Pricing a Multi-Asset American Put Option by a Finite Element Method Approach

Kai Chen
Rutgers University

Abstract

This paper reports the work of final project for Computational Finance. In this project, a multi-asset American put option pricing framework is derived based on finite element methods. The value function of an American put option is known to satisfy a partial differential variational inequality. The problem is then reduced to a linear complementarity problem (LCP) in each time step, which is solved by the projected successive overrelaxation (PSOR) method. The convergence of this approach is analyzed in detail. The numerical experiment on maximum of three assets is presented and carefully studied. All the work is conducted in C++.

Keywords: Multi-asset American put option; finite element method; linear complementarity problem; projected successive overrelaxation method, convergence analysis.

Final project for Computational Finance (MATH 623)

1. Introduction

In the Black-Scholes-Merton framework, the American option pricing problem has no closed form solution and a numerical procedure has to be employed. A useful and classical technique for numerical option pricing is *binomial tree* models, which was first constructed by Cox, Ross and Rubinstein in 1979 [1]. The main drawback of a binomial model is that it is quite hard to adapt to more complex situations [2]. Same problems arise in even more general Monte Carlo simulation frameworks [3].

Different from simulation, the finite difference method is devoted to solving the underlying differential equation by converting it to a difference equation, which has been widely used in practice since it was first applied to option pricing by Eduardo Schwartz in 1977 [4]. In addition, a more mathematically complex method, finite element method, has been applied to numerous engineering problems, such as heat transfer, fluid flow, mass transport, and electromagnetic potential. Although the number of literature in financial application is very limited, it is shown that in most cases, finite element method performs better than finite difference method [5]. Therefore, we conduct our project on finite element application.

This paper is organized as follows. The mathematical formulation of multi-asset American put option is derived in Section 2. The variational formulation of the original problem is introduced in Section 3. The finite element approximation technique is illustrated in Section 4. Then to solve the problem numerically, the projected successive overrelaxation method is explained in Section 5. The experiment results together with convergence and error analysis are given in Section 6.

2. Multi-Asset American Put Option

An American option gives its holder the right to buy a specific underlying at a given price anytime before a given expiration time. We consider the underlying as a vector of stocks $S(t) = (S_1(t), \ldots, S_n(t))$ that follows the risk-neutral price dynamics described by the n-dimensional stochastic differential equation (SDE):

$$dS_i(t) = (r - q_i)S_i(t)dt + \sigma_i S_i(t)dW_i(t), \quad S_i(0) = S_i, \quad i = 1, 2, \ldots, n \tag{1}$$

where $r \geq 0$ is the risk-free interest rate under the risk-neutral measure; $\sigma_i > 0$ are the volatilities for assets; q_i is the continuously compounded dividend rate of the ith stock; $S_i > 0$ is the initial asset price for ith stock.

Moreover, W_i are n correlated standard Brownian motions with the correlation matrix $\rho_i j$, such that:

$$dW_i(t)dW_j(t) = \rho_{ij}dt \tag{2}$$

A standard payoff of a put option on a basket (portfolio) of multi-assets is given by:

$$\left(K - \sum_{i=1}^{n} w_i S_i(t)\right)^+, \quad t \leq T \tag{3}$$

where K is the strike price and w_i are portfolio weights. Notice that if $w_i = \frac{1}{n}$, it will be an option on the arithmetic average of the portfolio. The majority of index and basket options are written in this style. Other common payoffs in practice can be found in [6].

Given a filtration $\mathcal{F}(t)$, then a *stopping time* τ is defined as a random variable taking values in $(0, \infty)$ and satisfying:

$$\{\tau \leq t\} \in \mathcal{F}(t), \quad \forall t \geq 0 \tag{4}$$

Denote $\mathcal{T}_{t,T}$ as the set of all stopping times for $S(t)$ with values in the interval (t, T). The value of an American option is then given by:

$$V(t,s) = \sup_{\tau \in \mathcal{T}_{t,T}} E[e^{-r(\tau - t)} g(S(\tau)) | S(t) = s] \tag{5}$$

where $g(t, x)$ is the payoff of the American option when $S(t) = x$ at time t.

