Conception et Implémentation d'un pricer Monte-Carlo générique en C++

Version finale

Jérôme Lelong

3 avril 2013

Le but de cette deuxième partie du cours de "Modélisation et Programmation" est de réfléchir à la structure d'un outil permettant de calculer des prix d'options par une méthode de Monte Carlo ainsi que leur dérivée par rapport au spot, puis de l'implémenter en C++.

La section 1 présente le problème auquel on s'intéresse et les motivations pratiques. La dynamique choisie pour modéliser l'évolution des cours des actifs (les sous–jacents) est détaillée dans la section 2. La section 3 vous propose une architecture possible pour le pricer. L'exécutable développé devra lire les paramètres des produits dans un fichier texte comme expliqué à la section 5. Finalement, vous trouverez en section 6 la liste des produits que votre pricer devra prendre en charge.

1 Motivations pratiques

Lorsque l'on s'intéresse à un produit dérivé, il est important de pouvoir calculer son prix et de savoir construire son portefeuille de couverture.

Notons $S = (S_t,\ t \geq 0)$ la dynamique du sous-jacent et $\mathcal{F}$ sa filtration naturelle, en pratique nous nous restreindrons au cas du modèle de Black Scholes multidimensionnel (c.f. section 2.2). Pour simplifier les choses, nous allons supposer que le payoff des options que nous allons considérer s'écrit sous la forme

$$\varphi(S_{t_0}, S_{t_1}, \ldots, S_{t_N})$$

où $0 = t_0 < t_1 < \cdots < t_N = T$ est une grille de dates de constatation non forcément équi–réparties.

Exemple. *Voici quelques exemples de fonctions φ dans le cas d'un sous-jacent unique.*

- $N = 1$ *et* $\varphi(S_0, S_1) = (S_1 - K)_+$ *correspond à une option d'achat*
- $T = 1$, $N = 12$ *et* $\varphi(S_0, S_1, \ldots, S_{12}) = (S_{12} - K)_+ \mathbf{1}_{\{\forall 0 \leq i \leq 12,\, S_i \geq L\}}$ *correspond à une option d'achat barrière à monitoring discret (une date de constatation par mois). L'option est annulée si à une des dates de constatation, la valeur du sous-jacent est inférieure à L.*
- $T = 1$, $N = 365$ *et* $\varphi(S_0, \ldots, S_{365}) = (\frac{1}{365}\sum_{i=0}^{364} S_i - K)_+$ *correspond à une option d'achat asiatique où la moyenne continue du sous-jacent $\frac{1}{T}\int_0^T S_u du$ a été remplacée par une moyenne discrète.*

Notons r le taux d'intérêt instantané que nous supposerons constant dans ce projet. Le prix à l'instant $0 \leq t \leq T$, avec $t_i \leq t < t_{i+1}$ d'une telle option est donné par

$$v(t, S_{t_0}, \ldots, S_{t_i}, S_t) = \mathrm{e}^{-r(T-t)}\, \mathbb{E}(\varphi(S_{t_0}, S_{t_1}, \ldots, S_{t_N})|\mathcal{F}_t)$$

Cette quantité sera calculée par une méthode de Monte Carlo.

Prix à l'instant 0 Attardons nous un instant sur le cas particulier du calcul du prix à l'instant 0

$$v(0, S_0) = \mathrm{e}^{-rT}\, \mathbb{E}(\varphi(S_{t_0}, S_{t_1}, \ldots, S_{t_N}))$$

On approche $v(0, S_0)$ par

$$\mathrm{e}^{-rT}\, \frac{1}{M} \sum_{j=1}^{M} \varphi(S_{t_0}^{(j)}, S_{t_1}^{(j)}, \ldots, S_{t_N}^{(j)})$$

où les $N-$uplets $(S_{t_0}^{(j)}, S_{t_1}^{(j)}, \ldots, S_{t_N}^{(j)})$ pour $j = 1, \cdots, M$ sont i.i.d selon la loi de $(S_{t_0}, S_{t_1}, \ldots, S_{t_N})$.

