

Monte Carlo pour les nuls

Simon Léger

06 Avril 2006

Résumé

Le but de ce papier est de donner une introduction aux techniques de Monte Carlo utilisées en finance, essentiellement afin de pricer des produits assez complexes, dits exotiques. Ce papier s'adresse à un public non expérimenté et se veut d'approche assez générale bien qu'expliquant comment construire efficacement un tel produit en informatique et en donnant les méthodes permettant d'améliorer son efficacité.

En partant de l'origine du principe de Monte-Carlo, nous verrons comment ceci nous permet de pricer des produits financiers, puis comment implémenter un tel système. Enfin nous aborderons certaines techniques permettant d'en améliorer l'efficacité.

Table des matières

1	Origine	3
1.1	Application en géométrie	3
2	Cadre financier	5
2.1	Environnement de travail	5
2.2	Cas du Call classique	6
2.3	Résultats obtenus	7
2.4	Pourquoi celà marche-t-il ?	8
2.5	Option <i>Asiatique</i>	8
2.6	Cas général pour un asset	9
2.7	Cas général pour n assets	9
3	Comment construire son pricer Monte Carlo	10
3.1	Point de vue adopté	10
3.2	Générer les nombres aléatoires	10
3.2.1	Introduction au problème	10
3.2.2	Les différents types de générateurs	10
3.2.3	Passer d'uniforme à normal	12
3.3	Construire des vecteurs corrélés	12
4	Calcul des grecques	13
4.1	Que sont ces <i>grecques</i> ?	13
4.2	Approche simple	14
4.3	Méthodes plus avancées	15
4.3.1	Différenciation des chemins	15
4.3.2	Autres méthodes	15
5	Conclusion	15

Introduction

Pourquoi pour "les nuls" ?

Parce que cet article peut s'adresser à n'importe qui intéressé par cette technique et notamment à son utilisation en finance. Il se veut une sorte d'introduction complète, si tant est que cela puisse être possible... Par là j'entends qu'il va essayer de traiter un vaste domaine sans rentrer en détail dans les approfondissements, de nombreux articles précis pouvant être trouvés sur le net.

J'ai donc essentiellement écrit cet article pour les étudiants ou tout simplement pour les gens curieux et souhaitant découvrir ce formidable outil. Je suis moi-même un de ses grand fan et je dois dire que j'ai eu du mal à trouver des articles soit suffisamment simples pour que je puisse les comprendre, soit suffisamment développés pour pouvoir apprendre quelque chose de nouveau. J'espère que ce papier comblera ce manque.

Pour ceux qui seraient curieux d'en savoir plus sur moi je les invite à aller faire un tour sur mon site <http://homepages.nyu.edu/~s11544/index.html>. Vous pouvez également me contacter par mail à simon.leger@nyu.edu.

1 Origine

1.1 Application en géométrie

La technique dite de Monte-Carlo vient en fait de la géométrie et plus précisément du calcul d'aire. C'est une technique qui a été rendue possible grâce au développement d'ordinateurs de plus en plus puissants.

Prenons un exemple tout simple, par exemple cherchons à calculer la valeur du nombre π , ou de la même manière l'aire d'un cercle, qui est égale, nous le rappelons, à $\mathcal{A} = \pi * r^2$.

Pour cela, nous allons considérer un carré de côté 1, avec un quart de cercle inclus dedans comme sur la figure 1 ci dessous.

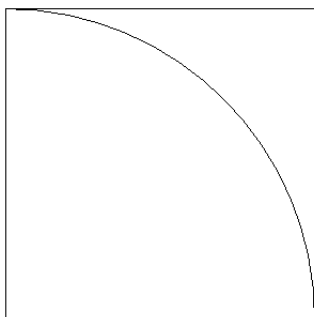


FIG. 1 – Cadre de l'exemple introductif

On va ensuite simuler des points aléatoires dans ce carré $[0,1]$, i.e. il faut simuler des couples de points (x,y) avec x et y suivant une loi uniforme sur $[0,1]$. Ceci veut dire que pour tous $a < b, c < d$ dans $[0,1]$,

$$\mathbb{P}(X \in [a, b], Y \in [c, d]) = (b - a) * (d - c)$$

Nous obtenons maintenant une figure similaire à celle ci avec seulement quelques points tirés au hasard :

