HEC Montréal

Évaluation d'options bi-dimensionnelles par programmation dynamique et analyse en composantes principales

par
Martin Blanchard

Sciences de la décision
Ingénierie financière

Mémoire présenté en vue de l'obtention
du grade de maîtrise ès sciences
(M.Sc.)

Août 2019

Table des matières

1 Introduction **6**

2 Revue de littérature **7**

3 Modèle **10**
3.1 Dynamique des deux sous-jacents 10
3.2 Programmation dynamique 10
3.3 Analyse en composantes principales 12
3.4 Algorithme ACP et programmation dynamique 14

4 Programmation parallèle **18**
4.1 OpenMP 18
4.2 MPI 20

5 Investigation numérique **22**
5.1 Résultats 22
5.2 Temps de calculs 30

6 Conclusion **33**

7 Annexe **35**

Liste des tableaux

1 Option européenne put-on-min algorithme BBR 22
2 Option européenne put-on-min algorithme ACP 23
3 Option américaine call-on-min algorithme BBR 24
4 Option américaine call-on-min algorithme ACP 25
5 Option américaine put-on-min algorithme BBR 26
6 Option américaine put-on-min algorithme ACP 26
7 Option américaine put-on-min algorithme BBR 27
8 Option américaine put-on-min algorithme ACP 27
9 Option américaine call-on-max algorithme BBR 28
10 Option américaine call-on-max algorithme ACP 28
11 Option américaine call-on-max algorithme BBR 29
12 Option américaine call-on-max algorithme ACP 29
13 Temps de calcul pour l'algorithme ACP $p = q = 72$ 30
14 Temps de calcul pour l'algorithme ACP $p = q = 144$ 30
15 Temps de calcul pour l'algorithme ACP $p = q = 300$ 31

Résumé

Nous abordons le problème d'évaluation d'options à deux sous-jacents. Notre méthode se base sur la programmation dynamique couplée aux éléments finis et à l'analyse en composantes principales. Cette dernière permet ainsi de limiter l'espace mémoire requis pour effectuer l'évaluation. Par ailleurs, nous proposons une interpolation bi-quadratique par morceaux pour approximer la fonction valeur à chaque étape du programme dynamique, ce qui permet d'augmenter l'efficacité.

La programmation dynamique nécessite une discrétisation de l'espace état pour chaque pas de temps. Puisque les points des disrétisations sont indépendants, il existe une opportunité de parallélisation sur les prix des sous-jacents que nous exploitons afin de réduire le temps de calcul.

Nous comparons nos approximations avec les formes fermées existantes ainsi que d'autres méthodes d'approximation utilisées dans la littérature autant en termes de précision que de temps de calcul. Nos résultats montrent que notre méthode est plus efficace que des méthodes alternatives.

Mots-clés : Options bi-dimensionnelles, programmation dynamique, méthode des éléments finis, analyse en composantes principales, programmation parallèle

Remerciements

Je tiens à remercier mes co-directeurs, Hatem Ben-Ameur et Michèle Breton, pour leur encadrement, pour m'avoir poussé et pour leur disponibilité. Leurs idées et leur support ont été essentiels à la réalisation de ce mémoire.

Je tiens également à remercier mes amis et ma famille qui m'ont encouragé et qui ont toujours eu confiance en mes capacités tout au long de ce processus.

1 Introduction

L'ingénierie financière est une discipline qui se situe à la croisée de plusieurs autres. Son objectif est financier, ses fondements sont mathématiques et statistiques et elle effectue son travail grâce à la programmation informatique. Un des nombreux problèmes sur lequel elle se penche est de quantifier la valeur aujourd'hui d'un événement futur incertain. Bien que la réponse puisse être très simple dans certains cas, les cas les plus complexes exigent un savant mélange de toutes les disciplines énumérées.

Dans ce mémoire, nous traitons d'un cas particulier de ce problème, à savoir l'évaluation d'options comportant deux sous-jacents. L'ajout de dimensions au problème d'évaluation d'options le rend exponentiellement plus complexe, ce qui est communément référé comme étant la malédiction de la dimension. Les solutions développées à une dimension se heurtent ainsi rapidement aux limites computationnelles des ordinateurs disponibles, soit en terme de mémoire, du nombre d'opérations à effectuer (temps de calcul), ou des deux.

Nous utilisons la programmation dynamique couplée aux éléments finis afin d'évaluer le prix d'options à deux sous-jacents. Nous utilisons aussi l'analyse en composantes principales (ACP) afin de créer de nouvelles dynamiques orthogonales, et donc indépendantes si Gaussiennes.

Nous explorons les opportunités de parallélisation du problème sur un ordinateur portable avec la librairie OpenMP, où la mémoire est partagée, mais aussi sur plusieurs ordinateurs de Calcul Canada à mémoire distribuée avec la librairie Message Passing Interface (MPI).

Le mémoire est organisé comme suit. La section 2 contient la revue de littérature. La section 3 décrit et explicite le modèle. S'en suit une discussion de la parallélisation de notre algorithme à la section 4 et des résultats numériques à la section 5. Nous revenons sur notre travail en conclusion et fournissons les détails de nos calculs en annexe.

2 Revue de littérature

Une option est un contrat entre deux contreparties dans lequel une des deux s'engage à verser à l'autre un montant conditionnel au prix d'un ou de plusieurs sous-jacents. Les options les plus communes sont les options d'achat et de vente, qui donnent à leur détenteur l'option mais pas l'obligation d'acheter (respectivement, de vendre) une action à un prix fixé d'avance. Une option *européenne* permet d'exercer l'option à l'échéance du contrat, alors qu'une option *américaine* permet d'exercer à tout moment précédant la date d'échéance. Entre les deux se trouvent les options *bermudéennes*, qui permettent l'exercice à certains moments précis pendant la durée du contrat.

La première grande avancée dans le domaine de l'évaluation des options est apportée par Black et Scholes (1973) qui proposent une formule fermée en formulant un rendemen log-normal du sous-jacent. Ceux-ci assument des conditions "idéales" parmi lesquelles une volatilité et un taux d'intérêt constants ainsi qu'un marché pur et parfait.

Margrabe (1978) propose une forme fermée pour les options d'échanges à deux dimensions dans le modèle de Black Scholes. Stultz (1982) fournit une formule fermée pour les options sur le maximum ou le minimum de deux sous-jacents. Cette formule sera très utile pour valider nos résultats.

Hartley (2000) utilise la méthode des éléments finis pour les options américaines, ainsi que Kovalov, Linetsky et Marcozzi (2007), qui s'en servent pour approximer la solution d'une équation aux dérivées partielles, qui est elle-même une approximation d'une inéquation aux dérivées partielles variationnelle. L'ajout d'un terme de pénalité leur permet ainsi de considérer l'équation aux dérivées partielles comme tenant en compte la possibilité d'exercice. Dans la même veine, Berridge et Shumacher (2008) utilisent la méthode des éléments finis sur une grille construite par simulation quasi-Monte Carlo obtenue par échantillonage par importance. Ils calculent ensuite des matrices de transitions qu'ils utilisent pour reculer dans le temps à partir de l'expiration.

Cox, Ross et Rubenstein (1979) proposent leur modèle d'arbre binomial, qui recoupe le modèle de Black Scholes pour les options européennes, mais qui se prête également à l'évaluation d'options américaine et au paiement de dividendes. Boyle (1988) utilise la même intuition pour évaluer des options bi-dimensionnelles en formant une pyramide inversée sur le prix des sous-jacents. Ce type d'arbre à 5 cinq branches est utilisé pour évaluer entre autres une option américaine à 2 sous-jacents. Cette méthode est aussi utilisée par

Kamrad et Ritchken (1991) qui l'étendent à k sous-jacents.

Les méthodes de simulation ont longtemps été considérées comme inappropriées à l'évaluation d'option américaines en raison de la difficulté d'avoir une fonction de décision précise à tout point de la simulation. Tiley (1993) ouvre la voie à une nouvelle famille d'approche, soit les méthodes de groupement ("bundling"). Il approxime la valeur de détention en regroupant des points de simulation proches les uns des autres. Cette valeur sera la valeur présente de toutes les trajectoires regroupées dans le groupe au temps suivant. Il décide ainsi approximativement si l'option doit être exercée ou non.

Barraquand et Martineau (1995) présentent une technique de simulation permettant d'évaluer des options américaines à plusieurs sous-jacents, jusqu'à 400 selon eux. Ils procèdent en divisant l'espace d'état selon l'axe d'exécution. Broadie et Glasserman (1997) proposent deux bornes pour le prix de l'option, une supérieure et une inférieure en projetant deux prix, un biaisé par le haut et un autre biaisé par le bas. Raymar et Zwecher (1997) utilisent la simulation Monte Carlo pour calculer le prix d'une option d'achat sur le maximum de plusieurs sous-jacents qui paient des dividendes continus.

La technique de régression non-paramétrique est utilisée par Carrière (1996) avant que Longstaff et Schwartz (2001) utilisent la méthode des moindres carrés pour mener leurs simuluations. Detemple, Feng et Tian (2003) construisent des bornes supérieures et inférieures sur le prix d'une option sur le minimum de deux sous-jacent qu'ils obtiennent par simulation Monte Carlo.

Bally, Pagès et Printemps (2003) et (2005) utilisent une méthode prise du domaine du traitement de signal appelée la quantification vectorielle ("vector quantization"). Les idées d'arbres et de grilles n'y sont pas complètement absentes, mais la simulation est tout de même utilisée pour déterminer la grille à chaque pas de temps. Jin et Al. (2007) et (2013) utilisent cette méthode avec celle de Barraquand et Martineau pour évaluer des options sur plusieurs sous-jacents en réduisant la dimensionnalité du problème.

Très près de cette méthode se trouvent aussi Broadie, Glasserman et Ha (2000) et Broadie et Glasserman (2004). Ils simulent le processus des sous-jacents par pas de temps, et utilisent la position à chaque pas de temps comme étant la grille à ce point. Ils calculent ensuite des poids qui agissent comme la probabilité de transition d'un point de la grille à chaque point de la grille suivante. Ces poids ignorent complètement le fait que les points sont liées par la même simulation. Différents critères de sélection des poids sont explorés, soit l'entropie maximale et la méthodes des moindres carrés.

Anderson et Broadie (2004) incorporent le primal et le dual dans le calcul

d'intervalles de confiance pour leur simulation, respectivement pour la borne inférieure et supérieure. Ils attribuent l'utilisation du dual dans l'expression de l'option bermudéenne simultanément à Haugh et Kogan (2004) et à Rogers (2002).