Prix à un instant $t > 0$ Le calcul du prix à l'instant t est légèrement plus compliqué car il faut calculer une espérance conditionnelle. Lorsque cela est possible, une manière de traiter le calcul de cette espérance conditionnelle est de réécrire le sous-jacent S_{t+u} à l'instant $t+u$ pour $u > 0$ en fonction de la valeur S_t d'un sous–jacent à l'instant t et d'une quantité indépendante de $\mathcal{F}_t$. Nous verrons dans la suite que cela est en particulier possible dans le modèle de Black–Scholes et que l'on peut écrire

$$S_{t+u} = S_t \tilde{S}_u$$

où $\tilde{S}$ est un processus indépendant de $\mathcal{F}_t$, ie. indépendant du passé jusqu'à l'instant t inclus. Ainsi le prix à l'instant t se réécrit

$$v(t, S_{t_0}, \ldots, S_{t_i}, S_t) = \mathrm{e}^{-r(T-t)}\, \mathbb{E}(\varphi(s_{t_0}, \ldots, s_{t_i}, s_t\tilde{S}_{t_{i+1}-t}, \ldots, s_t\tilde{S}_{t_N-t}))\Big|\begin{array}{l} s_{t_k} = S_{t_k},\ k = 0, \cdots, i \\ s_t = S_t \end{array} \tag{1}$$

Les lettres s minuscules désignent des variables déterministes. On peut alors approcher le prix à l'instant t par la moyenne Monte Carlo suivante

$$\mathrm{e}^{-r(T-t)}\, \frac{1}{M} \sum_{j=1}^{M} \varphi(s_{t_0}, s_{t_1}, \ldots, s_{t_i}, s_t\tilde{S}_{t_{i+1}-t}^{(j)}, \ldots, s_t\tilde{S}_{t_N-t}^{(j)}) \tag{2}$$

où les $N-$uplets $(\tilde{S}_{t_{i+1}-t}^{(j)}, \ldots, \tilde{S}_{t_N-t}^{(j)})$ sont i.i.d selon la loi de $(\tilde{S}_{t_{i+1}-t}, \ldots, \tilde{S}_{t_N-t})$. Les quantités $(s_{t_0}, s_{t_1}, \ldots, s_{t_i}, s_t)$ représentent les valeurs réellement observées jusqu'à l'instant t sur la trajectoire du sous-jacent sur laquelle on souhaite calculer le prix de l'option. D'un point du vue pratique, les valeurs $(s_{t_0}, s_{t_1}, \ldots, s_{t_i}, s_t)$ sont les cotations du sous-jacent observées sur le marché jusqu'à la date t.

2 Modèle de Black Scholes

2.1 Cas de la dimension 1

Considérons un modèle de Black–Scholes pour modéliser l'évolution d'un sous–jacent

$$S_t = S_0 \, \mathrm{e}^{(r-\sigma^2/2)t+\sigma B_t}$$

où B est un mouvement Brownien standard réel, $r > 0$ le taux d'intérêt instantané supposé constant et $\sigma > 0$ la volatilité du sous–jacent. Remarquons que pour tout $t, u \geq 0$, on peut écrire

$$S_{t+u} = S_t \, \mathrm{e}^{(r-\sigma^2/2)u+\sigma(B_{t+u}-B_t)} = S_t \tilde{S}_u \tag{3}$$

où $\tilde{S}$ est **indépendant** de $\mathcal{F}_t$ et la dynamique de $\tilde{S}$ est donnée par

$$\tilde{S}_u = \mathrm{e}^{(r-\sigma^2/2)u+\sigma\tilde{B}_u}$$

où $\tilde{B}$ est un mouvement Brownien réel standard indépendant de $\mathcal{F}_t$. Remarquons que $\tilde{S}$ suit la même dynamique que S mais a pour valeur initiale 1. La discrétisation de S sur la grille $t_0, t_1, \ldots, t_N$ s'écrit alors

$$S_{t_{i+1}} \overset{Loi}{=} S_{t_i} \, \mathrm{e}^{(r-\sigma^2/2)(t_{i+1}-t_i)+\sigma\sqrt{t_{i+1}-t_i}G_{i+1}}$$

où la suite $(G_i)_{i\geq 1}$ est une suite i.i.d. selon la loi normale centrée réduite.

