x+\theta)\geq\xi}\frac{p(x+\theta)}{p(x)}\right)\\
L_2(\xi,C,\mu,x) &= C-\xi-\frac{1}{1-\alpha}(\varphi(x+\mu)-\xi)_+\frac{p(x+\mu)}{p(x)}\\
L_3(\xi,\theta,x) &= e^{-2\rho|\theta|^b}1_{\varphi(x-\theta)\geq\xi}\frac{p^2(x-\theta)}{p(x)p(x-2\theta)}\frac{\nabla p(x-2\theta)}{p(x-2\theta)}\\
L_4(\xi,\mu,x) &= e^{-2a(|\mu|^2+1)-2\rho|\mu|^b}(\varphi(x-\mu)-\xi)_+^2\frac{p^2(x-\mu)}{p(x)p(x-2\mu)}\frac{\nabla p(x-2\mu)}{p(x-2\mu)}
\end{aligned}$$

On montre que cet algorithme converge p.s., et si on applique une moyennisation comme dans la section précédente et qu'on note $\bar{Z}_n = (\bar{\xi_n},\bar{C_n})$ le vecteur moyennisé, on la convergence en loi :

$$\sqrt{n}(\bar{Z}_n-(\xi_\alpha^*,C_\alpha^*)) \underset{d}{\to} \mathcal{N}(0,\Sigma^*)$$

où les composantes diagonales de la matrice de covariance asymptotique Σ^* sont données par :

$$\begin{aligned}
\Sigma_{1,1}^* &= \frac{1}{f_X^2(\xi_\alpha^*)}\mathrm{Var}\left(1_{\varphi(X+\theta^*)>\xi_\alpha^*}\frac{p(X+\theta^*)}{p(X)}\right)\\
\Sigma_{2,2}^* &= \frac{1}{(1-\alpha)^2}\mathrm{Var}\left((\varphi(X+\mu^*)-\xi_\alpha^*)_+\frac{p(X+\mu^*)}{p(X)}\right)
\end{aligned}$$

dont on sait que les deux variances sont minimales dans le cadre de notre réduction de variance par translation.

4 Simulations

Pour les applications numériques, nous avons travaillé sur deux exemples.

	Number of steps	alpha	averaging	Method	V@R	Var(V@R)	CV@R	Var(CV@R)
0	10000	0.95	yes	stochastic-gradient	21.452780	0.075522	32.578392	0.174044
1	10000	0.95	yes	importance-sampling	21.482272	0.072569	32.391713	1.358754
2	10000	0.95	no	stochastic-gradient	23.878402	0.090869	30.502722	0.517216
3	10000	0.95	no	importance-sampling	23.805341	0.051533	30.360307	0.140216
4	100000	0.95	yes	stochastic-gradient	24.196353	0.011945	30.599172	0.014219
5	100000	0.95	yes	importance-sampling	24.016554	0.004534	30.469336	1.088522
6	100000	0.95	no	stochastic-gradient	24.588136	0.014264	30.395326	0.085318
7	100000	0.95	no	importance-sampling	24.554293	0.004154	30.332801	0.038515
8	500000	0.95	yes	stochastic-gradient	24.509240	0.002171	30.403581	0.002266
9	500000	0.95	yes	importance-sampling	24.445870	0.000736	30.362363	0.362121
10	500000	0.95	no	stochastic-gradient	24.597759	0.003733	30.354722	0.031293
11	500000	0.95	no	importance-sampling	24.590335	0.000887	30.277554	0.331285

FIGURE 1 – $\alpha = 0.95$

4.1 Position courte sur une option de vente

Nous reprenons exactement l'exemple 1 de l'article [1]. Cela nous permet au passage de vérifier les résultats de notre implémentation avec ceux de l'article sus-cité.