La première utilisation de l'anlyse des composantes principales à l'évaluation d'options multidimensionnelles peut être attribuée à Reisinger et Wittum (2007). Ceux-ci développent à la fois une discrétisation de l'espace, couplée à une technique qu'ils nomment l'expansion asymptotique, par laquelle ils explorent l'espace deux dimensions à la fois grâce à l'ACP, en formant des plans avec la première et la $n$ième composante principale, sur lesquelles ils calculent la solution de l'équation de la chaleur en ignorant les autres dimensions. Ils transforment ainsi un problème n-dimensionnel en $n-1$ problèmes 2-dimensionnels.

Se basant sur la thèse de Reisinger, Ekedahl, Hansander et Lehto (2007) publient un cahier de recherche dans lequel ils reprennent et testent ses affirmations, et découvrent que sa méthode perd en efficacité lorsque la corrélation entre les sous-jacents est faible.

Reisinger développe son idée dans Reisinger et Wissman (2015), où ils utilisent l'expansion asymptotique sur des options plus complexes tels que des swaptions sur le LIBOR. De son côté, Wittum développe une méthode de parallélisation afin de travailler sur les temps de calcul de la méthode précédente dans Schober, Schröder et Wittum (2015).

Du côté des options bi-dimensionnelles, Ben-Abdellatif, Ben-Ameur et Rémillard (2019) présentent un modèle de résolution numérique applicable à toute forme d'options exerçable avant l'expiration en utilisant la programmation dynamique et la méthode des différences finies. Nous prendrons leur papier comme base en tentant d'utiliser l'ACP pour améliorer l'efficacité et la portabilité de l'algorithme.

3 Modèle

Nous présentons ici notre modèle et introduisons l'ACP pour une meilleure utilisation de la mémoire de l'ordinateur.

3.1 Dynamique des deux sous-jacents

Nous nous plaçons dans le modèle Black Scholes bivarié. Nous utilisons les notations suivantes pour les variables du modèle :

S_t^1 : Prix du premier sous-jacent au temps t ;

S_t^2 : Prix du deuxième sous-jacent au temps t ;

d_1 : Taux de dividende du premier sous-jacent ;

d_2 : Taux de dividende du deuxième sous-jacent ;

σ_1 : Volatilité du premier sous-jacent ;

σ_2 : Volatilité du deuxième sous-jacent ;

r : Taux d'intérêt sans risque ;

T : Temps jusqu'à l'expiration du contrat.

Le mouvement des sous-jacents suit un processus qui respecte l'équation suivante :

$$dS_t^i = S_t^i(r - d_i)dt + S_t^i\sigma_i dW_t^i, \qquad \text{i} = 1,2,$$

où (W_t^1, W_t^2) représentent des mouvements Brownien corrélés avec

$$Corr(W_t^1, W_t^2) = \rho.$$

Cela implique que le prix au temps $u > t$ suit une loi log-normale. La relation suivante décrit le mouvement du prix à travers le temps :

$$S_u^i = S_t^i e^{(r-d_i-\frac{\sigma_i^2}{2})\Delta t+\sigma_i(W_u^i-W_t^i)}, \ pour \ u \geq t \geq 0. \tag{1}$$

3.2 Programmation dynamique

Pour une option américaine, la valeur d'exercice est toujours connue et est spécifique au contrat. Dans le cas d'options bi-dimensionnelle, celle-ci peut être calculée sur le maximum des deux actions par exemple, ou une moyenne pondérée des deux prix à l'exercice. Par contre, la valeur de détention, qui

provient du non-exercice et de la probabilité d'un exercice futur doit être estimée à chaque pas de temps pour être comparée à la valeur d'exercice. La valeur de l'option est alors le maximum des deux, car le propriétaire exercera l'option si c'est plus avantageux que de la garder, et vice versa. Selon le modèle, la valeur de détention au temps t dépend des valeurs possibles au temps $t + \Delta t$ de la façon suivante :

$$\mathbb{E}_t^{\mathbb{Q}}[e^{-r\Delta t} v_{t+\Delta t}(S^1_{t+\Delta t}, S^2_{t+\Delta t}) | S^1_t = s^1_t, S^2_t = s^2_t]$$

où $v_t(S^1_t, S^2_t)$ est la valeur de l'option au temps t.

Il existe donc une relation de récurrence entre la valeur à un temps donné et les valeurs futures possibles. Puisqu'à l'expiration du contrat la valeur est égale à la valeur d'exercice, nous pouvons remonter les incréments de temps à partir de ce moment pour arriver à la valeur au temps 0.

Ben-Abdellatif, Ben-Ameur et Rémillard (BBR) utilisent la programmation dynamique couplée aux éléments finis pour calculer l'espérance. Ils forment une partition $a_1 < a_2 < ... < a_{p-1} < a_p$ pour le premier sous-jacent et $b_1 < b_2 < ... < b_{q-1} < b_q$ pour le deuxième, avec $a_0 = b_0 = 0$ et $a_{p+1} = b_{q+1} = \infty$. La partition qu'ils forment sur les deux axes les poussent à calculer des matrices de transitions sur les probabilités que la paire d'action finisse dans un rectangle $[a_i, a_{i+1}) \times [b_j, b_{j+1})$, noté R_{ij}. Les probabilités de transitions qu'ils calculent sont alors :

$$T^{\nu\mu}_{klij} = \mathbb{E}^{\mathbb{Q}}[(S^1_{t+\Delta t})^\nu (S^2_{t+\Delta t})^\mu \mathbb{I}((S^1_{t+\Delta t}, S^2_{t+\Delta t}) \in R_{ij}) | S^1_t = a_k, S^2_t = b_l]$$

pour $0 \leq i, j, k, l \leq p$. Puisque les variables S^1 et S^2 sont corrélées, la distribution normale bivariée est nécessaire pour calculer ces matrices. Les paramètres ν et μ prennent les valeurs 0 et 1 pour une interpolation linéaire, menant au calcul de 4 matrices, T^{00}_{klij}, T^{10}_{klij}, T^{01}_{klij} et T^{11}_{klij}.

Avec les 4 facteurs (i, j, k, l), cette matrice contient 4 dimensions différentes. Elle prend en compte la position de départ de chaque sous-jacent, les dimensions k et l, ainsi qu'une fenêtre d'arrivée pour chacun d'eux, les dimensions i et j. Sa taille augmente exponentiellement avec la taille de la grille p, ce qui cause rapidement des problèmes de mémoire. Par exemple, pour $p = 100$, chaque matrice aura 10^8 éléments, excédant déjà les capacités des ordinateurs portables.

La première solution à ce problème est de ne pas stocker les matrices de transitions. Plutôt que de les calculer au début du programme et aller chercher les valeurs lorsqu'elles sont nécessaires, nous pouvons les incorporer

dans les calculs. Ainsi, à chaque fois que le programme aura besoin d'une probabilité, il la calculera sur le champ. Cette solution ne nécessitera que le stockage de la matrice de prix, qui est de dimension p^2 plutôt que p^4.

Cette solution présente comme désavantage le re-calcul des probabilités à chaque fois qu'elle sont nécessaires. Il faudra les calculer une fois par pas de temps, ce qui finit par être un transfert du problème de la mémoire disponible vers un temps de calcul significatif.

La deuxième solution est l'utilisation de la programmation parallèle. Elle consiste à utiliser plusieurs ordinateurs différents qui mémorisent chacun une partie de la matrice. Chaque ordinateur est en charge des calculs qui touchent la partie de la matrice qu'il a en mémoire. Un échange d'information est effectué entre un ordinateur qui coordonne les autres et tient en compte des prix de l'option à chaque pas de temps.

Cette solution ne permet pas au programme de fonctionner sur les ordinateurs personnels et nécessite un travail additionnel de répartition de la mémoire entre les ordinateurs disponibles et une allocation du travail précise en fonction de l'emplacement des différentes parties de la mémoire disponible. C'est la solution explorée par BBR.

Une troisième solution consiste à former des combinaisons linéaires des mouvements des sous-jacents pour former de nouveaux mouvements non-corrélés. Puisque notre modèle est Gaussien, les mouvements non-corrélés seront aussi indépendants. Ceci permet de réduire l'espace mémoire requis pour stocker l'information nécessaire aux calculs de l'algorithme.

Nous mettons en place cette dernière solution en utilisant l'ACP pour obtenir les facteurs nécessaires à une combinaison linéaire indépendante des mouvements des sous-jacents. Nous donnons d'abord une définition de l'ACP.

3.3 Analyse en composantes principales

L'ACP est une méthode statistique qui a été introduite par Karl Pearson (1901). Elle a pour but de tracer de nouveaux axes à partir d'un jeu de données à plusieurs dimensions. Ces axes, nommés composantes principales, constituent de nouvelles coordonnées par lesquelles un problème peut être examiné. L'ACP crée ces coordonnées par combinaison linéaire des variables originales afin qu'elles soient non-corrélés.

En plus de créer des nouvelles variables non-corrélées, une des propriétés de l'ACP est de numéroter ces axes selon leur variance, afin d'identifier les variables qui contribuent le plus à la variance totale du jeu de données. Ainsi,

la première composante principale sera la combinaison linéaire des variables originales telle que la variance des données selon cet axe est maximale. Les composantes suivantes suiveront la même logique, mais avec la contrainte d'être orthogonales aux précédentes. Elles capturent donc successivement la variabilité restante.

Nous obtenons cette décomposition en calculant la matrice variance-covariance d'un ensemble de données, souvent obtenu à partir de l'observation d'un phénomène physique. La diagonalisation de cette matrice (par définition carrée) mène à une série de vecteurs et de valeurs propres. Ces valeurs propres indiquent la variance que représente le vecteur correspondant, et ainsi la valeur propre la plus grande représentera la première composante principale, et ainsi de suite.

Or, puisque notre modèle est purement théorique, nous disposons déjà d'une matrice de variance-covariance sur les mouvements browniens des prix. Il n'est donc pas nécessaire de passer par l'étape d'estimer la matrice à partir d'un jeu d'observations, et il suffira d'effectuer une diagonalisation pour obtenir les composantes principales.

Puisque

$$Var(ln(S_u^i)|ln(S_t^i)) = \sigma_i \Delta t$$

et que

$$Cov(ln(S_u^1), ln(S_u^2)|ln(S_t^1), ln(S_t^2)) = \sigma_1 \sigma_2 \Delta t$$

nous avons la matrice de variance-covariance suivante :

$$\begin{pmatrix} \sigma_1 & \sigma_1\sigma_2 \\ \sigma_1\sigma_2 & \sigma_2 \end{pmatrix}$$

où nous avons divisé la matrice par Δt.