2.2 Cas multidimensionnel

Considérons D actifs évoluant chacun selon un modèle de Black Scholes de dimension 1 et corrélés entre eux. La dynamique S de ces D actifs s'écrit

$$S_t^d = S_0^d \, \mathrm{e}^{(r-(\sigma^d)^2/2)t+\sigma^d B_t^d}, \quad d = 1 \ldots D \tag{4}$$

où r est le taux d'intérêt, $(\sigma^1, \ldots, \sigma^D)$ sont les volatilités de chacun des actifs et $B = (B^1, \ldots, B^D)^T$ est un vecteur de D mouvements Browniens standards et réels de matrice de corrélation $\mathrm{Cov}(B_t, B_t) = \Gamma t$ définie par $\mathrm{Cov}(B_t^i, B_t^j) = \Gamma_{ij} t$ avec $\Gamma_{ij} = \rho$ pour tout $i \neq j$ et $\Gamma_{ii} = 1$. Le paramètre ρ doit être choisi dans $]-\frac{1}{D-1}, 1[$ de telle sorte que Γ soit définie positive pour assurer que le marché soit complet. Attention B n'est pas un mouvement Brownien standard à valeurs dans $\mathbb{R}^D$ car ses composantes ne sont pas indépendantes mais on peut le réécrire en utilisant un mouvement Brownien standard à valeurs dans $\mathbb{R}^D$, noté dans la suite $W = (W^1, \ldots, W^d)^T$ et qui sera vu comme un vecteur colonne. Ainsi on a l'égalité en loi $(B_t, t \geq 0) = (LW_t, t \geq 0)$ pour tout $t \geq 0$ où L est la factorisée de Cholesky de la matrice Γ, .i.e $\Gamma = L'L$ avec L triangulaire inférieure. L'équation (4) se réécrit alors

$$S_t^d = S_0^d \, \mathrm{e}^{(r-(\sigma^d)^2/2)t+\sigma^d L^d W_t}, \quad d = 1 \ldots D$$

où L^d est la ligne d de la matrice L. Ainsi la quantité $L^d W_t$ est bien un réel. De cette équation, on remarque facilement que l'on peut déduire

$$\begin{aligned} S_{t_{i+1}}^d &= S_{t_i}^d \, \mathrm{e}^{(r-(\sigma^d)^2/2)(t_{i+1}-t_i)+\sigma^d L^d(W_{t_{i+1}}-W_{t_i})}, \quad d = 1 \ldots D \\ S_{t_{i+1}}^d &\overset{Loi}{=} S_{t_i}^d \, \mathrm{e}^{(r-(\sigma^d)^2/2)(t_{i+1}-t_i)+\sigma^d\sqrt{(t_{i+1}-t_i)}L^d G_{i+1}}, \quad d = 1 \ldots D \end{aligned}$$

où la suite $(G_i)_{i\geq 1}$ une suite i.i.d de vecteurs gaussiens centrés de matrice de covariance identité. Ainsi pour simuler le processus S sur la grille $(t_i)_{i=0,\dots,N}$ il suffit de savoir simuler des vecteurs gaussiens centrés de matrice de covariance identité.

Vous avez vu en cours de *Processus Stochastique* qu'il était possible de construire un portefeuille de couverture pour les options d'achat et de vente dans le modèle de Black Scholes en dimension 1. Ce portefeuille de couverture est entièrement déterminé par la quantité d'actifs risqués à posséder à chaque instant t qui est donnée par la dérivée du prix par rapport à S_t, i.e

$$\frac{\partial v(t, S_{t_0}, \cdots, S_{t_i}, S_t)}{\partial S_t}$$

Cette théorie est en fait valable bien au delà des options d'achat et de vente, elle est connue sous le nom de *couverture en delta*. Dans ce projet, nous l'appliquerons à d'autres produits bien plus complexes. Ces quantités sont connues sous le nom de delta de l'option et sont en pratique approchées par une méthode de différences finies

$$\frac{\mathrm{e}^{-r(T-t)}}{M 2 s_t h} \sum_{j=1}^{M} \Big(\varphi(s_{t_0}, s_{t_1}, \dots, s_{t_i}, s_t(1+h)\tilde{S}^{(j)}_{t_{i+1}-t}, \dots, s_t(1+h)\tilde{S}^{(j)}_{t_N-t}) \\ - \varphi(s_{t_0}, s_{t_1}, \dots, s_{t_i}, s_t(1-h)\tilde{S}^{(j)}_{t_{i+1}-t}, \dots, s_t(1-h)\tilde{S}^{(j)}_{t_N-t}) \Big) \tag{5}$$

où les $N-$uplets $(\tilde{S}^{(j)}_{t_{i+1}-t}, \dots, \tilde{S}^{(j)}_{t_N-t})$ sont i.i.d selon la loi de $(S_{t_{i+1}-t}, \dots, S_{t_N-t})$.