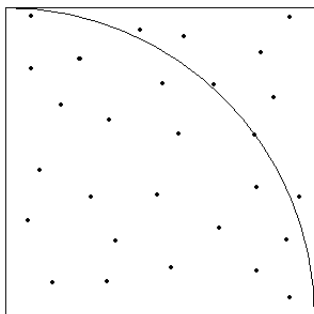


FIG. 2 – Mesure de l'espace occupé

Admettons que nous avons simulé 100 points, comment cela nous permet-il de déterminer l'aire du quart de cercle, et ainsi de déterminer une valeur approchée de π ? Si l'on réfléchit, l'espace de ce carré de côté 1 peut être rempli par une infinité de points et alors l'aire contenue dans le quart de cercle ne serait que le rapport de ces points au nombre total de points simulés qui remplissent l'aire du carré qui est de 1... On en déduit donc que l'aire du quart de cercle est en fait le nombre de points à l'intérieur rapporté au nombre de points total simulés. Si l'aire de notre carré était de 2 par exemple, il faudrait multiplier ce rapport par 2 également.

Certains vont se demander à ce stage comment compter le nombre de points à l'intérieur du cercle d'une manière pratique par exemple, et bien il suffit de se rappeler qu'un point de coordonnées (x,y) est dans le cercle centre de rayon r ssi $x^2 + y^2 < r$.

On voit donc ici comment se dessine l'algorithme :

```
double N=nombre max de simulations
double count=0
pour i=1 a N
  x=uniforme [0,1]
  y=uniforme [0,1]
  si (x^2+y^2<1) faire
    count=count+1
fin i
double result=count/N
```

On obtient ainsi l'aire de notre quart de cercle avec une assez bonne précision pour N assez grand (nous préciserons ceci plus tard...). En multipliant ce résultat par 4 on obtient une assez bonne mesure de π . Nous noterons ici que la vitesse de convergence de notre algorithme est directement liée à la performance de notre générateur de nombres aléatoires, i.e. on approchera beaucoup plus vite le résultat si les points tirés sont uniformément répartis dans le carré, en fait, nous pouvons même aller jusqu'à dire que ces points n'ont pas vraiment besoin d'avoir un caractère totalement aléatoire...

2 Cadre financier

2.1 Environnement de travail

Bon, maintenant que l'on a compris comment le principe de Monte Carlo pouvait s'appliquer au calcul d'aire, essayons de voir comment appliquer son principe en finance, et notamment au pricing de produits dérivés. Ce qu'il faut comprendre c'est que cette technique ne permet pas de prédire l'évolution du prix d'un actif, mais plutôt de la simuler lorsqu'on lui a déjà donné une forme précise, une équation de diffusion. Nous obtenons ainsi de très nombreux chemins sur lesquels nous pourrions appliquer notre fonction de prix du produit dérivé et ainsi, en supposant que chacun des ces chemins est équiprobable, moyenner ces prix et obtenir une approximation de notre prix du dérivé.

Tout ceci paraît bien théorique pour le moment, plaçons nous dans un cadre simple, l'équation de Black-Scholes.

Il s'agit du cas particulier de l'équation de diffusion classique :

$$dS_t = \mu(t, S_t)dt + \sigma(t, S_t)dW_t$$

où

$$\begin{cases} \mu(t, S_t) &= \mu S_t \\ \sigma(t, S_t) &= \sigma S_t \end{cases}$$

Le stock évolue ainsi selon une équation bien déterminée, il augmente au taux r et subit des chocs. La solution bien connue de cette équation est :

$$S_T = S_t * e^{(r - \frac{1}{2}\sigma^2)(T-t) + \sigma(W_T - W_t)}$$

où $W_T - W_t \rightsquigarrow \mathcal{N}(0, T-t)$. On voit donc que l'on peut réécrire cette solution sous la forme suivante :

$$S_T = S_t * e^{(r - \frac{1}{2}\sigma^2)(T-t) + X}$$

où $X \rightsquigarrow \mathcal{N}(0, \sigma^2(T-t))$. Si l'on sait tirer une $\mathcal{N}(0, 1)$, il suffit de la multiplier par $\sigma\sqrt{T-t}$ pour obtenir notre X . On arrive maintenant à simuler notre processus de diffusion de l'actif S_t .