En diagonalisant cette matrice, nous obtenons deux vecteurs, μ^1 et μ^2 tels que :

$$\begin{cases} ln(C_u^1) = \mu_1^1 ln(S_u^1) + \mu_2^1 ln(S_u^2) \\ ln(C_u^2) = \mu_1^2 ln(S_u^1) + \mu_2^2 ln(S_u^2) \end{cases}$$

sont des variables orthogonales. Les nouveaux mouvements C_u^1 et C_u^2 suivent maintenant les équations suivantes :

$$C_u^i = e^{\mu_i^1(ln(S_t^1)+(r-d_1-\frac{\sigma_1^2}{2})\Delta t)+\mu_2^i(ln(S_t^2)+(r-d_2-\frac{\sigma_2^2}{2})\Delta t)+\mu_1^i\sigma_1(W_u^1-W_t^1)+\mu_2^i\sigma_2(W_u^2-W_t^2)}$$

Ce qui implique que, sachant la position des processus au temps t, la loi de $ln(C_u^i)$ est normale de moyenne $\mu_1^i(ln(S_t^1)+(r-d_1-\frac{\sigma_1^2}{2})\Delta t)+\mu_2^i(ln(S_t^2)+(r-d_2-\frac{\sigma_2^2}{2})\Delta t)$ et de variance $(\mu_1^i)^2\sigma_1^2\Delta t+(\mu_2^i)^2\sigma_2^2\Delta t+2\rho\mu_1^i\mu_2^i\sigma_1\sigma_2\Delta t$. Il est à noter que non seulement les nouvelles variables sont-elles non-corrélées, mais puisqu'elle sont de loi normale, elle sont alors aussi indépendantes. Cette propriété sera cruciale dans notre approche.

3.4 Algorithme ACP et programmation dynamique

En utilisant les composantes principales, nous avons maintenant de nouveaux axes avec lesquels parcourir l'espace d'état. La valeur de détention à un point de la grille devient alors :

$$\mathbb{E}_t^{\mathbb{Q}}[e^{-r\Delta t}w_{t+\Delta t}(C_{t+\Delta t}^1, C_{t+\Delta t}^2)|ln(C_t^1)=a_i', ln(C_t^2)=b_j']$$

où w est la même fonction de valeur de l'option, mais adaptée aux variables C_i.

Nous divisons ces deux axes de la grille en p et en q points respectivement selon la position future des mouvements, en couvrant ± 5 fois la variance totale du processus sur la durée du contrat. C'est dire que

$$a_1' = \mu_1^1(ln(S_t^1)+(r-d_1-\tfrac{\sigma_1^2}{2})T)+\mu_2^1(ln(S_t^2)+(r-d_2-\tfrac{\sigma_2^2}{2})T)-\tfrac{5}{p}* \\ ((\mu_1^1)^2\sigma_1^2+(\mu_2^1)^2\sigma_2^2+2\rho\mu_1^1\mu_2^1\sigma_1\sigma_2)T$$

avec des incrémentations de

$$\tfrac{10}{p}*((\mu_1^1)^2\sigma_1^2+(\mu_2^1)^2\sigma_2^2+2\rho\mu_1^1\mu_2^1\sigma_1\sigma_2)T$$

entre chaque point de la grille, et de façon analogue pour la deuxième composante. Nous gardons comme BBR $a_0'=b_0'=0$ et $a_{p+1}'=b_{q+1}'=\infty$

Nous disposons maintenant d'une grille formée de deux vecteurs. Notre algorithme partant du temps 0, il nous suffit de calculer la valeur d'exercice à ce temps. Or, puisque nous travaillons maintenant avec les composantes principales plutôt qu'avec les logs-prix des sous-jacents, il est nécessaire de partir des logs-composantes principales et de retrouver le prix des sous-jacents correspondant à ces valeurs, car chaque point de la grille des composantes est liée à un point de la grille originale.

Nous résolvons donc le système suivant :

$$\begin{cases} a'_i = \mu_1^1 ln(S_{ij}^1) + \mu_2^1 ln(S_{ij}^2) \\ b'_j = \mu_1^2 ln(S_{ij}^1) + \mu_2^2 ln(S_{ij}^2) \end{cases}$$

pour S_{ij}^1, S_{ij}^2 et pour tout $1 \leq i \leq p$, $1 \leq j \leq q$.

L'obtention des valeurs des sous-jacents aux points étudiés par l'algorithme ACP permet ensuite de calculer la valeur d'exercice à ces points. Nous stockons cette matrice d'exercice car elle sera utilisée à chaque étape afin de la comparer avec la matrice de détention pour les points correspondant. Par ailleurs, nous initialisons la matrice des valeurs à la matrice d'exercice pour $t = T$.

Nous utilisons l'intégration numérique pour évaluer la valeur de détention. Les matrices de transition deviennent :

$$\begin{aligned} T_{klij}^{\nu\mu} &= \mathbb{E}^{\mathbb{Q}}[(C_{t+\Delta t}^1)^\nu (C_{t+\Delta t}^2)^\mu \mathbb{I}(ln((C_{t+\Delta t}^1), ln(C_{t+\Delta t}^2)) \in [a'_i, a'_{i+1}) \times [b'_j, b'_{j+1}))| \\ &\qquad ln(C_t^1) = a'_k, ln(C_t^2) = b'_l] \\ T_{klij}^{\nu\mu} &= \mathbb{E}^{\mathbb{Q}}[(C_{t+\Delta t}^1)^\nu \mathbb{I}(ln(C_{t+\Delta t}^1) \in [a'_i, a'_{i+1}))|ln(C_t^1) = a'_k] * \\ &\qquad \mathbb{E}^{\mathbb{Q}}[(C_{t+\Delta t}^2)^\mu \mathbb{I}(ln(C_{t+\Delta t}^2) \in [b'_j, b'_{j+1}))|ln(C_t^2) = b'_l] \\ T_{klij}^{\nu\mu} &= T_{ki}^{1\nu} * T_{lj}^{2\mu} \end{aligned}$$

où le premier exposant désigne la composante principale et le deuxième désigne l'exposant du mouvement dans l'espérance.

Ainsi, par indépendance nous avons pu diviser les matrices et les faire passer de quatre dimensions (rectangle de départ à rectangle d'arrivée) à deux dimension (segment de départ à segment d'arrivée). Nous calculons ainsi 2 matrices à deux dimensions plutôt qu'une seule à quatre.

Pour évaluer l'intégrale, nous devons évaluer ces matrices de probabilités de transitions. Cela nous permet d'évaluer le prix de l'option en chaque point au temps t. Nous donnons les détails en annexe. Par contre, pour ce faire nous devons aussi avoir une fonction W complète au temps $t + \Delta t$, c'est-à-dire définie sur tout l'espace d'état. Puisque nous avons la valeur de la fonction en des points discrets, nous allons rendre la fonction valeur continue en procédant à deux type d'interpolation, soit bi-linéaire et bi-quadratique.

Nous utilisons l'interpolation bi-linéaire pour comparer nos résultats fidèlement avec BBR. Nous programmerons aussi l'interpolation bi-quadratique pour observer le gain d'efficacité qu'elle peut nous fournir. Ce type d'interpolation nous permet de capturer les courbes de la fonction, mais nécessite un effort computationnel plus lourd. Nous donnons les équations en annexe. Nous cherchons donc (pour la bi-linéaire) des paramètres $\alpha_{ij}^{\nu\mu}$ tels que :

$$\hat{w}_{t+\Delta t}(C^1_{t+\Delta t}, C^2_{t+\Delta t}) \approx e^{-r\Delta t}(\textstyle\sum_{ij} \alpha_{ij}^{00} + \alpha_{ij}^{10} C^1_{t+\Delta t} + \alpha_{ij}^{01} C^2_{t+\Delta t} + \alpha_{ij}^{11} C^1_{t+\Delta t} C^2_{t+\Delta t})$$

ce qui implique la résolution du système suivant :

$$\begin{cases} \hat{w}_{t+\Delta t}(a'_{i-1}, b'_{j-1}) = \tilde{w}_{t+\Delta t}(a'_{i-1}, b'_{j-1}) \\ \hat{w}_{t+\Delta t}(a'_{i-1}, b'_{j}) = \tilde{w}_{t+\Delta t}(a'_{i-1}, b'_{j}) \\ \hat{w}_{t+\Delta t}(a'_{i}, b'_{j-1}) = \tilde{w}_{t+\Delta t}(a'_{i}, b'_{j-1}) \\ \hat{w}_{t+\Delta t}(a'_{i}, b'_{j}) = \tilde{w}_{t+\Delta t}(a'_{i}, b'_{j}) \end{cases}$$

Pour les points sur les bornes, nous les plaçons égaux aux facteurs du point le plus près. La valeur de l'option en un point de la grille sera alors :

$$\hat{w}(a'_i, b'_j) =$$
$$max(v^e(a'_i, b'_j), e^{-r\Delta t}(\textstyle\sum_{ij} \alpha_{ij}^{00} T_{ij}^{10} T_{ij}^{20} + \alpha_{ij}^{10} T_{ij}^{11} T_{ij}^{20} + \alpha_{ij}^{01} T_{ij}^{10} T_{ij}^{21} + \alpha_{ij}^{11} T_{ij}^{11} T_{ij}^{21}))$$

L'algorithme alterne ensuite entre le calcul des valeurs de la grille et l'interpolation de cette grille discrète sur l'ensemble de l'espace d'état, avec chaque aller-retour représentant un incrémentation de temps.

Le système d'équation présenté ci-haut correspond à la situation de la Figure 1. Nous pouvons y observer, en orange, une fonction convexe similaire à la fonction que nous tentons d'estimer et en bleu une interpolation bi-linéaire sur cette fonction. Les quatres coins représentent chacun une équation du sytème, et la résolution du système nous offre les paramètres alphas de l'interpolation.