On vend un put de strike $K = 110$, de maturité $T = 1$ an sur une action suivant une dynamique Black-Scholes, de prix initial $S_0 = 100$ et de volatilité $\sigma = 20\%$. Le put est vendu au prix $P_0 = 10.7$. Le taux sans risque annuel est $r = 5\%$. La fonction de perte est donc donnée par :

$$\varphi(X) = (K - S_T)_+ - e^{rT}$$

avec

$$S_T = S_0 e^{(r - \frac{\sigma^2}{2})T + \sigma\sqrt{T}X}$$

où $X \sim \mathcal{N}(0, 1)$ sous la probabilité risque neutre.

On vérifie sans problème que les hypothèses du début de la section 3 sont vérifiées pour la loi normale réduite, avec les paramètres $b = 2$ et $\rho = \frac{1}{2}$ (pour une loi normale non réduite, il faudrait prendre $\rho = \frac{1}{2\sigma^2}$). Pour la fonction de perte, un contrôle exponentiel avec $a = 1$ suffit. On reporte dans les tableaux 1, 2 et 3 les résultats numériques pour différentes valeurs des paramètres et méthodes. On utilise toujours le même pas $\gamma_n = \frac{1}{n^{0.75}+100}$.

	Number of steps	alpha	averaging	Method	V@R	Var(V@R)	CV@R	Var(CV@R)
12	10000	0.99	yes	stochastic-gradient	32.760843	0.217475	41.165910	0.480798
13	10000	0.99	yes	importance-sampling	32.698907	0.134891	40.414738	2.205867
14	10000	0.99	no	stochastic-gradient	34.020310	0.304375	38.205156	1.351255
15	10000	0.99	no	importance-sampling	33.924106	0.176050	38.140476	0.103767
16	100000	0.99	yes	stochastic-gradient	33.901467	0.023705	38.492340	0.031132
17	100000	0.99	yes	importance-sampling	33.835331	0.005780	38.529282	0.055549
18	100000	0.99	no	stochastic-gradient	34.050170	0.042616	38.110977	0.248290
19	100000	0.99	no	importance-sampling	34.031145	0.009829	38.114205	0.187050
20	500000	0.99	yes	stochastic-gradient	34.027004	0.005962	38.228993	0.006647
21	500000	0.99	yes	importance-sampling	33.996851	0.001010	38.244195	0.000397
22	500000	0.99	no	stochastic-gradient	34.031567	0.013396	38.096409	0.077154
23	500000	0.99	no	importance-sampling	34.040830	0.002115	38.171775	0.014612

FIGURE 2 – $\alpha = 0.99$

	Number of steps	alpha	averaging	Method	V@R	Var(V@R)	CV@R	Var(CV@R)
24	10000	0.9995	yes	stochastic-gradient	46.438123	3.264472	53.574985	11.698141
25	10000	0.9995	yes	importance-sampling	45.533528	1.267177	50.341843	5.675744
26	10000	0.9995	no	stochastic-gradient	46.158809	4.097028	49.059447	13.717015
27	10000	0.9995	no	importance-sampling	45.493951	1.201583	48.206489	0.345016
28	100000	0.9995	yes	stochastic-gradient	45.708331	0.206818	48.895336	0.449039
29	100000	0.9995	yes	importance-sampling	45.378333	0.026816	48.408595	0.049518
30	100000	0.9995	no	stochastic-gradient	45.480652	0.594363	48.053455	2.563300
31	100000	0.9995	no	importance-sampling	45.408803	0.047069	48.053669	0.084336
32	500000	0.9995	yes	stochastic-gradient	45.461374	0.034640	48.230160	0.049624
33	500000	0.9995	yes	importance-sampling	45.385618	0.003179	48.167939	0.008991
34	500000	0.9995	no	stochastic-gradient	45.444934	0.206752	48.239249	0.745667
35	500000	0.9995	no	importance-sampling	45.412529	0.012260	48.083977	0.010205

FIGURE 3 – $\alpha = 0.9995$

Pour chaque ligne, on a fait tourner la même simulation 100 fois, et on a renseigné la moyenne statistique et la variance statistique des résultats dans les colonnes. On remarque que sans moyennisation, la méthode d'importance sampling fournit toujours une réduction de variante très importante. La moyennisation offre une réduction de variance supplémentaire seulement à partir de 100000 itérations : en effet, avec un nombre d'itérations trop faible, la moyennisation souffre des potentielles valeurs aberrantes des toutes premières itérations (phénomène visible dans les lignes du tableaux concernant les simulations à 10000 itérations).