2.2 Cas du Call classique

Essayons d'utiliser le cadre précédent afin de pricer un call *européen* classique dont le payoff est $[S_T - K]^+$ où K est le *Strike* du call choisi à l'origine du contrat et T est la *maturité*. Européen signifie que cette fonction de payoff ne peut être exercée qu'à la maturité T et pas avant comme ce serait le cas pour un Call dit *Américain*.

Regardons sur la figure ci dessous un exemple de simulation et regardons comment en déduire le prix de notre option. Notons que les trajectoires simulées ont rarement un aspect aussi régulier, mais il est difficile (même impossible en fait) de tracer un mouvement brownien (ceci est du au fait que la longueur de son chemin est en fait infini, quelque soient les intervalles sur lesquels on le regarde).

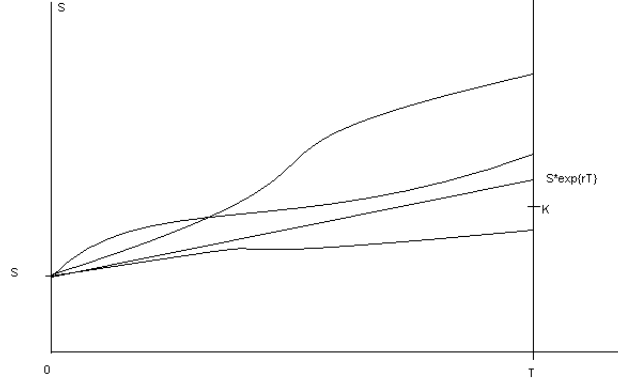


FIG. 3 – Simulation simple

Les chemins simulés ont ainsi une distribution lognormale et en espérance le prix S_T vérifie : $\mathbb{E}[S_T] = S_t e^{r(T-t)}$. Ceci résulte du fait que pour une variable aléatoire normale $X \rightsquigarrow \mathcal{N}(m, \sigma^2)$, on a :

$$\mathbb{E}_t(e^X) = e^{m + \frac{1}{2}\sigma^2}$$

ce qui donne ici :

$$\begin{cases} \mathbb{E}_t[S_T] &= S_t e^{(r - \frac{1}{2}\sigma^2)(T-t)} \mathbb{E}_t e^{\mathcal{N}(0, \sigma^2(T-t))} \\ &= S_t e^{(r - \frac{1}{2}\sigma^2)(T-t) + \frac{1}{2}\sigma^2(T-t)} \\ &= S_t e^{r(T-t)} \end{cases}$$

Nous obtenons donc un vecteur de possibilités de prix en T respectant la distribution supposée de l'actif, notons les $(S_T^1, S_T^2, \dots, S_T^N)$ et appliquons la fonction de payoff à chacune de ces valeurs, le vecteur de payoffs résultant est : $([S_T^1 - K]^+, [S_T^2 - K]^+, \dots, [S_T^N - K]^+)$ et le prix P de l'option est déterminé par :

$$P = \frac{\sum_{i=1}^N [S_T^i - K]^+}{N}$$

2.3 Résultats obtenus

Bon, maintenant que l'on sait pricer une option toute simple, on a envie de savoir si cela marche vraiment, et dans quelle mesure le prix obtenu colle-t-il à la réalité. Cet exemple n'a pas été choisi au hasard, il est particulièrement utile dans ce cas, puisqu'il existe une formule fermée, i.e. un prix théorique

du call sous les hypothèses choisies. Le prix du call P est donné par :

$$P = S_t \mathcal{N}(d_1) - K e^{-r(T-t)} \mathcal{N}(d_2)$$

où :

$$\begin{cases} d_1 &= \frac{\ln(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}} \\ d_2 &= d_1 - \sigma\sqrt{T-t} \end{cases}$$

Prenons un exemple simple dans lequel le prix théorique du Call est de 4.94387 et programmons un algorithme simple en C++ (à préférer à VBA par exemple pour sa rapidité, facteur crucial sur ce type d'applications). Nous utiliserons ici le générateur de nombres aléatoires `rand` du C++. Voici les prix et temps (en secondes) en fonction du nombre de simulations, ceci étant effectué sur un PC à 3Ghz :

	300k		1M		10M		50M	
Generator	Price	Time	Price	Time	Price	Time	Price	Time
rand C++	4.96	2.156	4.935	7.172	4.9432	70.98	4.9461	355.62

On voit ainsi que le prix converge assez rapidement sur ce type de produits simple, mais d'ailleurs pourquoi trouve-t-on le même prix ?