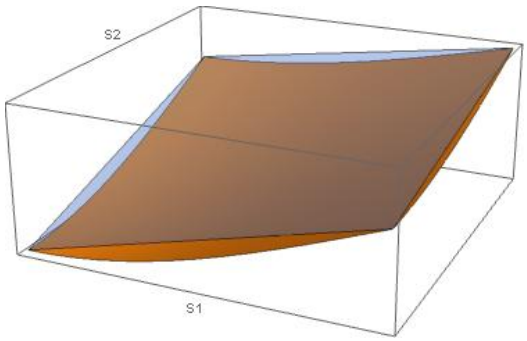

FIGURE 1 – Interpolation bi-linéaire sur une fonction convexe

En sommaire, les étapes de l'algorithme sont les suivantes :

(1) Calcul des valeurs et des vecteurs propres à partir de la matrice de variance-covariance

(2) Calcul des vecteurs qui forment la grille d'état

(3) Calcul des matrices de probabilités

(4) Calcul de la grille d'exercice

(5) Calcul parallèle

Le calcul parallèle consiste à l'incrémentation en arrière dans le temps. Au vu de l'indépendance entre les différents points dans ces calculs, le problème est un excellent candidat à la parallélisation. Par contre, puisque les informations calculées par la portion en série sont nécessaires aux calculs de la portion parallèle, celle-ci doit absolument être effectuée avant. Nous avons donc un algorithme en deux partie, soit une en série et une deuxième en parallèle. Les détails des calculs en parallèle seront données dans la prochaine section.

4 Programmation parallèle

De par la nature de l'algorithme que nous avons mis de l'avant, une opportunité de parallélisation s'ouvre à nous. En effet, bien que l'ensemble de la grille au temps $t + \Delta t$ soit nécessaire au calcul de la grille du temp t, les différents points de la grille au temps t sont complètement indépendants les uns des autres. Ainsi, puisque la grande majorité du temps de calcul se trouve dans le recul d'un pas de temps, c'est-à-dire dans le calcul de la grille au temps t, nous avons une grande opportunité de gain en temps de calcul avec la programmation parallèle. Nous explorons les librairies OpenMP et MPI afin d'utiliser les processeurs à mémoire partagée et à mémoire distribuée respectivement.

La programmation en parallèle provient d'un besoin toujours plus grand d'efficacité computationnelle. Or, le dévelopement de processeurs plus puissants est confronté à des limites physiques. Pour passer outre ces limites, la solution explorée est de diviser la tâche et de faire réaliser les parties par différentes ressources.

Pour bien comprendre la parallélisation des calculs nous devons commencer dans le processeur, l'unité qui les effectue. Un processeur contient lui-même un ou plusieurs coeurs, qui sont capables d'exécuter des programmes. Mettre plusieurs coeurs dans un processeur constitue une façon d'allouer plus qu'une ressource aux calculs. Pour procéder à la parallélisation du calcul, des threads sont créés. Nous pouvons imaginer le programme global rouler à plusieurs endroits différents simultanément, comme quatre travailleurs construisant la même maison ensemble, mais chacun en charge d'ériger son propre mur. La particularité du thread est que, puisque les coeurs sont dans le même processeurs, la mémoire sera la même pour chacun, donc partagée.

Par opposition, une autre méthode d'effectuer des calculs parallèles est d'utiliser plusieurs processeurs, comme construire plusieurs maisons en même temps. Les différents calculs ne partagent alors pas leur mémoire et ne sont ainsi pas appelés threads, mais processus. Il s'agit de la mémoire ditribuée.

4.1 OpenMP

Pour des calculs à plusieurs coeurs qui partagent la même mémoire, la parallélisation du problème peut se faire à l'aide de la librairie OpenMP. Le partage de la mémoire implique que les différentes portions du problème n'auront pas à se communiquer l'information car tous se la partagent. Tout

les coeurs peuvent lire et changer la valeur des variables du problème. Cela ne vient pas sans complication, et le programme doit être construit de sorte à ce qu'un coeur ne modifie pas la valeur d'une variable qu'un autre coeur a besoin par inadvertance.

Bien que le partage de la mémoire soit très pratique du point de vue computationnel, la construction de processeurs à mémoire partagée devient rapidement extrêmement coûteuse à mesure que le nombre de coeurs augmente. Il existe ainsi très peu d'ordinateurs à mémoire partagée. Par contre, il existe un point d'équilibre coût-bénéfice et même les ordinateurs à plusieurs processeurs auront une portion à mémoire partagée sur chacun de leur processeurs. Cela est aussi le cas des ordinateurs portables, qui auront souvent 4 coeurs.

Bien que toutes les tâches ne puissent pas être effectuées en parallèle, les boucles sont souvent d'excellents candidats à la parallélisation, pourvu que les itérations ne soient pas dépendantes entre elles. Nous utiliserons donc OpenMP sur l'ordinateur portable à notre disposition, car l'algorithme ACP s'y prête bien.

Le partage de la mémoire est un avantage en terme des besoins de communications entre les différents threads. Ces besoins deviennent pratiquement inexistants car chaque thread pige dans la même mémoire commune. Cela pose par contre le danger que les threads se nuisent dans leurs calculs. En effet, si un thread modifie une variable, le prochain qui en aura besoin n'aura plus accès à la variable originale, mais plutôt à la variable modifiée.

Par exemple, la parallélisation suivante sur la première boucle commencerait 4 threads qui prendraient les valeur i = 0,1,2 et 3. Les quatre threads initialiseront j à 0, mais le plus rapide à effectuer les calculs terminera la boucle intérieure avec j++, augmentant sa valeur à 1. Si les autres processus sont encore en train d'utiliser la variable j, celle-ci ne sera plus à 0, mais à 1 puisqu'il n'y a qu'une seule case mémoire pour j que tout les processus modifient.

```
#pragma parallel for
for i = 0 ; i < p ; i++
      for j = 0 ; j < q ; j++
            Calcul
      end
end
```

Qui plus est, le deuxième thread à terminer et à effectuer l'incrémentation j++ ne mettra plus la valeur à 1, mais bien à 2, puis à 3 pour le 3e, etc. Ainsi, même si tout les processus prennent exactement le même temps de calcul, la valeur de j ne sautera pas de 0 à 1, mais de 0 à 4 (nombre de threads), alors que les threads auront besoin de j=1 à la fin de leurs premiers calculs.

Pour résoudre ce problème nous avons utilisé le concept de variables privées et de variables partagées. Une variable privée est déclarée au début de la parallélisation et existe à l'intérieur de chaque thread. Des copies différentes de la variables peuvent ainsi avoir des valeurs différentes pour chaque thread et ne s'influenceront pas les uns les autres. Les variables qui ne sont pas à risque d'être écrasées par un autre processus demeurent partagées. Dans notre cas, nous devons privatiser les incrémentations internes (j) de nos boucles et une variable artifice qui stocke la contribution de chaque zone du temps $t+1$ au temps t, mais gardons les matrices de transition, les facteurs d'interpolation et la grille de prix partagées, car les deux premières ne sont que lues et jamais modifiées dans la boucle et que la dernière n'est modifiée que point par point, ainsi ne créera pas de conflit entre deux threads.

4.2 MPI

Considérant les contraintes économiques à la construction d'ordinateurs à mémoire partagée, une solution est d'utiliser plusieurs ordinateurs, c'est-à-dire de mettre plusieurs processeurs différents côte à côte. Il n'importe alors plus de vérifier que les coeurs viennent modifier la mémoire commune et s'immiscent dans les calculs les uns des autres, mais plutôt qu'ils puissent se communiquer l'information qu'ils calculent chacun de leur côté et qu'ils aient tous accès à l'information nécessaire pour effectuer leurs calculs.

Il est donc nécessaire pour l'utilisation de la librairie MPI d'identifier un processeur qui aura pour tâche d'effectuer la communication entre tous les autres. Le processeur en charge de la communication est appelé le maître et les autres sont les esclaves. Puisque chaque processeur fonctionne indépendamment, tout calcul fait sur un processeur est complètement ignoré par les autres. Or, nous avons bien identifié que la grille est parallélisable à un pas de temps donné, mais que la grille complète du temps $t + \Delta t$ est nécessaire au calcul de la grille du temps t. Le maître a alors pour tâche de distribuer la grille à travers les esclaves, de récupérer les résultats de leurs calculs et de coller les différents morceaux de la grille ensemble. Il doit ensuite renvoyer la grille complète à tout les esclaves, afin qu'ils puissent à nouveau calculer la

portion de grille qui leur est assignée.

Nous paralléliserons à la fois le calcul des valeurs au temps t à partir des valeurs au temps $t + \Delta t$ et le calcul des facteurs alpha impliqués dans l'interpolation, car ces deux valeurs se calculent indépendamment des autres points qui leur sont adjacents. Appelons n_proc le nombre de processus totaux. L'algorithme de parallélisation est comme suit :

(1) Calcul en série (grilles, matrices de transitions et matrice des prix d'exercice à l'expiration)

(2) Calcul par les esclaves des facteurs alphas. Chaque processeur se fait assigner l'entier supérieur à $p*q/(n\_proc-1)$ points. Pour chacun de ces points, 4 facteurs seront calculés pour l'interpolation linéaire et 9 pour l'interpolation quadratique. Les facteurs sont stockés dans des matrices $4 \times$ *nombre de points par processeur* ou $9 \times$ *nombre de points par processeur*.

(3) Envoi par les esclaves de la matrices des facteurs au maître.

(4) Réception par le processeur maître des calculs des esclaves et complétion des 4 ou des 9 matrices de facteurs

(5) Envoi des 4 ou des 9 matrices de facteurs du maître vers tous les esclaves.

(6) Calcul par les esclaves de $p*q/(n\_proc-1)$ valeurs au temps t à partir des valeurs au temps $t + \Delta t$ et des matrices de facteurs. Les valeurs sont stockées dans des vecteurs de taille $p * q/(n\_proc - 1)$.

(7) Envoi des vecteurs des prix au temps t des esclaves vers le maître.

(8) Réception par le processeur maître des calculs des esclaves et complétion de la matrice des prix au temps t.

(9) Envoi de la matrice de prix vers tous les esclaves.

(10) Si $t = 0$, arrêt, sinon retour à l'étape 2.

Nous utilisons pour rouler notre algorithme la grappe Graham, située à l'Université de Waterloo et accessible à travers Calcul Canada. Celle-ci possède un total de 36160 coeurs. Les résultats ainsi que les temps de calcluls requis pour les obtenir sont maintenant donnés dans la prochaine section.

5 Investigation numérique

Nous testons maintenant notre algorithme pour voir sa performance. Nous commencerons par examiner le cas européen afin d'évaluer la qualité des résultats face à une forme fermée. Les options américaines seront ensuite évaluées par des options bermudéennes avec un grand nombre de dates d'exercices. Nous comparons nos résultats à chaque fois à ceux de BBR. Finalement, nous discuterons de la parrallélisation de la programmation et analyserons les temps de calculs avec l'utilisation des librairies présentées à la section 4.

5.1 Résultats

Pour s'assurer de l'exactitude de notre méthode, nous évaluons d'abord une option pour laquelle il existe une forme fermée, à savoir une option put-on-min européenne, une option qui permet de vendre le minimum de deux sous-jacents à un prix K après un temps T, sans possibilité d'exercice sauf à l'expiration.