Pour un produit faisant intervenir D sous-jacents, son portefeuille de couverture contient les D actifs et la quantité d'actifs d à détenir à l'instant t est donnée par

$$\frac{\partial v(t, S_{t_0}, \cdots, S_{t_i}, S_t)}{\partial S^d_t}$$

3 Architecture du Pricer Monte Carlo

3.1 Calcul des prix

L'ambition de cette partie est d'écrire un outil permettant de pricer n'importe quel produit dans un modèle de Black Scholes de dimension 1 ou plus en ne spécifiant que 2 choses : le payoff de l'option et la manière de générer une trajectoire du modèle.

Votre pricer devra contenir les classes suivantes

```
class BS {
    int size_; /*! nombre d'actifs du modèle */
    double r_; /*! taux d'intérêt */
    double rho_; /*! paramètre de corrélation */
    PnlVect *sigma_; /*! vecteur de volatilités */
    PnlVect *spot_; /*! valeurs initiales du sous-jacent */
    /**
    * Génère une trajectoire du modèle et la stocke dans path
    *
    * @param path (output) contient une trajectoire du modèle. C'est une matrice
    * de taille d x (N+1)
    * @param T (input) maturité
```

```
    * @param N nombre de dates de constatation
    */
    void asset (PnlMat *path, double T, int N, PnlRng *rng) ;
}

class Option {
    double T_; /*! maturité */
    int TimeSteps_; /*! nombre de pas de temps de discrétisation */
    int size_; /* dimension du modèle, redondant avec BS::size_ */
    /**
    * Calcule la valeur du payoff sur la trajectoire passée en paramètre
    *
    * @param path (input) est une matrice de taille d x (N+1) contenant une
    * trajectoire du modèle telle que créée par la fonction asset.
    * @return phi(trajectoire)
    */
    virtual double payoff (const PnlMat *path) = 0;
}

class MonteCarlo {
    BS *mod_; /*! pointeur vers le modèle */
    Option *opt_; /*! pointeur sur l'option */
    PnlRng *rng; /*! pointeur sur le générateur */
    double h_; /*! pas de différence finie */
    int samples_; /*! nombre de tirages Monte Carlo */
}
```

De la classe `Option` vont dériver toutes les classes implémentant une option particulière. La liste des produits à implémenter se trouve Section 6. Le paramètre `rng` est un générateur de nombres aléatoires qui doit être créé (avec `pnl_rng_create`) et initialisé (avec `pnl_rng_sseed`) dans la fonction `main`.

Prix à l'instant 0 Dans un premier temps, on se limitera au calcul du prix à l'instant 0. Implémenter la méthode `price` à la classe `MonteCarlo` permettant de calculer par une méthode de Monte Carlo le prix à l'instant 0 d'une option. Cette fonction calculera également la largeur de l'intervalle de confiance de l'estimateur à 95%. On rappelle que la variance de l'estimateur peut être approchée par

$$\xi_M^2 = \mathrm{e}^{-2rT}\left[\frac{1}{M}\sum_{j=1}^{M}(\varphi(S_{t_0}^{(j)}, S_{t_1}^{(j)}, \ldots, S_{t_N}^{(j)}))^2 - \left(\frac{1}{M}\sum_{j=1}^{M}\varphi(S_{t_0}^{(j)}, S_{t_1}^{(j)}, \ldots, S_{t_N}^{(j)})\right)^2\right]$$

et que l'intervalle de confiance à 95% est alors donné par

$$\left[\mathrm{e}^{-rT}\,\frac{1}{M}\sum_{j=1}^{M}\varphi(S_{t_0}^{(j)}, S_{t_1}^{(j)}, \ldots, S_{t_N}^{(j)}) \pm \frac{1.96\xi_M}{\sqrt{M}}\right]$$

```
/**
* Calcule du prix de l'option à la date 0
*
* @param prix (ouptut) contient le prix
* @param ic (ouptut) contient la largeur de l'intervalle de confiance
*                   sur le calcul du prix
*/
void price (double &prix, double &ic);
```

Cette méthode `price` devra faire appel aux méthodes `asset` et `payoff`.

Prix à un instant $t > 0$ Une fois le calcul du prix à l'instant 0 testé et validé, vous pourrez passer au calcul du prix à tout instant $t > 0$.