2.4 Pourquoi celà marche-t-il ?

Très bien, jusqu'ici on a commencé à voir plusieurs applications de ce principe qui semble assez intuitif, mais je ne vous ai pas encore dit pourquoi *celà marche*. Le principe repose en fait sur la loi des grands nombres : si X_1, X_2, \dots, X_N est une suite de variables aléatoires iid de loi X (i.e. si ce sont N tirages indépendants de loi X , alors :

$$\lim_{n \rightarrow \infty} \frac{X_1 + X_2 + \dots + X_n}{n} = \mathbb{E}[X]$$

Voilà, maintenant on est convaincu que ce principe marche (et surtout on sait pourquoi, et la satisfaction intellectuelle ça n'a pas de prix...), c'est-à-dire que pour un nombre suffisant de tirages, la moyenne des prix trouvés converge vers le prix théorique.

2.5 Option *Asiatique*

Voilà un autre exemple intéressant, mais pour d'autres raisons, car ici le principe de Monte Carlo prend toute son ampleur, bien qu'il existe d'autres méthodes pour ce type de produits. Pourquoi celà ? Parce que cette fois-ci, il n'existe pas de formule fermée donnant le prix d'un call asiatique contrairement au call européen. Tiens d'ailleurs qu'est ce qu'un call asiatique ? Le plus simple pour répondre est de donner la formule du payoff :

$$P = \left[\frac{1}{T-t} \int_t^T S_u du - K \right]^+$$

Ce produit permet donc d'éviter les fluctuations importantes à maturité en prenant en fait la moyenne du stock sur l'intervalle de temps considéré. En pratique on ne regarde pas vraiment l'intervalle exactement, ce qui poserait des problèmes de puissance de calcul, mais en fait un point par mois par exemple. C'est ce que nous allons considérer dans cette section.

Par analogie avec le cas du call européen, on voit bien comment résoudre notre problème grâce à une simulation Monte-Carlo, on va ainsi simuler des chemins suivis par le stock aux différentes dates considérées et y appliquer à nouveau notre payoff.

On sent bien ici qu'il va être nettement plus dur d'obtenir un prix très précis dû au grand nombre de possibilités de chemins... On se sent sans doute prêt maintenant à passer au cas général.

2.6 Cas général pour un asset

Le cas général lorsqu'on applique une option dépendant d'un seul asset pourrait se résumer ainsi : pouvez vous me donner le prix en t de l'option possédant la fonction de payoff suivante $f(S_t, t \leq T)$ où f sera appliquée en T uniquement et où l'on suppose un certain type de diffusion de l'asset ? Et la réponse est donc oui, et le processus de raisonnement est exactement similaire est très simple : simuler le stock aux dates nécessaires pour le calcul de f un très grand nombre de fois, appliquer la fonction f à chacun de ces chemins et moyenner les prix obtenus, ce qui nous donne le prix désiré. Le reste n'est *qu'un* problème de programmation...

2.7 Cas général pour n assets

Le problème ici est très similaire, la seule différence est qu'il faille simuler les valeurs des stocks aux différentes dates et qu'habituellement ces valeurs ne sont pas indépendantes (à partir du moment où on suppose une certaine forme de corrélation entre les assets). Finalement, au lieu de simuler n browniens indépendants, on veut simuler n browniens vérifiant :

$$\mathbb{E}[dW_i dW_j] = \rho_{ij} dt$$

Le *seul* problème se posant est donc un problème de simulation numérique. Comment à partir de simulations de valeurs de n browniens indépendants peut on simuler ces Browniens corrélés ? La réponse est simple, il faut juste utiliser une méthode numérique permettant de donner la matrice de décomposition de Cholesky, qui est une matrice triangulaire. On applique cette matrice (multiplication matricielle) à la matrice de base des Browniens simulés et on obtient maintenant une matrice de simulation de ces browniens corrélés. On peut vérifier la précision du résultat en calculant de manière simple la matrice de corrélation et en vérifiant le résultat avec la matrice théorique de corrélation d'où on est parti. On a ainsi une manière simple de

simuler les Browniens corrélés ce qui permet de passer assez simplement du cas de un asset au cas de n assets.