Le tableau 1 représente les résultats de BBR, de Boyle et la forme fermée de Schultz. Les paramètres sont $S_0^1 = 40$, $S_0^2 = 40$, $d_1 = 0$, $d_2 = 0$, $\sigma_1 = 0.2$, $\sigma_2 = 0.3$, $\rho = 0.5$, $r = 0.04879$, $T = 0.58333$. Le tableau 2 représente les résultats de notre algorithme. Afin de comparer correctement les deux algorithmes, nous avons utilisé exactement le même nombre de processus parallèles que BBR pour les grilles de même taille.

Table 1: Option européenne put-on-min algorithme BBR

	Grille $p*q$				
	72^2	144^2	300^2	Boyle	Forme fermée
$K = 35$	1.411	1.392	1.388	1.425	1.387
$K = 40$	3.837	3.805	3.800	3.778	3.798
$K = 45$	7.543	7.508	7.501	7.475	7.500
Nb Coeurs	576	1728	1800		
Temps	0.93	5.40	34.48		10 pas
$K = 35$	1.504	1.410	1.391	1.392	1.387
$K = 40$	3.970	3.832	3.805	3.795	3.798
$K = 45$	7.694	7.537	7.507	7.499	7.500
Nb Coeurs	576	1728	1800		
Temps	2.14	10.29	67.01		50 pas

TABLE 2: Option européenne put-on-min algorithme ACP

	Grille $p*q$			Grille $p*q$		
	Interpolation bi-linéaire			Interpolation bi-quadratique		
	72^2	144^2	300^2	72^2	144^2	300^2
$K = 35$	1.438	1.400	1.390	1.386	1.387	1.387
$K = 40$	3.869	3.816	3.803	3.798	3.799	3.799
$K = 45$	7.570	7.517	7.504	7.500	7.500	7.500
Nb Coeurs	576	1728	1800	576	1728	1800
Temps (10 pas)	0.18	0.57	4.31	0.30	1.47	11.17
$K = 35$	1.638	1.451	1.402	1.371	1.386	1.387
$K = 40$	4.143	3.887	3.819	3.788	3.797	3.799
$K = 45$	7.844	7.588	7.520	7.495	7.499	7.500
Nb Coeurs	576	1728	1800	576	1728	1800
Temps (50 pas)	0.25	1.42	20.49	0.58	2.56	34.27

Nous pouvons observer de ces tableaux qu'à grille égale, la méthode de l'ACP est légèrement moins précise que la méthode BBR, bien qu'avec davantage de point la différence diminue. Nous remarquons par contre que les temps de calculs sont significativement réduits. Il est ainsi clair que l'algorithme ACP est au final plus efficace que celui de BBR.

Par ailleurs, nous avons aussi testé une interpolation quadratique, avec pour hypothèse que cette interpolation épouserait mieux les courbes de la fonction valeur et minimiserait ainsi les erreurs. Les résultats pointent en effet dans cette direction, puisqu'avec 72 points avec une interpolation quadratique nous réussissons à avoir une précision comparable à celle que nous obtenons avec 300 points d'interpolation linéaire, mais aussi avec l'algorithme BBR. Si nous comparons les temps de calcul de BBR à 300 points à ceux de l'algorithme ACP à 72, nous obtenons pour un niveau de précision comparable un gain de temps de plus de 99%.

TABLE 3: Option américaine call-on-min algorithme BBR

			Grille $p*q$		Borne	Borne
S_0^2	S_0^1	72^2	144^2	300^2	inférieure	supérieure
90	90	0.893	0.799	0.782	0.63	0.97
	100	1.691	1.560	1.536	1.27	1.88
	110	2.526	2.364	2.335	1.92	2.77
	120	3.177	2.984	2.950	2.49	3.37
100	100	3.392	3.233	3.205	2.62	3.98
	110	5.341	5.174	5.145	4.26	6.36
	120	6.680	6.608	6.576	5.51	7.58
	130	7.639	7.402	7.361	6.36	7.95
110	110	9.455	9.362	9.348	10.00	13.66
	120	12.225	12.118	12.098	10.37	14.50
	130	13.511	13.335	13.304	11.58	14.53
	140	14.147	13.912	13.869	12.51	14.44
Nb Coeurs		576	1728	1800		
Temps		2.93	15.05	78.76	100 pas	

Le tableau 3 contient les résultats pour un Call-on-min américain de BBR, comparé aux bornes de Detemple et al. (03). Les paramètres sont $K = 100$, $d_1 = 0.05$, $d_2 = 0.05$, $\sigma_1 = 0.2$, $\sigma_2 = 0.2$, $\rho = 0$, $r = 0.06$, $T = 1$. Le tableau 4 contient les résultats de l'ACP pour la même option.

TABLE 4: Option américaine call-on-min algorithme ACP

		Grille $p*q$			Grille $p*q$		
		Interpolation bi-linéaire			Interpolation bi-quadratique		
S_0^2	S_0^1	72^2	144^2	300^2	72^2	144^2	300^2
90	90	1.072	0.854	0.795	0.799	0.783	0.779
	100	1.897	1.623	1.553	1.528	1.526	1.530
	110	2.795	2.448	2.358	2.343	2.325	2.329
	120	3.513	3.096	2.979	2.964	2.943	2.942
100	100	3.668	3.324	3.227	3.247	3.215	3.202
	110	5.500	5.253	5.161	5.078	5.141	5.135
	120	7.063	6.720	6.600	6.549	6.575	6.566
	130	8.041	7.548	7.396	7.361	7.355	7.350
110	110	10.000	10.000	10.000	10.000	10.000	10.000
	120	12.406	12.162	12.115	12.186	12.095	12.098
	130	13.853	13.447	13.332	13.370	12.309	13.299
	140	14.552	14.049	13.905	13.858	13.856	13.860
Nb Coeurs		576	1728	1800	576	1728	1800
Temps (100 pas)		0.59	2.92	38.12	1.46	6.96	109.58

Nos résultats sont conséquents avec ceux de BBR. Comme nous l'avons observé dans le cas européen, notre algorithme quadratique à 72 points ou à 144 points donne des résultats comparables à l'utilisation de 300 points pour BBR. Or, nous pouvons observer sur cette option que BBR ne semble pas être arrivé à une convergence complète à 300 points. Nos résultats à 300 semblent à cet effet une étape supplémentaire lorsqu'on les compare à la série de raffinement des points de 72 à 144 à 300 de BBR. Ils sont évidemment dans les bornes de Detemple et al., qui sont assez larges.

Les tableaux 5 à 8 présentent les résultats pour une option américaine Put-on-min. Pour les tableaux 5 et 6, les paramètres sont donnés dans le tableau 1, alors que les tableaux 7 et 8 ont pour paramètres $K = 100$, $d_1 = 0$, $d_2 = 0$, $\sigma_1 = 0.6$, $\sigma_2 = 0.6$, $\rho = 0$, $r = 0.06$, $T = 0.5$. Les intervalles de confiances à 95% de Rogers, Jin et al. sont données dans le tableau 7 ainsi que les résultats de Hartley.

TABLE 5: Option américaine put-on-min algorithme BBR

	Grille $p*q$			
	72^2	144^2	300^2	Boyle
$K=35$	1.436	1.416	1.413	1.450
$K=40$	3.918	3.887	3.881	3.870
$K=45$	7.713	7.678	7.671	7.645
Temps	1.01	5.79	36.63	10 pas
$K=35$	1.535	1.440	1.422	1.423
$K=40$	4.064	3.926	3.899	3.892
$K=45$	7.880	7.727	7.697	7.689
Nb Coeurs	576	1728	1800	
Temps	1.96	9.94	66.45	50 pas

TABLE 6: Option américaine put-on-min algorithme ACP

	Grille $p*q$			Grille $p*q$		
	Interpolation bi-linéaire			Interpolation bi-quadratique		
	72^2	144^2	300^2	72^2	144^2	300^2
$K=35$	1.463	1.425	1.415	1.410	1.412	1.412
$K=40$	3.950	3.897	3.884	3.879	3.880	3.880
$K=45$	7.737	7.687	7.673	7.669	7.670	7.669
Nb Coeurs	576	1728	1800	576	1728	1800
Temps (10 pas)	0.08	0.35	7.88	0.07	0.44	6.51
$K=35$	1.671	1.482	1.432	1.400	1.416	1.417
$K=40$	4.235	3.981	3.913	3.880	3.891	3.893
$K=45$	8.022	7.775	7.710	7.683	7.689	7.690
Nb Coeurs	576	1728	1800	576	1728	1800
Temps (50 pas)	0.30	2.15	40.59	0.35	2.36	29.28

TABLE 7: Option américaine put-on-min algorithme BBR

	Grille $p*q$					
(S_0^1, S_0^2)	72^2	144^2	300^2	Rogers	Jin et al.	Hartley
(80, 80)	37.938	37.416	37.312	[37.35, 37.65]	[37.1000, 37.4022]	37.30
(80, 100)	32.775	32.205	32.091	[32.12, 32.26]	[31.8421, 32.1451]	32.08
(80, 120)	29.826	29.265	29.152	[29.18, 29.32]	[28.8860, 29.2434]	29.14
(100, 100)	25.889	25.205	25.066	[24.93, 25.23]	[24.8296, 25.1576]	25.06
(100, 120)	21.779	21.067	20.920	[20.89, 21.09]	[20.6850, 20.9932]	20.91
(120,120)	16.864	16.102	15.942	[15.99, 16.19]	[15.6737, 16.0017]	15.92
Nb Coeurs	576	1728	1800			
Temps	1.66	8.70	46.72		51 pas	

TABLE 8: Option américaine put-on-min algorithme ACP

	Grille $p*q$			Grille $p*q$		
	Interpolation bi-linéaire			Interpolation bi-quadratique		
(S_0^1, S_0^2)	72^2	144^2	300^2	72^2	144^2	300^2
(80, 80)	38.857	37.694	37.383	37.207	37.279	37.288
(80, 100)	33.740	32.499	32.166	31.976	32.057	32.065
(80, 120)	30.739	29.542	29.222	29.037	29.118	29.126
(100, 100)	26.964	25.534	25.152	24.894	25.017	25.033
(100, 120)	22.848	21.394	21.005	20.735	20.870	20.885
(120,120)	17.940	16.432	16.027	15.725	15.886	15.903
Nb Coeurs	576	1728	1800	576	1728	1800
Temps (51 pas)	0.18	1.40	16.44	0.40	2.42	29.74

Les deux derniers contrats évalués sont des options américaines Call-on-max. Ils ont tout deux pour paramètres $K = 100$ (sauf pour le deuxième contrat où il varie), $d_1 = 0.1$, $d_2 = 0.1$, $\sigma_1 = 0.2$, $\sigma_2 = 0.2$, $\rho = 0.3$, $r = 0.05$, $T = 1$.