Aussi de façon générale, l'effet de réduction de variance de l'importance sampling par rapport à l'algorithme naïf de gradient stochastique est beaucoup plus marqué sans moyennisation : mention spéciale à la simulation par importance sampling pour le calcul de la CV@R avec $\alpha = 0.9995$, avec 500000 itérations et sans moyennisation, qui offre une réduction de variance d'un facteur 74 par rapport à son équivalent pour l'algorithme naïf.

Enfin, la réduction de variance de l'importance sapling reste très efficace même avec un nombre d'itérations faible, ce qui était un des buts recherché : avoir une bonne précision sans devoir faire beaucoup d'itérations, dans le cas où la simulation de la fonction de perte est coûteuse.

Pour mettre en évidence des vitesses de convergence, on procède comme suit. On commence par fixer $\alpha = 0.99$. Puis, on estime $Z_\alpha^* = (\xi_\alpha^*, C_\alpha^*)$ en prenant la moyenne statistique sur 100 simulations de 500000 itérations de l'algorithme avec importance sampling, pour un pas $\gamma_n = \frac{1}{n^{0.75}+100}$. Enfin, pour deux méthodes (gradient sotchastique naïf avec moyennisation, importance sampling avec moyennisation), on trace la courbe :

$$\log(n) \longmapsto \log\left(\mathbb{E}[(Z_n - Z_\alpha^*)^2]\right)$$

pour des valeurs de $n \in [\![10000, 200000]\!]$, en progressant par pas de 10000, et en prenant à chaque fois 100 simulations pour le calcul de l'espérance. Ces courbes sont données en figures 4 et 5.

Les log-log plots sont bien affines, cependant on n'observe pas une pente égale à -1 ce qui devrait normalement être le cas d'après les théorèmes de convergence vus précédemment. Nous ne savons pas si cela vient de notre méthode pour évaluer la vitesse de convergence ou d'une autre raison.

4.2 Loi exponentielle

Nous avons voulu tester les deux algorithmes sur des variables exponentielles, notamment car on dispose de formules fermées pour comparer. On