3 Comment construire son pricer Monte Carlo

3.1 Point de vue adopté

Le but de cet article n'étant pas d'expliquer pas à pas comment coder son pricer Monte Carlo, nous n'entrerons pas dans les détails de la programmation, tout le monde étant censé savoir comment utiliser les techniques d'héritage ou autres artifices...

Nous privilégierons donc les aspects théoriques de l'implémentation d'un tel pricer, tel que la génération de nombres aléatoires, le calcul des grecques et comment améliorer l'efficacité de ces calculs.

3.2 Générer les nombres aléatoires

3.2.1 Introduction au problème

Nous rentrons maintenant dans un des aspects les plus importants de la construction pratique d'un pricer Monte Carlo : la génération de nombres aléatoires. C'est en effet d'elle dont va dépendre l'efficacité de notre pricer, tant au niveau de la précision des résultats qu'au niveau de la rapidité des calculs.

Afin d'éclairer mes propos, rien de mieux qu'un tableau pour illustrer les résultats obtenus avec différents générateurs, nous reviendrons ensuite sur ces générateurs. Le prix exact du call dans cet exemple est de 4.94387 :

	300k		1M		10M		50M	
Generator	Price	Time	Price	Time	Price	Time	Price	Time
RandC	4.96	2.156	4.935	7.172	4.9432	70.98	4.9461	355.62
ParkMiller	4.987	1.968	4.962	6.532	4.9448	67.7	4.9446	324.09
MersenneTwister	4.936	1.984	4.949	6.547	4.9461	65.08	4.9435	325.73
Sobol	4.94354	1.95	4.94372	6.42	4.94385	64.26	4.94387	319.95

Le générateur de nombres aléatoires *RandC* est en fait le générateur par défaut du C++ de nombres uniformes. Comme on le voit celui-ci n'arrive pas avec 50 millions de simulations à atteindre l'efficacité de *Sobol* en seulement 300 000 simulations !

3.2.2 Les différents types de générateurs

Il existe tout d'abord deux principales catégories de générateurs de nombres aléatoires : les *pseudo* générateurs et les *quasi* générateurs, qui conduisent respectivement aux techniques de Monte Carlo et de *quasi* Monte Carlo. Les

premiers sont les plus simples à utiliser, puisque les générateurs sont supposés produire des nombres vraiment aléatoires selon un sens commun, ie on ne peut pas prédire le prochain nombre. Cependant, il est nécessaire à cette étape de signaler qu'il est de toute manière impossible de construire de vrais nombres aléatoires avec un code informatique. Ces générateurs sont en fait basés sur la congruence linéaire :

$$x_{n+1} = (ax_n + b) \bmod(m)$$

Ce type de générateurs produit une suite (pseudo aléatoire) de nombres compris entre 0 et $m - 1$ qui est donc de période m . En divisant les résultats obtenus par m , on obtient une suite de nombres répartis uniformément sur $[0,1]$.

Certains générateurs possèdent certaines améliorations basées sur ce principe permettant d'augmenter la période, ainsi le générateur très utilisé de Mersenne Twister a une période de 10^{6000} ! Il utilise 624 mots générateurs et est ainsi équidistribué dans 623 dimensions, ce qui peut être pratique dans le cas de nombreux assets.

L'autre type de générateurs de nombres aléatoires, auquel appartient *Sobol*, correspond aux générateurs de nombres quasi aléatoires. Ici le but est très différent, puisqu'il ne s'agit plus de générer des nombres vraiment aléatoires, mais plutôt de construire des nombres qui remplissent de manière uniforme l'intervalle $[0,1]$ (*low discrepancy sequences*), et ceci en dimension multiple également. On comprend tout à fait que c'est amplement suffisant pour ce que nous essayons de faire en repensant à l'exemple introductif traité en début de cet article. En effet, le calcul d'aire suppose que nous considérons de manière équivalente toutes les portions d'espace sur notre carré et peu importe le caractère vraiment aléatoire de ces nombres. Le principe est le même pour les applications financières, nous voulons explorer selon notre modèle toutes les possibilités suivies par notre asset sans privilégier une direction en particulier et ce type de générateurs est très bien adapté pour cela, comme nous pouvons le voir sur le tableau précédant qui met bien en évidence l'efficacité de ce générateur.