TABLE 9: Option américaine call-on-max algorithme BBR

	Grille $p*q$				
(S_0^1, S_0^2)	72^2	144^2	300^2	Broadie et Glasserman	Jin et al.
(70, 70)	0.244	0.239	0.238	[0.231, 0.266]	[0.2224, 0.2436]
(80, 80)	1.296	1.273	1.269	[1.191, 1.287]	[1.2159, 1.2865]
(90, 90)	4.179	4.119	4.108	[3.923, 4.216]	[4.0141, 4.1365]
(100, 100)	9.522	9.442	9.427	[9.046, 9.673]	[9.2672, 9.4302]
(110, 110)	17.216	17.072	17.040	[16.516, 17.504]	[16.8415, 17.0037]
(120, 120)	26.449	26.237	26.161	[25.471, 26.643]	[25.8637, 26.1471]
(130, 130)	36.537	36.176	36.033	[35.161, 36.646]	[35.6321, 35.9155]
Nb Coeurs	576	1728	1800		
Temps	1.28	7.35	118.78	4 pas	

TABLE 10: Option américaine call-on-max algorithme ACP

	Grille $p*q$			Grille $p*q$		
	Interpolation bi-linéaire			Interpolation bi-quadratique		
(S_0^1, S_0^2)	72^2	144^2	300^2	72^2	144^2	300^2
(70, 70)	0.248	0.240	0.238	0.238	0.238	0.238
(80, 80)	1.299	1.276	1.270	1.268	1.268	1.268
(90, 90)	4.164	4.120	4.109	4.104	4.105	4.106
(100, 100)	9.495	9.439	0.425	9.416	9.420	9.421
(110, 110)	17.099	17.042	17.028	17.016	17.022	17.023
(120,120)	26.192	26.139	26.126	26.114	26.120	26.121
(130,130)	36.013	35.960	35.948	35.938	35.942	35.944
Nb Coeurs	576	1728	1800	576	1728	1800
Temps (4 pas)	0.01	0.11	1.02	0.03	0.18	1.86

TABLE 11: Option américaine call-on-max algorithme BBR

(S_0^1, S_0^2)	K	Grille $p*q$ 72^2	144^2	300^2	Raymard et Zwecher	Binomial
(100,110)	70	40.998	40.940	40.931	[40.852, 40.986]	40.930
	100	14.037	13.959	13.946	[13.923, 14.075]	13.945
	130	2.152	2.106	2.097	[2.173, 2.263]	2.093
(100,100)	70	34.601	34.537	34.526	[34.453, 34.605]	34.525
	100	9.632	9.559	9.547	[9.542, 9.694]	9.543
	130	1.100	1.071	1.065	[1.125, 1.187]	1.062
(100,90)	70	30.838	30.786	30.777	[30.698, 30.824]	30.776
	100	7.237	7.172	7.160	[7.169, 7.303]	7.158
	130	0.706	0.684	0.680	[0.732, 0.786]	0.677
Nb Coeurs	576	1728	1800			
Temps	1.25	5.09	105		10 pas	

TABLE 12: Option américaine call-on-max algorithme ACP

		Grille $p*q$ Interpolation bi-linéaire			Grille $p*q$ Interpolation bi-quadratique		
(S_0^1, S_0^2)	K	72^2	144^2	300^2	72^2	144^2	300^2
(100,110)	70	41.032	40.955	40.935	40.927	40.929	40.929
	100	14.130	13.990	13.954	13.935	13.942	13.943
	130	2.212	2.124	2.101	2.095	2.095	2.094
(100,100)	70	34.647	34.555	34.531	34.513	34.522	34.523
	100	9.734	9.591	9.554	9.535	9.542	9.543
	130	1.142	1.083	1.068	1.065	1.064	1.064
(100,90)	70	30.864	30.797	30.780	30.768	30.775	30.775
	100	7.322	7.199	7.167	7.153	7.157	7.157
	130	0.733	0.692	0.682	0.681	0.679	0.678
Nb Coeurs		576	1728	1800	576	1728	1800
Temps (10 pas)		0.04	0.32	3.79	0.07	0.46	5.79

Les résultats sont encore concluants avec tout les types de contrats et de paramètres étudiés. Nous observons un gain de temps constant entre la

méthode de BBR et la nôtre, et des gains de précision importants obtenus par l'utilisation de l'interpolation quadratique.

5.2 Temps de calculs

Notre algorithme s'ajustant parfaitement à tout type de fonction d'exercice, les paramètres qui influencent le temps de calcul sont la taille de la grille et l'interpolation. Nous pouvons donc examiner l'effet de parallélisation que nous obtenons à travers OpenMP, utilisé sur PC et la librairie MPI, utilisée sur les serveurs de Calcul Canada. Comme précédemment, les grilles de 72, 144 et 300 points utilisent respectivement 576, 1728 et 1800 processus parallèles. Nous rapportons les résultats de pour les différentes tailles de grilles dans les tableaux 13, 14 et 15.

TABLE 13: Temps de calcul pour l'algorithme ACP $p = q = 72$

	Interpolation linéaire			Interpolation quadratique		
NE	Séquentiel	OpenMP	MPI	Séquentiel	OpenMP	MPI
4	4.52	1.65	0.01	8.14	6.29	0.03
10	13.39	6.04	0.1	24.51	16.60	0.15
50	75.47	27.92	0.28	137.11	76.33	0.47
51	75.11	28.22	0.18	139.21	88.80	0.40
100	149.76	55.50	0.59	268.80	180.60	1.46

TABLE 14: Temps de calcul pour l'algorithme ACP $p = q = 144$

	Interpolation linéaire			Interpolation quadratique		
NE	Séquentiel	OpenMP	MPI	Séquentiel	OpenMP	MPI
4	73	25	0.11	126	100	0.18
10	217	77	0.41	380	274	0.79
50	1268	540	1.79	2118	1572	2.46
51	1421	607	1.40	2264	1656	2.42
100	2894	1166	2.92	4587	3231	6.96

TABLE 15: Temps de calcul pour l'algorithme ACP $p = q = 300$

	Interpolation linéaire			Interpolation quadratique		
NE	Séquentiel	OpenMP	MPI	Séquentiel	OpenMP	MPI
4	1439	601	1.02	2490	1883	1.86
10	4127	2318	4.33	7516	5427	7.82
50	23454	10804	30.54	42244	24033	31.78
51	22259	10719	16.44	43897	28760	29.74
100	44925	21156	38.12	85825	61795	109.58

Nous pouvons constater à travers ces tableaux que même si le même ordinateur est utilisé pour les colonnes PC et OpenMP, il est possible d'obtenir un gain de temps intéressant simplement en utilisant la librairie OpenMP pour paralléliser les boucles et utiliser les 4 processeurs de la machine. Il est ainsi facile de paralléliser des boucles avec itérations indépendantes. Cette approche devrait être considérée pour tout programme qui s'y prête, car elle offre presque gratuitement un gain de calcul intéressant.

Nous remarquons aussi un gain d'efficacité beaucoup plus important pour l'interpolation linéaire que quadratique. Les temps avec l'utilisation d'OpenMP peuvent être presque divisés par 3 lors de l'interpolation linéaire, alors qu'avec l'interpolation quadratique ils ne sont divisés que par deux dans les meilleurs des cas.

Un programme parfaitement parallélisable devrait, en théorie, avoir des gains d'efficacité linéaires, c'est-à-dire qu'en doublant le nombre de ressources nous divisons en deux le temps de calcul. Or, comme nous l'avons décrit aux sections 3 et 4, notre algorithme est en deux parties. La première étant effectuée en série, elle n'est aucunement affectée par l'ajout de ressources.

Ce phénomène a été décrit dans les années 60 par Gene Amdahl qui a formulé une loi empirique qui porte son nom depuis. Il pose une limite théorique à l'accélération du programme obtenu par l'ajout de ressources, en arguant que la portion séquentielle du programme sera inévitablement le boulet qui empêchera l'ordinateur d'améliorer ses performances. Selon lui, si $p\%$ du programme est séquentiel, alors le temps de calcul pourra être divisé par au plus $\frac{1}{p}$. Par exemple, si 1% du temps de calcul est séquentiel, alors le temps de calcul ne pourra être divisé que par 100, peu importe combien de ressources seront allouées car rien ne peut être fait pour se débarasser de ce

1%.

Or, un des points fort de notre algorithme en comparaison avec BBR est justement la diminution de la partie séquentielle au profit de la partie parallélisable. Nous ne calculons pas, contrairement à eux, des matrices de probabilités de transitions à 4 dimensions, mais plutôt à deux, que nous multiplions ensuite pour calculer les espérances. Ainsi, la vaste majorité de nos calculs se font dans la section parallèle.

Nous pouvons observer ce phénomène de deux façon dans nos résultats. D'abord, nous divisons nos temps de calculs par plus de 1000 dans certains cas, ce qui suggère selon la loi d'Amdahl que notre portion séquentielle prend au plus $0,1\%$ du temps de calcul de notre algorithme. Par ailleurs, nous observons que nos temps de calculs sont environ proportionnels à la quantité de pas de temps effectués. Puisque les pas de temps correspondent à la partie parallélisable, cela confirme d'une autre façon que notre partie séquentielle est négligeable.

Il est important de spécifier par contre que la loi d'Amdhal est bien davantage une borne qu'un objectif. Dans les faits les processeurs doivent se communiquer entre eux, ce qui engendre évidemment une perte de temps et de ressource. En dépit de ces temps, notre algorithme fait tout de même une utilisation extrêmement efficace des ressources ajoutées. Il est aussi intéressant de noter que bien que l'utilisation de threads ne nécessite pas de communication, les résultats sont moins bons en proportion de l'ajout de ressources.

Par ailleurs, une autre avancée importante de notre algorithme est la réduction drastique de la mémoire requise dû à la division des dimension par l'ACP. Cela nous permet non seulement de facilement rouler notre programme sur un ordinateur portable, mais nous permettrait aussi d'augmenter facilement la grille davantage, car nous sommes très loin des capacités maximales en terme de mémoire des ordinateurs, même portables. Pour donner un ordre de grandeur, à $p = q = 300$, nos 4 matrices de transitions (pour l'interpolation linéaire) comportent 90000 éléments, donc 360000 en tout. BBR n'a que 2 matrices, mais chacune d'elle contient 8100000000 éléments, pour un total de $1,62 \times 10^{10}$ éléments à stocker. Notre algorithme pourrait effectuer une grille de $p = q = 63000$ et quand même nécessiter moins de mémoire.