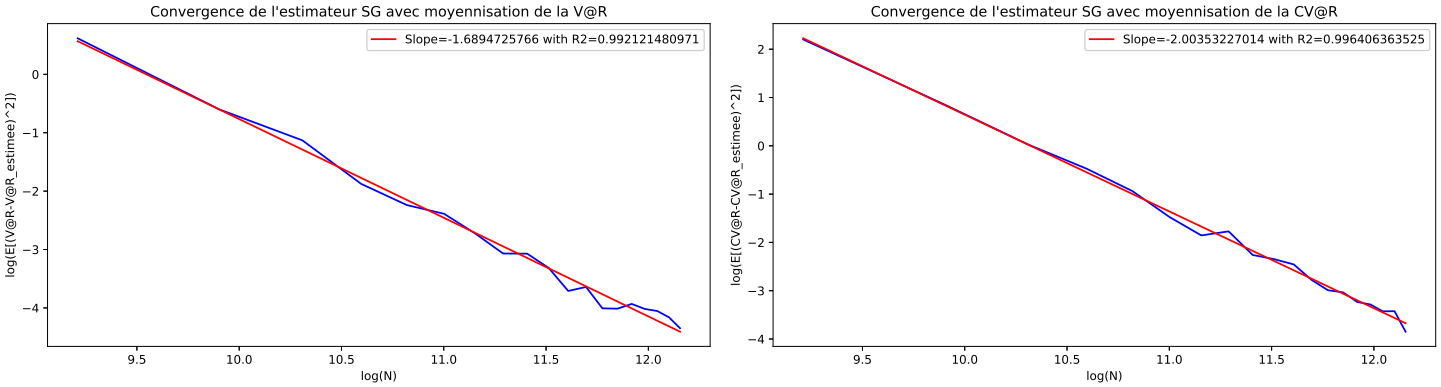

FIGURE 4 – Vitesses de convergence : algo naïf

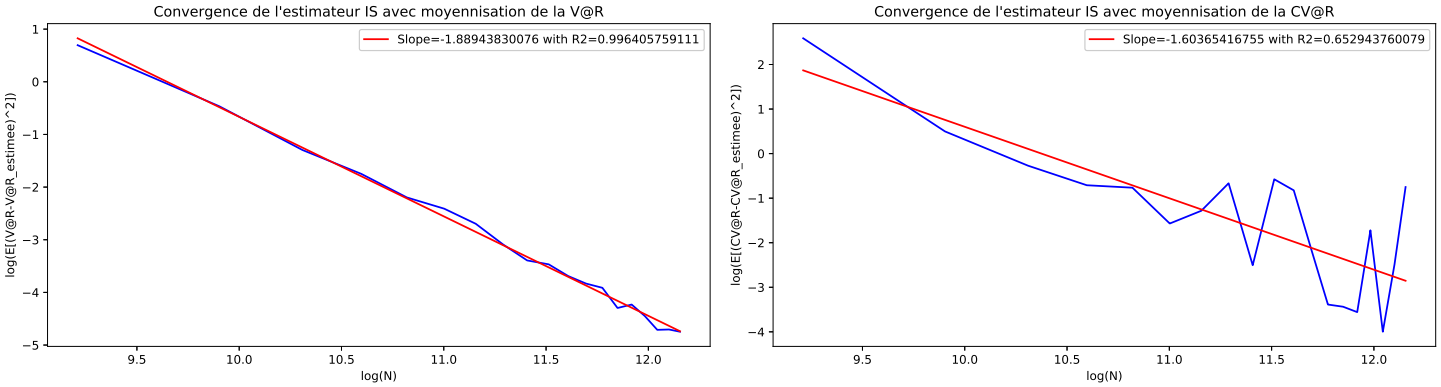

FIGURE 5 – Vitesses de convergence : importance sampling

prend donc $X \sim \mathcal{E}(\lambda)$, et notre fonction de perte est l'identité :

$$\varphi(X) = X$$

Pour la fonction de perte, on a un contrôle exponentiel avec $a = 1$. Il y a cependant un léger problème pour appliquer l'algorithme avec importance sampling pour une loi exponentielle : en effet la densité de probabilité $x \longmapsto \lambda e^{-\lambda x}\, 1_{x\geq 0}$ n'est pas presque sûrement strictement positive sur $\mathbb{R}$.

Pour pallier à ce problème, on considère une variable aléatoire Y distribuée selon une loi exponentielle symétrique, donc la densité de probabilité est donnée par :

$$\forall x \in \mathbb{R}, p(x) = \frac{\lambda}{2} e^{-|x|}$$

On remarque que pour une fonction F mesurable, on a :

$$\mathbb{E}[F(X)] = 2\mathbb{E}[F(Y)1_{Y\geq 0}] = \mathbb{E}(G(Y))$$

où on a posé $G(y) = 2F(y)\, 1_{y\geq 0}$. La loi de Y vérifie toutes les hypothèses de la section 3 avec les paramètres $b = 1$ et $\rho = \lambda$. On peut donc appliquer les résultats de cette section et on obtient d'une part :

$$\begin{aligned}
\forall \theta \geq 0, \mathbb{E}[F(X)] &= \mathbb{E}[G(Y)] \\
&= \mathbb{E}\left[G(Y+\theta)\frac{p(Y+\theta)}{p(Y)}\right] \\
&= \mathbb{E}\left[2F(Y+\theta)\, 1_{Y+\theta\geq 0}\frac{p(Y+\theta)}{p(Y)}\right] \\
&= \mathbb{E}\left[F(X+\theta)\, 1_{X+\theta\geq 0}\frac{p(X+\theta)}{p(X)}\right]
\end{aligned}$$

et d'autre part :