Pour ce faire, il faut être capable de calculer l'espérance définie par l'équation (1) en utilisant l'estimateur (2). Cette espérance peut-être calculée par une méthode de Monte Carlo standard pourvu que l'on soit capable de générer des trajectoires i.i.d selon la loi conditionnelle de S sachant $\mathcal{F}_t$, i.e. grâce à la propriété (3) des trajectoires de la forme

$$(s_{t_0}, s_{t_1}, \ldots, s_{t_i}, s_t \tilde{S}^{(j)}_{t_{i+1}-t}, \ldots, s_t \tilde{S}^{(j)}_{t_N-t})$$

Ajouter à la classe `BS` la méthode

```
/**
* Calcule une trajectoire du sous-jacent connaissant le
* passé jusqu' à la date t
*
* @param path (output) contient une trajectoire du sous-jacent
* donnée jusqu'à l'instant t par la matrice past
* @param t (input) date jusqu'à laquelle on connait la trajectoire
* t n'est pas forcément une date de discrétisation
* @param N nombre de pas de constatation
* @param T date jusqu'à laquelle on simule la trajectoire (maturité)
* @param past (input) trajectoire réalisée jusqu'a la date t
*/
void asset (PnlMat *path, double t, int N, double T,
            PnlRng *rng, const PnlMat *past);
```

Cette méthode génère une trajectoire où la matrice `past` contient toute l'information du passé par rapport auquel on conditionne jusqu'à l'instant t, i.e les valeurs de $s_{t_0}, s_{t_1}, \ldots, s_{t_i}, s_t$. La trajectoire générée est stockée dans la variable `path`.

A ce stade, nous pouvons ajouter une nouvelle méthode `MonteCarlo::price` permettant de calculer le prix à l'instant t.

```
/**
* Calcule le prix de l'option à la date t
*
* @param  past (input) contient la trajectoire du sous-jacent
```

```
*                  jusqu'à l'instant t
* @param t (input) date à laquelle le calcul est fait
* @param prix (ouptut) contient le prix
* @param ic (ouptut) contient la largeur de l'intervalle de confiance
*                   sur le calcul du prix
*/
void price (const PnlMat *past, double t, double &prix, double &ic);
```

3.2 Calcul du delta

Dans cette partie, on cherche à écrire une méthode `MonteCarlo::delta` qui calcule le delta de l'option à un instant t à partir de la formule (5). Remarquons dans cette formule qu'il faut être capable de simuler 2 trajectoires utilisant les mêmes aléas Browniens mais shiftées l'une par rapport à l'autre. Considérons une trajectoire $(s_{t_0}, s_{t_1}, \dots, s_{t_i}, s_t\tilde{S}^{(j)}_{t_{i+1}-t}, \dots, s_t\tilde{S}^{(j)}_{t_N-t})$, nous souhaitons à partir de cette trajectoire construire pour $d = 1 \dots D$ les trajectoires

$$(s_{t_0}, s_{t_1}, \dots, s_{t_i}, g^d_+(s_t)\tilde{S}^{(j)}_{t_{i+1}-t}, \dots, g^d_+(s_t)\tilde{S}^{(j)}_{t_N-t})$$
$$(s_{t_0}, s_{t_1}, \dots, s_{t_i}, g^d_-(s_t)\tilde{S}^{(j)}_{t_{i+1}-t}, \dots, g^d_-(s_t)\tilde{S}^{(j)}_{t_N-t})$$

où

$$g^d_+(x) = \begin{pmatrix} x^1 \\ \vdots \\ x^{d-1} \\ x^d(1+h) \\ x^{d+1} \\ \vdots \\ x^D \end{pmatrix} \qquad g^d_-(x) = \begin{pmatrix} x^1 \\ \vdots \\ x^{d-1} \\ x^d(1-h) \\ x^{d+1} \\ \vdots \\ x^D \end{pmatrix}$$

Pour ce faire, nous pouvons créer une nouvelle méthode `BS::shift_asset`.

```
/**
*
* Shift d'une trajectoire du sous-jacent
*
* @param  path (input) contient en input la trajectoire
* du sous-jacent
* @param shift_path (output) contient la trajectoire path
* dont la composante d a été shiftée par (1+h) à partir de la date t.
* @param t (input) date à partir de laquelle on shift
* @param h (input) pas de différences finies
* @param d (input) indice du sous-jacent à shifter
* @param timestep (input) pas de constatation du sous-jacent
*/
void shift_asset (PnlMat *_shift_path, const PnlMat *path,
                  int d, double h, double t, double timestep);
```