Le problème de ce type de générateurs (bien sûr il en faut un...) est la difficulté de leur implémentation et surtout de leur initialisation en dimensions multiples. Par exemple, Sobol peut s'initialiser jusqu'en dimension 600 en restant très efficace, et certaines personnes prétendent l'utiliser en dimension 6000, mais là il devient très difficile d'obtenir des informations afin de vérifier cela... Nous noterons également qu'il existe d'autres générateurs de nombres quasi uniformes qui peuvent être plus ou moins efficaces selon les contextes.

Notons que la vitesse de convergence de Monte Carlo dans le cas de nombres pseudo aléatoires est en $\mathcal{O}(n^{-1/2})$ tandis que dans le cas de quasi Monte Carlo la vitesse de convergence est en $\mathcal{O}(\log^d n/n)$ où d est le nombre de dimensions du problème.

3.2.3 Passer d'uniforme à normal

Maintenant que nous savons générer des nombres uniformes dans $[0,1]$, il nous reste à passer de cette loi à une loi normale $\mathcal{N}(0,1)$. En effet, si l'on souhaite ensuite passer d'une $\mathcal{N}(0,1)$ à une $\mathcal{N}(m,\sigma^2)$ il suffit d'utiliser la propriété suivante :

$$X \rightsquigarrow \mathcal{N}(0,1) \Rightarrow (m + \sigma X) \rightsquigarrow \mathcal{N}(m,\sigma^2)$$

La manière classique de réaliser cette inversion est d'utiliser l'algorithme de Box-Muller qui transforme deux uniformes en deux normales en même temps. Cependant cet algorithme n'est pas très bon, spécialement pour les générateurs quasi Monte Carlo puisqu'il abîme les propriétés de ces suites (ordre et uniformité), et de plus il est plus lent que l'algorithme d'inversion de Moro par exemple. Expliquons rapidement comment cet algorithme fonctionne : si l'on a x_1, x_2 deux variables uniformes, alors en appliquant :

$$\begin{cases} y_1 &= \sqrt{-2\log(x_1)}\cos(2\pi x_2) \\ y_2 &= \sqrt{-2\log(x_1)}\sin(2\pi x_2) \end{cases}$$

on obtient deux normales y_1, y_2 indépendantes.

Cet algorithme est basé sur celui de Beasley&Springer (1977), qui est très efficace, sauf pour les queues de la distribution, pour lesquelles Moro apporte une correction. Ces queues sont dans ce cas modélisées en utilisant les séries de Chebyshev tronquées. Habituellement, on utilise Beasley&Springer pour $|x| \leq 0.42$ et l'amélioration de Moro pour $|x| > 0.42$. Notons ici un fait pratique très important, la fonction `NORMSINV()` de Excel qui est supposée faire ce travail retourne des résultats très erronés pour des valeurs extrêmes en fin de queue, tandis que l'algorithme de Moro est très précis sur toute la distribution. De plus il n'est pas difficile de trouver un code source C++ pour cet algorithme sur internet.

3.3 Construire des vecteurs corrélés

Maintenant que nous savons générer une loi normale, il ne nous reste plus qu'à voir comment corrélérer plusieurs vecteurs aléatoires, afin de simuler les lois jointes, notamment utiles dans le cas de pricings de rainbow options.

Dans le cas de deux assets corrélés uniquement, il existe une manière simple de créer deux normales corrélées, si l'on a X et Y et que l'on souhaite les corrélérer avec une corrélation ρ il suffit de prendre X et $Z = \rho X + \sqrt{1 - \rho^2}Y$. On peut facilement vérifier que Z et X sont corrélés avec la bonne corrélation ρ et que ce sont toujours des normales standards.

Dans le cas de n assets que l'on souhaite corrélérer, il faut utiliser une méthode plus compliquée mais qui repose sur le même principe. Pour cela nous avons besoin comme input de la matrice de corrélation des n assets

que nous notons Σ . On utilise dans ce cas la décomposition de *Cholesky* en écrivant :

$$\Sigma = U^T U$$

où U est une matrice triangulaire inférieure. Alors si N_1, \dots, N_n sont n normales indépendantes,

$$\begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = U \begin{pmatrix} N_1 \\ \vdots \\ N_n \end{pmatrix}$$

les X_i sont maintenant n normales corrélés avec la matrice de corrélation Σ donnée. Il est conseillé de vérifier cette nouvelle matrice de corrélation obtenue afin d'être sûr qu'il n'y a pas d'erreur dans la décomposition et que l'on a simulé suffisamment de variables pour obtenir une corrélation suffisamment proche de la valeur théorique.