$$\begin{aligned}
\forall \theta \geq 0, \nabla Q(\theta) &= \mathbb{E}\left[G^2(Y-\theta)\frac{p^2(Y-\theta)}{p(Y)p(Y-2\theta)}\frac{\nabla p(Y-2\theta)}{p(Y-2\theta)}\right] \\
&= \mathbb{E}\left[4F^2(Y-\theta)\, 1_{Y-\theta\geq 0}\frac{p^2(Y-\theta)}{p(Y)p(Y-2\theta)}\frac{\nabla p(Y-2\theta)}{p(Y-2\theta)}\right] \\
&= \mathbb{E}\left[F^2(X-\theta)\, 1_{X-\theta\geq 0}\frac{2p^2(X-\theta)}{p(X)p(X-2\theta)}\frac{\nabla p(X-2\theta)}{p(X-2\theta)}\right]
\end{aligned}$$

et on peut donc appliquer les algorithmes de la section 3 en simulant effectivement une loi exponentielle (il suffit de modifier légèrement les fonctions L_i).

	Number of steps	alpha	averaging	Method	Analytic V@R	Analytic CV@R	V@R	Var(V@R)	CV@R	Var(CV@R)
0	10000	0.9995	yes	stochastic-gradient	3.80045	4.30045	7.854967	0.043501	8.535331	0.072340
1	10000	0.9995	yes	importance-sampling	3.80045	4.30045	10.107782	0.116194	10.372266	0.100459
2	10000	0.9995	no	stochastic-gradient	3.80045	4.30045	4.472599	0.231954	5.018492	0.473612
3	10000	0.9995	no	importance-sampling	3.80045	4.30045	6.868665	0.261685	7.360381	0.237758
4	100000	0.9995	yes	stochastic-gradient	3.80045	4.30045	4.371399	0.005641	4.899784	0.009945
5	100000	0.9995	yes	importance-sampling	3.80045	4.30045	3.906342	0.001709	4.387178	0.005345
6	100000	0.9995	no	stochastic-gradient	3.80045	4.30045	3.933281	0.094596	4.428647	0.110085
7	100000	0.9995	no	importance-sampling	3.80045	4.30045	3.815929	0.009484	4.306178	0.072225
8	500000	0.9995	yes	stochastic-gradient	3.80045	4.30045	3.956782	0.000831	4.463559	0.001912
9	500000	0.9995	yes	importance-sampling	3.80045	4.30045	3.810734	0.000311	4.306484	0.000907
10	500000	0.9995	no	stochastic-gradient	3.80045	4.30045	3.815118	0.021567	4.329908	0.023375
11	500000	0.9995	no	importance-sampling	3.80045	4.30045	3.809287	0.001899	4.304452	0.012831

FIGURE 6 – $\alpha = 0.9995$

Pour les simulations, on a fixé $\alpha = 0.9995$. Les résultats sont regroupés dans le tableau 6. Pour chaque ligne, on a fait tourner la même simulation 100 fois, et on a renseigné la moyenne statistique et la variance statistique des résultats dans les colonnes. On ajoute les valeurs données par les formules closes à titre de comparaison.

On fait les mêmes remarques que pour l'exemple précédent concernant la moyennisation. On remarque aussi que la réduction de variance de l'importance sampling est un peu plus faible par rapport à l'exemple précédent, probablement car la fonction de perte est plus simple (moments quadratiques plutôt qu'exponentiels dans l'exemple précédent).

Pour les vitesses de convergence, on trace les mêmes courbes que pour l'exemple précédent (toujours $\alpha = 0.99$), à ceci près qu'au lieu d'estimer au prélalable $(\xi_\alpha^*, C_\alpha^*)$, on prend les valeurs données par les formules closes. Ces courbes sont données en figures 7 et 8. Cette fois, on obtient des pentes proches de -1, ce qui est cohérent avec les théorèmes de la section 2 et 3.