Il ne reste vous plus ensuite qu'à écrire sur le principe des méthodes `MonteCarlo::price` une méthode `MonteCarlo::delta`

```
/**
*
* Calcul du delta de l'option à la date t
*
* @param  past (input) contient la trajectoire du sous-jacent
*                  jusqu'à l'instant t
* @param t (input) date à laquelle le calcul est fait
* @param delta (ouptut) contient le vecteur de delta
* @param ic (ouptut) contient la largeur de l'intervalle de confiance
*                   sur le calcul du delta
*/
void delta (const PnlMat *past, double t, PnlVect *delta, PnlVect *ic);
```

4 Simulation de la couverture

Pour les praticiens de la finance, parmi les critères qui interviennent dans le choix d'un modèle plutôt qu'un autre, la répartition du *P&L* (Profit and Loss, i.e. erreur de couverture) joue un rôle tout à fait central. Un modèle est jugé sur sa bonne capacité à couvrir les produits exotiques.

Maintenant que nous avons implémenté un pricer Monte Carlo capable de calculer les prix et les deltas de produits financiers à n'importe quelle date entre 0 et la maturité T du produit, nous allons créer un nouvel outil qui va reprendre la classe `BS` précédemment écrite à laquelle il faut rajouter le vecteur membre `trend` qui représente le paramètre μ apparaissant dans la dynamique du modèle de Black Scholes lorsque celle-ci est exprimée sous la probabilité historique et non sous la probabilité risque neutre.

$$S_t^d = S_0^d \, \mathrm{e}^{(\mu^d - (\sigma^d)^2/2)t + \sigma^d L^d W_t}, \quad d = 1 \dots D$$

Ajouter la méthode `simul_market` à la classe `BS` renvoyant une simulation du marché (c'est à dire une simulation du modèle sous la probabilité historique) avec un nombre de dates `H`, typiquement on prendra une date par jour ou par semaine. On considère la grille de discrétisation $0 = \tau_1 < \cdots < \tau_H = T$ de pas T/H. On supposera que $\{\tau_0, \dots, \tau_H\} \supset \{t_0, \dots, t_N\}$.

Le long de cette trajectoire, calculer le prix et le delta à chaque date τ_i et construire le portefeuille de couverture. Notons p_i et δ_i les prix et deltas calculés le long de cette trajectoire à la date τ_i alors l'évolution de la part investie au taux sans risque s'écrit.

$$\begin{cases} V_0 & = p_0 - \delta_0 \cdot S_0 \\ V_i & = V_{i-1} * \mathrm{e}^{\frac{r*T}{H}} - (\delta_i - \delta_{i-1}) \cdot S_{\tau_i}, \quad i = 1, \dots, H \end{cases}$$

L'erreur de couverture est donnée par $P\&L = V_H + \delta_H \cdot S_{\tau_H} - \texttt{payoff}$.

5 Le parser

Le logiciel de pricing que vous allez réaliser sera utilisé grâce à une interface console basique. Il devra être capable de lire les paramètres du problème à traiter dans un fichier texte dont la syntaxe est détaillée au paragraphe 5.1

5.1 Le format attendu

Votre projet devra être alimenté par un fichier texte du type

```
##ceci est un commentaire

option size 40
strike 100
spot 100
maturity 3
volatility 0.2
interest rate   0.05
correlation 0.2

option type basket
payoff coefficients 0.025

timestep number 20
sample number 10000
```

Les différentes clés peuvent apparaître dans un ordre quelconque dans le fichier sauf la clé `option size` que l'on supposera toujours être la première.

Il faut être capable de lire un tel fichier puis de remplir une pseudo table de hachage à partir des clés trouvées. Cette table est ensuite utilisée pour initialiser le bon type d'option en fonction de la ligne `option type`.

Voici la liste des clés devant être reconnues par le parseur.

Clé	Type
option size	entier
spot	vecteur
maturity	réel
volatility	vecteur
interest rate	réel
correlation	réel
trend	vecteur
strike	réel
option type	string
payoff coefficients	vecteur
lower barrier	vecteur
upper barrier	vecteur
timestep number	entier
time grid	vecteur
sample number	entier

Si une clé attend un paramètre vectoriel et que le fichier ne contient qu'une seule valeur pour cette clé, alors cette valeur sera affecté à chacune des composantes du vecteur.