On est maintenant capable de simuler conjointement autant de variables qu'on le souhaite et donc de pricer à priori n'importe quelle option ou produit exotique. Une fois ce travail achevé, il reste une étape importante, qui est celle de connaître ses risques, et de savoir les couvrir. Pour cela, il est nécessaire de calculer les grecques relatives au produit considéré.

4 Calcul des grecques

4.1 Que sont ces grecques ?

Ces grecques sont des valeurs numériques qui indiquent la sensibilité du prix du produit considéré à la date courante par rapport à l'un des inputs de ce prix.

Par exemple, dans le cas simple d'un Call classique, le *delta* est la dérivée du prix maintenant de l'option par rapport à la valeur actuelle du sous-jacent. Citons les principales grecques :

- le delta (δ) : c'est la dérivée du prix de l'option par rapport au prix du sous-jacent.
- le gamma (γ) : c'est la dérivée seconde du prix de l'option par rapport au prix du sous-jacent.
- le vega : c'est la dérivée du prix de l'option par rapport à la volatilité.
- le theta (θ) : c'est la dérivée du prix de l'option par rapport au temps.
- le rho (ρ) : c'est la dérivée du prix de l'option par rapport au taux d'intérêt.

On peut bien sûr imaginer toutes les autres grecques que l'on souhaite. Par exemple dans le cas de rainbow options, on peut calculer la grecque relative à la corrélation entre les stocks et toutes les grecques partielles que l'on peut imaginer...

Dans le cas où l'on a une formule fermée pour le prix de l'option, on peut calculer la valeur des grecques de manière analytique en dérivant ce prix. Ce qui nous intéresse ici, c'est comment estimer ces grecques lorsque l'on a pas de formule pour le prix de l'option ?

4.2 Approche simple

Dans ce cas, on utilise à nouveau notre pricer Monte Carlo, et il n'y a pas grand chose à rajouter pour obtenir ces grecques. On utilise alors la méthode des différences finies. Par exemple, dans le cas du delta, on utilise la formule suivante :

$$\delta_P = \frac{P(S_0 + \epsilon) - P(S_0 - \epsilon)}{2\epsilon}$$

et dans le cas du gamma par exemple :

$$\gamma_P = \frac{P(S_0 + \epsilon) + P(S_0 - \epsilon) - 2P(S_0)}{\epsilon^2}$$

Ou mieux afin d'éliminer les termes d'ordre 2 et 3 :

$$\gamma_P = \frac{-P(S_0 + 2\epsilon) + 16P(S_0 + \epsilon) - 30P(S_0) + 16P(S_0 - \epsilon) - P(S_0 - 2\epsilon)}{12\epsilon^2}$$

Maintenant, comment mener à bien ce calcul en pratique ? Tout d'abord, un moyen de calculer efficacement ces grecques est d'utiliser le même chemin pour pricer l'option et l'option *décalée*. En effet, la différence entre les deux prix obtenus va être très faible et si l'on utilise des chemins différents, l'erreur due à l'approximation de Monte Carlo va l'emporter sur la différence de prix qui va passer inaperçue tandis que si l'on utilise les mêmes chemins, la seule différence de prix est celle due à la différence dans les inputs, et c'est donc la sensibilité que l'on cherche à calculer, et la vitesse de convergence des grecques en est grandement améliorée.