6 Conclusion

Dans ce mémoire nous avons exploré le problème de l'évaluation d'options américaines par programmation dynamique et différences finies avec l'analyse des composantes principales. L'ajout de l'ACP dans la méthode développée par BBR nous a permis de diviser les matrices de probabilité, ce qui a eu pour effet de réduire le temps de calcul ainsi que la mémoire nécessaire au calcul.

Nous avons par ailleurs amélioré l'interpolation utilisée par rapport à BBR, ce qui nous a donné un gain de précision impressionnant, qui compense largement le temps de calcul additionnel. Ce changement, couplé à l'effet de l'ACP nous permet d'avoir plus de précision en utilisant une grille deux fois plus petite.

Nous avons aussi ajusté notre algorithme au calcul parallèle afin d'en tirer profit. Cela nous a permis de comparer nos temps de calculs directement avec BBR, plutôt que de seulement comparer la qualité de nos résultats en fonction de la taille de la grille.

Notre méthode en est une de très peu qui ont exploré l'utilisation de l'ACP appliquée à l'évaluation d'options. Nous n'avons d'ailleurs pas utilisé l'ACP comme elle l'est souvent, à savoir pour couper des dimensions au problème, mais plutôt pour sa capacité à former des variables orthogonales.

La littérature rapporte que la réduction simple de dimensions n'est pas efficace pour l'évaluation d'options car trop d'informations se trouvent dans la succession de composantes à petite variance. Ainsi, la principale utilisation de l'ACP a été par Reisinger et Wittum qui ont tenté d'utiliser l'ACP pour explorer rapidement l'ensemble de l'espace d'état. C'est donc dire qu'eux non plus ne coupent pas directement des dimensions.

Il est aussi important de mentionner que contrairement à certaines mesures physiques, la finance est soumise à une grande quantité de sources de risques, et qu'en conséquence les corrélations peuvent être plutôts faibles. Même si nous parvenons à extraire certains mouvements globaux, comme le risque de marché, et certains mouvements de groupes, par exemple la performance des secteurs pétrolier, technologique ou bancaire, il existe toujours le risque idiosynchratique qui affecte chaque mouvement indépendamment. Or, l'ACP n'est pas construite pour capturer ces particularités.

Nous avons par contre fait des gains extraordinaires en examinant la qualité de l'interpolation utilisée et n'avons qu'utilisé une interpolation quadratique. Il y a lieu d'examiner les méthodes d'intégration numérique que nous

utilisons dans l'espoir d'améliorer nos résultats puisqu'il est raisonnable de croire que nous n'avons pas atteint un plateau à ce niveau.

7 Annexe

Soit $ln(C_t^i)$ le log-mouvement de la composante principale i, pour $i = 1, 2$.

Nous rappellons que

$ln(C_t^1) \sim N(\mu_1^1(ln(S_t^1)+(r-d_1-\frac{\sigma_1^2}{2})\Delta t)+\mu_2^1(ln(S_t^2)+(r-d_2-\frac{\sigma_2^2}{2})\Delta t); (\mu_1^1)^2\sigma_1^2\Delta t+(\mu_2^1)^2\sigma_2^2\Delta t + 2\rho\mu_1^1\mu_2^1\sigma_1\sigma_2\Delta t)$

Pour faciliter la lecture nous notons donc :

$\mu = \mu_1^1(ln(S_t^1) + (r - d_1 - \frac{\sigma_1^2}{2})\Delta t) + \mu_2^1(ln(S_t^2) + (r - d_2 - \frac{\sigma_2^2}{2})\Delta t)$

et

$\sigma = (\mu_1^1)^2\sigma_1^2\Delta t + (\mu_2^1)^2\sigma_2^2\Delta t + 2\rho\mu_1^1\mu_2^1\sigma_1\sigma_2\Delta t$

Calcul de T_0^{1ki}

$$T_0^{1ki} = \mathbb{E}^{\mathbb{Q}}[\mathbb{I}(ln(C_{t+\Delta t}^1) \in [a_i', a_{i+1}'])|ln(C_t^1) = a_k']$$
$$= \mathbb{Q}[a_i' \leq ln(C_{t+\Delta t}^1) \leq a_{i+1}'|ln(C_t^1) = a_k']$$
$$= \mathbb{Q}[ln(C_{t+\Delta t}^1) \leq a_{i+1}'|ln(C_t^1) = a_k'] - \mathbb{Q}[ln(C_{t+\Delta t}^1) \leq a_i'|ln(C_t^1) = a_k']$$
$$= \mathbb{Q}[\frac{ln(C_{t+\Delta t}^1)-\mu}{\sigma} \leq \frac{a_{i+1}'-\mu}{\sigma}|ln(C_t^1) = a_k'] - \mathbb{Q}[\frac{ln(C_{t+\Delta t}^1)-\mu}{\sigma} \leq \frac{a_i'-\mu}{\sigma}|ln(C_t^1) = a_k']$$
$$= \Phi(\frac{a_{i+1}'-\mu}{\sigma}) - \Phi(\frac{a_i'-\mu}{\sigma})$$

Calcul de T_1^{1ki}

$$T_1^{1ki} = \mathbb{E}^{\mathbb{Q}}[C^1_{t+\Delta t}\mathbb{I}(ln(C^1_{t+\Delta t}) \in [a'_i, a'_{i+1}])|ln(C^1_t) = a'_k]$$

$$= \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} C^1_{t+\Delta t} f(ln(C^1_{t+\Delta t}))dln(C^1_{t+\Delta t})$$

$$= \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} exp(a'_k + \mu + \sigma z)\phi(z)dz$$

$$= exp(a'_k + \mu) \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} exp(\frac{-z^2}{2} + \frac{2\sigma z}{2} - \frac{\sigma^2}{2} + \frac{\sigma^2}{2})dz$$

$$= exp(a'_k + \mu + \frac{\sigma^2}{2}) \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} exp(\frac{-(z-\sigma)^2}{2})dz$$

$$= exp(a'_k + \mu + \frac{\sigma}{2})(\Phi(\frac{a'_{i+1}-\mu}{\sigma} - \sigma) - \Phi(\frac{a'_i-\mu}{\sigma} - \sigma))$$

Calcul de T_2^{1ki}

$$T_2^{1ki} = \mathbb{E}^{\mathbb{Q}}[(C^1_{t+\Delta t})^2\mathbb{I}(ln(C^1_{t+\Delta t}) \in [a'_i, a'_{i+1}])|ln(C^1_t) = a'_k]$$

$$= \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} (C^1_{t+\Delta t})^2 f(ln(C^1_{t+\Delta t}))dln(C^1_{t+\Delta t})$$

$$= \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} exp(2a'_k + 2\mu + 2\sigma z)\phi(z)dz$$

$$= exp(2a'_k + 2\mu) \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} exp(\frac{-z^2}{2} + \frac{4\sigma z}{2} - \frac{4\sigma^2}{2} + \frac{4\sigma^2}{2})dz$$

$$= exp(2a'_k + 2\mu + 2\sigma^2) \int_{\frac{a'_i-\mu}{\sigma}}^{\frac{a'_{i+1}-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} exp(\frac{-(z-2\sigma)^2}{2})dz$$

$$= exp(2a'_k + 2\mu + 2\sigma)(\Phi(\frac{a'_{i+1}-\mu}{\sigma} - 2\sigma) - \Phi(\frac{a'_i-\mu}{\sigma} - 2\sigma))$$

Calcul de l'espérance (avec interpolation linéaire, l'interpolation quadratique est analogue)

$$\mathbb{E}_t^{\mathbb{Q}}[e^{-r\Delta t}W_{t+\Delta t}(C^1_{t+\Delta t}, C^2_{t+\Delta t})|ln(c^1_t) = a'_k, ln(c^2_t) = b'_l]$$

$$\approx \mathbb{E}_t^{\mathbb{Q}}[e^{-r\Delta t}\sum_{ij}(\alpha_{00} + \alpha_{10}C^1_{t+\Delta t} + \alpha_{01}C^2_{t+\Delta t} + \alpha_{11}C^1_{t+\Delta t}C^2_{t+\Delta t})\mathbb{I}(ln(C^1_{t+\Delta t}) \in R'_{i,j})|ln(C^1_t) = a'_k, ln(C^2_t) = b'_k]$$

$$= e^{-r\Delta t}(\sum_{ij}\alpha_{00}\mathbb{E}_t^{\mathbb{Q}}[\mathbb{I}(a'_i \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1} \cap b'_j \leq ln(C^2_{t+\Delta t}) \leq z_{j+1})|ln(C^1_t) = a'_k, ln(C^2_t) = b'_k] +$$
$$\alpha_{10}\mathbb{E}_t^{\mathbb{Q}}[C^1_{t+\Delta t}\mathbb{I}(a'_i \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1} \cap b'_j \leq ln(C^2_{t+\Delta t}) \leq b'_{j+1})|ln(C^1_t) = a'_k, ln(C^2_t) = b'_k] +$$
$$\alpha_{01}\mathbb{E}_t^{\mathbb{Q}}[C^2_{t+\Delta t}\mathbb{I}(a'_i \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1} \cap b'_j \leq ln(C^2_{t+\Delta t}) \leq b'_{j+1})|ln(C^1_t) = a'_k, ln(C^2_t) = b'_k] +$$
$$\alpha_{11}\mathbb{E}_t^{\mathbb{Q}}[C^1_{t+\Delta t}C^2_{t+\Delta t}\mathbb{I}(a'_i \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1} \cap b'_j \leq ln(C^2_{t+\Delta t}) \leq b'_{j+1})|ln(C^1_t) = a'_k, ln(C^2_t) = b'_k])$$

$$= e^{-r\Delta t}(\sum_{ij}\alpha_{00}\mathbb{E}_t^{\mathbb{Q}}[\mathbb{I}(a' \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1}|ln(C^1_t) = a'_k]\mathbb{E}_t^{\mathbb{Q}}[\mathbb{I}(b'_j \leq ln(C^1_{t+\Delta t}) \leq b'_{j+1}|ln(C^2_t) = b'_k] +$$
$$\alpha_{10}\mathbb{E}_t^{\mathbb{Q}}[C^1_{t+\Delta t}\mathbb{I}(a'_i \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1}|ln(C^1_t = a'_k]\mathbb{E}_t^{\mathbb{Q}}[\mathbb{I}(b'_j \leq ln(C^1_{t+\Delta t}) \leq b'_{j+1}|ln(C^2_t) = b'_k] +$$
$$\alpha_{01}\mathbb{E}_t^{\mathbb{Q}}[\mathbb{I}(a'_i \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1}|ln(C^1_t) = a'_k]\mathbb{E}_t^{\mathbb{Q}}[C^2_{t+\Delta t}\mathbb{I}(z_j \leq ln(C^1_{t+\Delta t}) \leq b'_{j+1}|ln(C^2_t) = b'_k] +$$
$$\alpha_{11}\mathbb{E}_t^{\mathbb{Q}}[C^1_{t+\Delta t}\mathbb{I}(a'_i \leq ln(C^1_{t+\Delta t}) \leq a'_{i+1}|ln(C^1_t) = a'_k]\mathbb{E}_t^{\mathbb{Q}}[C^1_{t+\Delta t}\mathbb{I}(b'_j \leq ln(C^1_{t+\Delta t}) \leq b'_{j+1}|ln(C^2_t) = b'_k])$$

$$= e^{-r\Delta t}(\sum_{ij}\alpha_{00}T_0^{1i}T_0^{2j} + \alpha_{10}T_1^{1i}T_0^{2j} + \alpha_{01}T_0^{1i}T_1^{2j} + \alpha_{11}T_1^{1i}T_1^{2j})$$