5.2 Suggestion de solution de parseur

Voici une suggestion pour réaliser cette tâche. Considérons les déclarations suivantes.

```
// liste des types
typedef enum {
  T_NULL,
  T_INT,
  T_DOUBLE,
  T_VECTOR,
  T_STRING
} T_type;

// structure de comparaison pour la map
struct ltstr
{
  bool operator()(const char* s1, const char* s2) const
    {
      return strcmp(s1, s2) < 0;
    }
};

// structure pour les valeurs de la map
struct TypeVal
{
  bool is_set; // true if the value has been set
  T_type type; // true type of the value
  union {
    double V_double;
    int V_int;
    char *V_string;
    PnlVect *V_vector;
  }; // value
  TypeVal ();
  ~TypeVal ();
};

typedef std::map<const char *, TypeVal, ltstr> HashParam;

// Initialisation de la map< ... >
HashParam create_hash_map ()
{
  HashParam M;
  TypeVal T;

  T.type = T_STRING;
  M["option type"] = T;
```

```
  T.type = T_INT;
  M["option size"] = T;
  M["sample number"] = T;
  M["timestep number"] = T;

  T.type = T_DOUBLE;
  M["interest rate"] = T;
  M["correlation"] = T;
  M["maturity"] = T;
  M["strike"] = T;

  T.type = T_VECTOR;
  M["spot"] = T;
  M["payoff coefficients"] = T;
  M["upper barrier"] = T;
  M["lower barrier"] = T;
  M["volatility"] = T;

  return M;
}
```

1. Lire le fichier dans un tableau de chaînes de caractères `data` de façon à pouvoir le parcourir facilement.
2. Parcourir le tableau `data` et la map pour récupérer les valeurs des différentes clés et remplir la map.
3. Implémenter les fonctions suivantes permettant de récupérer la valeur d'une clé et qui seront utilisées dans les constructeurs des différentes classes

```
void set_from_map (const HashParam &, const char *, int &);
void set_from_map (const HashParam &, const char *, double &);
void set_from_map (const HashParam &, const char *, PnlVect **, int size);
void set_from_map (const HashParam &, const char *, char const **);
```

6 Produits à gérer

Voici la liste des options que votre pricer devra être capable d'évaluer (et de couvrir)

– Option panier (clé `basket`)

$$\left(\sum_{i=1}^{D} \lambda_i S_T^{(i)} - K\right)_+$$

où $\lambda_i \in \mathbb{R}$ et $K \in \mathbb{R}$. Le cas $K \leq 0$ permet de considérer une option de vente

– Option asiatique discrète en dimension $D = 1$ (clé `asian`)

$$\left(\frac{1}{N}\sum_{i=0}^{N} S_{t_i} - K\right)_+$$

– Option barrière basse (clé `barrier_l`)

$$\left(\sum_{d=1}^{D} \lambda_d S_T^{(d)} - K\right)_+ \mathbf{1}_{\forall 0\leq i\leq N, \forall 1\leq d\leq D,\ \bar{L}^d \leq S_{t_i}^{(d)}}$$

où $\bar{L} \in \mathbb{R}^D$

– Option barrière haute (clé `barrier_u`)

$$\left(\sum_{d=1}^{D} \lambda_d S_T^{(d)} - K\right)_+ \mathbf{1}_{\forall 0\leq i\leq N, \forall 1\leq d\leq D,\ S_{t_i}^{(d)} \leq \bar{U}^d}$$

où $\bar{U} \in \mathbb{R}^D$

– Option barrière (clé `barrier`)

$$\left(\sum_{d=1}^{D} \lambda_d S_T^{(d)} - K\right)_+ \mathbf{1}_{\forall 0\leq i\leq N, \forall 1\leq d\leq D,\ \bar{L}^d \leq S_{t_i}^{(d)} \leq \bar{U}^d}$$

où $\bar{L}, \bar{U} \in \mathbb{R}^D$ et $\bar{L} \leq \bar{U}$ terme à terme.

– Option performance sur panier (clé `performance`)

$$1 + \min\left(\left(\frac{1}{N}\sum_{i=1}^{N} \frac{\sum_{d=1}^{D} \lambda_d S_{t_i}^{(d)}}{\sum_{d=1}^{D} \lambda_d S_{t_{i-1}}^{(d)}} - 1\right)_+, 0.1\right)$$