Ce que l'on fait en termes mathématiques revient à la formulation suivante. Supposons que notre prix est une fonction d'un certain paramètre θ et que l'on cherche à calculer la sensibilité par rapport à ce paramètre :

$$\left\{ \begin{array}{l} \frac{\partial}{\partial \theta} \mathbb{E}^{\mathbb{Q}}(f(Y(\theta)) | \mathcal{F}_{T_0}) \simeq \frac{\partial}{\partial \theta} \hat{\mathbb{E}}^{\mathbb{Q}}(f(Y(\theta)) | \mathcal{F}_{T_0}) \\ \simeq \frac{1}{2\epsilon} \left(\hat{\mathbb{E}}^{\mathbb{Q}}(f(Y(\theta + \epsilon)) | \mathcal{F}_{T_0}) - \hat{\mathbb{E}}^{\mathbb{Q}}(f(Y(\theta - \epsilon)) | \mathcal{F}_{T_0}) \right) \\ = \frac{1}{n} \sum_{i=1}^n \frac{1}{2\epsilon} (f(Y(\omega_i, \theta + \epsilon)) - f(Y(\omega_i, \theta - \epsilon))) \end{array} \right.$$

Si ϵ n'est pas suffisamment petit, on n'est pas sûr de converger vers la valeur exacte, même avec un grand nombre de simulations. Si ϵ est assez petit, la convergence peut parfois être lente due à l'irrégularité de la fonction de payoff. Nous allons donc voir des méthodes plus évoluées pour calculer ces grecques.

4.3 Méthodes plus avancées

4.3.1 Différenciation des chemins

En repartant du calcul précédant, on peut le mener à bout de la façon différente suivante :

$$\left\{ \begin{array}{l} \frac{\partial}{\partial \theta} \mathbb{E}^{\mathbb{Q}}(f(Y(\theta)) | \mathcal{F}_{T_0}) \simeq \frac{\partial}{\partial \theta} \hat{\mathbb{E}}^{\mathbb{Q}}(f(Y(\theta)) | \mathcal{F}_{T_0}) \\ = \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial \theta} (f(Y(\omega_i, \theta))) = \frac{1}{n} \sum_{i=1}^n (f'(Y(\omega_i, \theta))) \frac{\partial Y(\omega_i, \theta)}{\partial \theta} \end{array} \right.$$

Ceci nécessite cependant plus d'information sur le payoff : f doit être dérivable, ainsi que sur la nature de θ . Ce calcul peut être mené à bout d'une façon très similaire :

$$\left\{ \begin{array}{l} \frac{\partial}{\partial \theta} \mathbb{E}^{\mathbb{Q}}(f(Y(\theta)) | \mathcal{F}_{T_0}) = \frac{\partial}{\partial \theta} \int_{\Omega} f(Y(\omega, \theta)) d\mathbb{Q}(\omega) = \int_{\Omega} \frac{\partial}{\partial \theta} f(Y(\omega, \theta)) d\mathbb{Q}(\omega) \\ = \int_{\Omega} f'(Y(\omega, \theta)) \frac{\partial Y(\omega, \theta)}{\partial \theta} = \mathbb{E}^{\mathbb{Q}} \left(f'(Y(\theta)) \frac{\partial Y(\omega, \theta)}{\partial \theta} | \mathcal{F}_{T_0} \right) \\ \simeq \hat{\mathbb{E}}^{\mathbb{Q}} \left(f'(Y(\theta)) \frac{\partial Y(\omega, \theta)}{\partial \theta} | \mathcal{F}_{T_0} \right) = \frac{1}{n} \sum_{i=1}^n (f'(Y(\omega_i, \theta))) \frac{\partial Y(\omega_i, \theta)}{\partial \theta} \end{array} \right.$$

L'avantage de cette formulation est de voir que les payoffs discontinus peuvent être traités de cette manière, en interprétant f comme une distribution.

4.3.2 Autres méthodes

Des méthodes plus complexes existent et sont plus efficaces dans certains cas, comme la méthode de réduction de variance, ou le calcul de Malliavin, qui dépassent le cadre de cet article, et dont une riche documentation peut être trouvée sur Internet, donc nous ne nous étendrons pas sur ces procédés.

5 Conclusion

Nous avons abordé de nombreux aspects du principe de Monte Carlo appliqué à la finance. Nous avons tout d'abord essayé de comprendre quel était le principe de cet outil, puis nous avons vu comment l'appliquer au pricing d'options en finance et comment réaliser un pricer basé sur ce principe étape par étape. Enfin, nous avons introduit les grecques et détaillé une manière simple de les calculer. Nous avons indiqué les autres améliorations qui peuvent être apportées, mais sans rentrer dans des démonstrations trop mathématiques, le lecteur curieux saura se reporter à la littérature très abondante dans ce domaine.