Système d'équation pour l'interpolation bi-quadratique

Pour l'interpolation bi-quadratique, la fonction w sera estimée avec 9 paramètres α de la façon suivante :

$$\hat{w}_{t+\Delta t}(C^1_{t+\Delta t}, C^2_{t+\Delta t}) \approx$$
$$\sum_{ij} \alpha^{00}_{ij} + \alpha^{10}_{ij} C^1_{t+\Delta t} + \alpha^{01}_{ij} C^2_{t+\Delta t} + \alpha^{11}_{ij} C^1_{t+\Delta t} C^2_{t+\Delta t} + \alpha^{20}_{ij} (C^1_{t+\Delta t})^2 +$$
$$\alpha^{21}_{ij} (C^1_{t+\Delta t})^2 C^2_{t+\Delta t} + \alpha^{12}_{ij} C^1_{t+\Delta t} (C^2_{t+\Delta t})^2 + \alpha^{02}_{ij} (C^2_{t+\Delta t})^2 + \alpha^{22}_{ij} (C^2_{t+\Delta t})^2 (C^2_{t+\Delta t})^2$$

ce qui implique la résolution du système suivant :

$$\begin{cases} \hat{w}_{t+\Delta t}(a'_{i-1}, b'_{j-1}) = \tilde{w}_{t+\Delta t}(a'_{i-1}, b'_{j-1}) \\ \hat{w}_{t+\Delta t}(a'_{i-1}, b'_{j}) = \tilde{w}_{t+\Delta t}(a'_{i-1}, b'_{j}) \\ \hat{w}_{t+\Delta t}(a'_{i-1}, b'_{j+1}) = \tilde{w}_{t+\Delta t}(a'_{i-1}, b'_{j+1}) \\ \hat{w}_{t+\Delta t}(a'_{i}, b'_{j-1}) = \tilde{w}_{t+\Delta t}(a'_{i}, b'_{j-1}) \\ \hat{w}_{t+\Delta t}(a'i, b'_{j}) = \tilde{w}_{t+\Delta t}(a'_{i}, b'_{j}) \\ \hat{w}_{t+\Delta t}(a'_{i}, b'_{j+1}) = \tilde{w}_{t+\Delta t}(a'_{i}, b'_{j+1}) \\ \hat{w}_{t+\Delta t}(a'_{i+1}, b'_{j-1}) = \tilde{w}_{t+\Delta t}(a'_{i+1}, b'_{j-1}) \\ \hat{w}_{t+\Delta t}(a'_{i+1}, b'_{j}) = \tilde{w}_{t+\Delta t}(a'_{i+1}, b'_{j}) \\ \hat{w}_{t+\Delta t}(a'_{i+1}, b'_{j+1}) = \tilde{w}_{t+\Delta t}(a'_{i+1}, b'_{j+1}) \end{cases}$$

La valeur à un point étant :

$$\hat{w}(a'_i, b'j) = max(v^e(a'_i, b'_j), e^{-r\Delta t}(\sum_i \sum_j \alpha^{00}_{ij} T^{10}_{ij} T^{20}_{ij} + \alpha^{10}_{ij} T^{11}_{ij} T^{20}_{ij} + \alpha^{01}_{ij} T^{10}_{ij} T^{21}_{ij} +$$
$$\alpha^{11}_{ij} T^{11}_{ij} T^{21}_{ij} + \alpha^{20}_{ij} T^{11}_{ij} T^{20}_{ij} + \alpha^{21}_{ij} T^{12}_{ij} T^{21}_{ij} + \alpha^{12}_{ij} T^{11}_{ij} T^{22}_{ij} + \alpha^{02}_{ij} T^{10}_{ij} T^{22}_{ij} + \alpha^{22}_{ij} T^{12}_{ij} T^{22}_{ij}))$$

Bibliographie

Références

[1] Andersen, L. and Broadie, M. (2004), Primal-dual simulation algorithm for pricing multidimensional american options, *Management Science*, 50(9) :1224–1234.

[2] Bally, V. and Pages, G. (2003), A quantization algorithm for solving multidimensional discrete-time optimal stopping problems, *Bernouilli*, 9(6) :1003–1049.

[3] Bally, V. and Pages, G. (2005), A quantization tree method for pricing and hedging multidimensional american options, *Mathematical Finance*, 15(1) :119–168.

[4] Barraquand, J. and Martineau, D. (1995), Numerical valuation of high dimensional multivariate american securities, *Journal of Financial and Quantitative Analysis*, 30(03) :383–405.

[5] Ben-Abdellatif, B.-A. H., M. and Rémillard, B. (2019), Dynamic programming and parallel computing for valuing two-dimensional american options, forthcoming in Journal of Systems Science and Complexity.

[6] Berridge, S. and Shumacher, J. (2008), An irregular grid approach for pricing high-dimensional american options, *Journal of Computational and Applied Mathematics*, 222(1) :94–111.

[7] Black, F. and Scholes, M. (1973), The pricing of options and corporate liabilities, *The Journal of Political Economy*, 81(3) :637–654.

[8] Boyle, P. (1988), A lattice framework for option pricing with two state variables, *Journal of Financial and Quantitative Analysis*, 23(1) :1–12.

[9] Broadie, G. P., M. and Ha, Z. (2000), *Probabilistic Constrained Optimization : Methodology and Applications*, chapter Pricing American options by simulation using a stochastic mesh with optimized weights, pages 32–50, Kluwer Academic Publishers.

[10] Broadie, M. and Glasserman, P. (1997), Pricing american-style securities using simulation, *Journal of Economic Dynamics and Control*, 21(8) :1323–1352.

[11] Broadie, M. and Glasserman, P. (2004), A stochastic mesh method for pricing high-dimensional american options, *Journal of Computational Finance*, 7(4) :35–72.

[12] Carriere, J. (1996), Valuation of the early-exercise price for options using simulations and nonparametric regression, *Insurance : Mathematics and Economics*, 19(1) :19–30.

[13] Cox, R. S., J. and Rubinstein, M. (1979), Option pricing : A simplified approach, *Journal of Financial Economics*, 7(3) :229–263.

[14] Detemple, F. S., J. and Tian, W. (2003), The valuation of american call options on the minimum of two dividend-paying assets, *Annals of applied probability*, 13(3) :953–983.

[15] Ekedahl, H. E., E. and Lehto, E. (2007), Dimension reduction for the black-scholes equation, department of Information Technology, Uppsala University.

[16] Hager, G. and Wellein, G. (2010), *Introduction to high performance computing for scientists and engineers*, Boca Raton : CRC Press.

[17] Hartley, P. (2000), Pricing a multi-asset american option, *Working paper. University of Bath.*

[18] Haugh, M. and Kogan, L. (2004), Pricing american options : a duality approach, *Operations Research*, 52(2) :258–270.

[19] Jin, L. X.-T. H., X. and Wu, Z. (2013), A computationally efficient state-space partitioning approach to pricing high-dimensional american options via dimension reduction, *European Journal of Operational Research*, 231(2) :362–370.

[20] Jin, T. H., X. and Sun, J. (2007), A state-space partitioning method for pricing high-dimensional american-style options, *Mathematical Finance*, 17(3) :399–426.

[21] Joliffe, I. (1986), *Principal Component Analysis*, Springer-Verlag.

[22] Kamrad, B. and Ritchken, P. (1991), Multinomial approximating models for options with k state variables, *Management Science*, 37(12) :1640–1652.

[23] Kovalov, L. V., P. and Marcozzi, M. (2007), Pricing multi-asset american options : A finite element method-of-lines with smooth penalty, *Journal of Scientific Computing*, 33(3) :209–237.

[24] Longstaff, F. and Schwartz, E. (2001), Valuing american options by simulation : a simple least-squares approach, *The Review of Financial Studies*, 14(1) :113–147.

[25] Margrabe, W. (1978), The value of an option to exchange one asset for another, *The Journal of Finance*, 33 :177–186.

[26] Raymar, S. and M., Z. (1997), A monte carlo valuation of american call options on the maximum of several stocks, *The Journal of Derivatives*, 5(1) :7–23.

[27] Reisinger, C. and Wissmann, R. (2015), Numerical valuation of derivatives in high-dimensional settings via partial differential equation expansions, *Journal of Computational Finance*, 18(4) :95–127.

[28] Reisinger, C. and Wittum, G. (2007), Efficient hierarchical approximation of high-dimensional option pricing problems, *SIAM Journal of Scientic Computing*, 29 :440–458.

[29] Rogers, L. (2002), Monte carlo valuation of american options, *Mathematical Finance*, 12(3) :271–286.

[30] Schober, S. P., P. and Wittum, G. (2015), Efficient parallel solution methods for high-dimensional option pricing problems, available at SSRN : https ://ssrn.com/abstract=2591254 or http ://dx.doi.org/10.2139/ssrn.2591254.

[31] Stultz, R. (1982), Options on the minimum or the maximum of two risky assets :analysis and applications, *Journal of Financial Economics*, 10(2) :161–185.

[32] Tilley, J. (1993), Valuing american options in a path simulation model, *Transactions of the Society of Actuaries*, 45 :83–104.