5 Annexe - Implémentation en C++

Pour l'implémentation, nous avons choisi de paramétriser nos classes et fonctions par quatre paramètres `template` :

— `Phi` qui paramétrise le type d'un foncteur `* -> double`

— `Gamma` qui paramétrise le type d'un foncteur `int -> double`

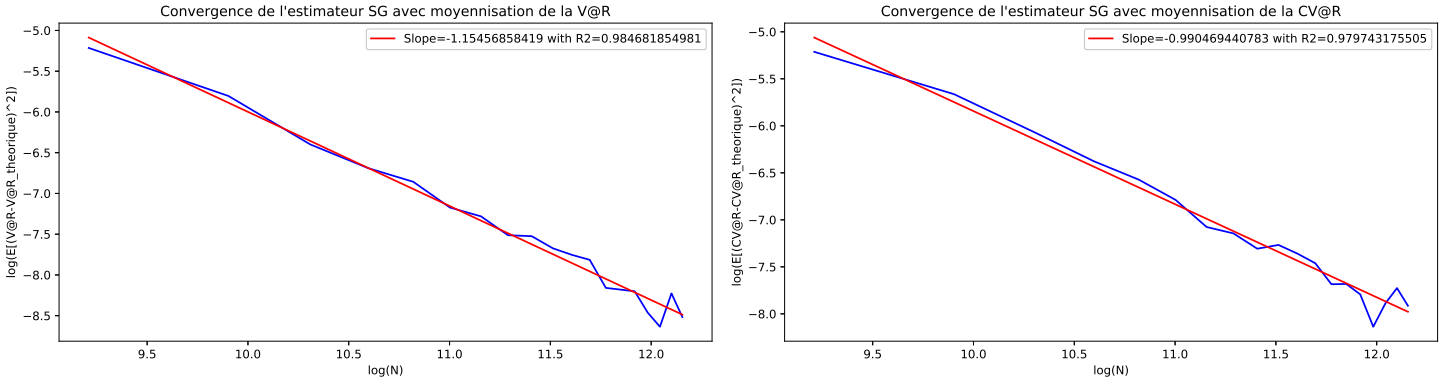

FIGURE 7 – Vitesses de convergence : algo naïf

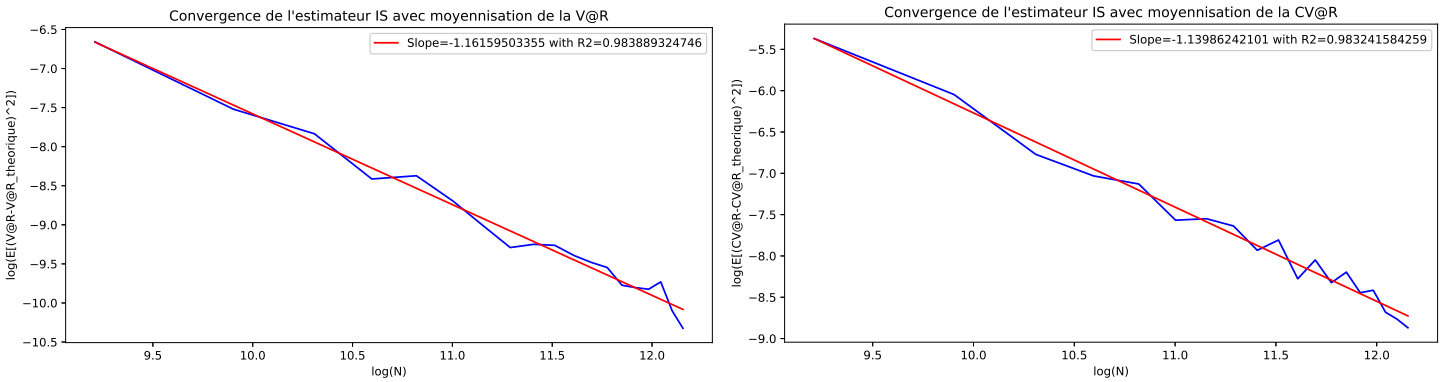

FIGURE 8 – Vitesses de convergence : importance sampling

— `Generator` qui paramétrise le type d'un générateur de nombre aléatoires
— `Distribution` qui paramétrise le type d'un foncteur `Generator -> *`

Par rapport aux notations des sections précédentes, le paramètre `Phi` représente le type de la fonction de perte φ, le paramètre `Gamma` représente le type de la suite (γ_n), et le paramètre `Distribution` représente le type de l'objet qui simule la loi de X. Ainsi, le type de retour du foncteur `Distribution` et le type d'entrée du foncteur `Phi` doivent être compatibles.

De façon générale, il est préférable d'utiliser des paramètres `template` pour les foncteurs à cause de la grande diversité d'objets se comportant comme des fonctions en C++ (fonction statiques, fonctions anonmyes avec ou sans environnement, surcharge de l'opérateur `()`...). Aussi, il n'y a donc pas besoin de paramétriser le type de retour de `Distribution` et le type d'entrée de `Phi` (qui correspondraient au type représentant l'espace des valeurs de X) : cela allège les classes et fonctions d'un paramètre `template`, mais au détriment d'erreurs de compilation plus lisibles dans le cas où les deux types ne sont pas compatibles pas par exemple.

Grâce à la paramétrisation décrite ci-dessus, la classe `approx_kernel` permettant de calculer la V@R et CV@R par l'algorithme de gradient stochastique naïf est directement utilisable avec n'importe quelle distribution aléatoire du standard C++11 (et n'importe quel générateur de nombres aléatoires) sans rien faire, de même pour la fonction de perte φ, sous réserve que lesdites distribution et fonction de perte vérifient bien sûr les hypothèses de la section 2. Cela est illustré par exemple dans les fichiers sources `short_put.cpp` pour la loi normale, et `exponential_distribution.cpp` pour la loi exponentielle.

Concernant la classe `IS_kernel` permettant de calculer la V@R et CV@R par l'algorithme avec importance sampling, bien qu'elle soit *a priori* elle aussi paramétrée de la même façon, elle n'est pas directement utilisable pour une distribution quelconque. En effet, le problème est qu'il faut renseigner les paramètres comme b, ρ ainsi que les fonctions faisant intervenir la densité de probabilité comme $(x, \theta) \longmapsto \frac{p(x+\theta)}{p(x)}$. La classe `importance_sampling_parameters` est là pour ça : il faudra renseigner à la main ces paramètres pour chaque nouvelle distribution. Nous l'avons fait pour la loi normale et la loi exponentielle. De même, il n'est pas possible d'utiliser directement n'importe quelle fonction de perte φ car il faut renseigner le paramètre de contrôle exponentiel a au moment du calcul. Ces limitations ne nous semblent pas pouvoir être levées, mais ne sont pas non plus trop gênantes, dans la mesure où il suffit par exemple de

spécialiser une fois pour toutes la classe `importance_sampling_parameters` pour chaque distribution souhaitée.

Un point fort supplémentaire de cette paramétrisation est qu'il est *a priori* possible d'utiliser n'importe quel type pour l'espace des valeurs de X de façon transparente (du moment que ce type possède des opérations +, -... etc), donc cela est très utile quand X est à valeurs vectorielles par exemple. Cependant nous n'avons pas illustré cette fonctionnalité dans nos exemples, qui utilisent uniquement des variables aléatoires à valeurs dans $\mathbb{R}$.

Une amélioration possible serait de paramétriser le type flottant utilisé, notre code se restreignant au type `double`.

Enfin, dans le cas où les différents types intervenant ne seraient pas connus à la compilation (entrés à l'exécution par un utilisateur via un système de scripting par exemple), il est très simple d'ajouter des mécanismes de type erasure par dessus pour contrevenir à ce problème, c'est pourquoi cette approche avec des paramètres `template` nous semble dans tous les cas supérieure à une approche dynamique.

Références

[1] Olivier Bardou, Noufel Frikha et G. Pagès. "Computation of VaR and CVaR using stochastic approximations and unconstrained importance sampling". In : *Monte Carlo Methods and Applications* 15 (2009).

[2] Olivier Bardou, Noufel Frikha et G. Pagès. "Estimation de la VaR et de la CVaR par algorithmes stochastiques". In : *Journées Jeunes Probabilistes et Statisticiens* (2010).