Thanks to the early work done by Bensoussan and Lions [7] and Jaillet, Lamberton, and Lapeyre [8], it is proved that $V(t, s)$ is the solution to the variational inequality, which can be represented in the a weak form of a set of inequalities:

$$\begin{aligned} \frac{\partial V}{\partial t} + \mathcal{G}V - rV \leq 0, &\quad t \in [0,T), s \in \mathbb{R}^n \\ V(t,s) \geq g(s), &\quad t \in [0,T), s \in \mathbb{R}^n \\ (\frac{\partial V}{\partial t} + \mathcal{G}V - rV) \cdot (V - g) = 0, &\quad t \in [0,T), s \in \mathbb{R}^n \end{aligned} \tag{6}$$

with the terminal condition:

$$V(T,s) = g(T,s), \quad s \in \mathbb{R}^n \tag{7}$$

where $\mathcal{G}$ is a partial differential operator, called *infinitesimal generator*, for the underlying stochastic differential process (1), which is given by:

$$\mathcal{G}u := \frac{1}{2}\sum_{i,j=1}^{n}\sigma_i\sigma_j\rho_{ij}\frac{\partial^2 u}{\partial x_i \partial x_j} + \sum_{i=1}^{n}(r - q_i - \frac{1}{2}\sigma_i^2)\frac{\partial u}{\partial x_i} \tag{8}$$

for a multi-variable function $u = u(x_1, \cdots, x_n)$. This generator is a standard operator for Black-Scholes framework. A detailed derivation can be found in [9].

Then we set $v(t,x) = V(T-t, e^x)$, the system (6) (7) can be transformed into another inequality system:

$$\begin{aligned} \frac{\partial v}{\partial t} - \mathcal{G}v + rv \le 0, \quad & t \in [0,T), x \in \mathbb{R}^n \\ v(t,x) \ge g(e^x), \quad & t \in [0,T), x \in \mathbb{R}^n \\ (\frac{\partial v}{\partial t} - \mathcal{G}v + rv)\cdot(v-g) = 0, \quad & t \in [0,T), x \in \mathbb{R}^n \end{aligned} \tag{9}$$

with the initial condition:

$$v(0,x) = g(e^x) \tag{10}$$

The reason that we do this trick is to transform a terminal condition at time T into an initial condition at time zero, which makes it easier to set up the variational problems in the following sections. A detailed proof can be found in [7] and in [10] for a textbook treatment.

For each $t \in [0,T)$, there exists the so-called *optimal exercise price* $s^*(t) \in (0,K)$ such that for all $s \le s^*(t)$, the value of the American put option is the value of immediate exercise. The payoff of an American put option is shown in the following figure.

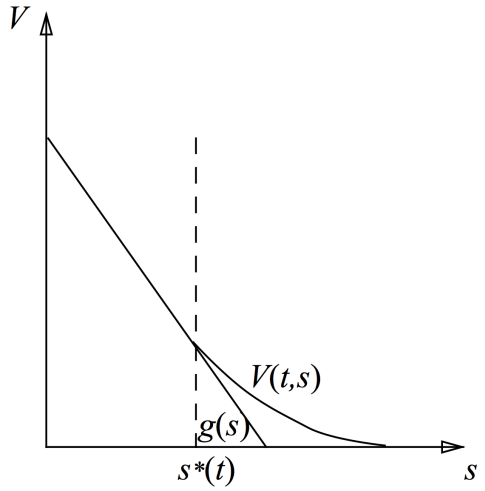

Fig. 1. American put option

As we can see from Figure 1, the region $\{(t,s)|s > s^*(t)\}$ is called the *continuation region* and the complement is called the *exercise region.* Since the optimal exercise price is not know as a priori, it is called a free boundary for the associated pricing partial differential equation and the problem of determining the option price is called a *free boundary problem* [11]. Notice that the inequality system (9) (10) does not involve the free boundary condition. However, it will be built in the system when we set up the variational form before applying the finite element methods.

3. Variational Formulation

In this section, we illustrate how to transform the system that is introduced in the previous part into a variational form and then we can apply finite element methods to. Notice that (9)(10) can be treated as a *parabolic obstacle problem* [12]. Thus, we shall work on the convex set $\mathcal{K} \in H^1(\mathbb{R}^n)$, such that:

$$\mathcal{K} := \{v \in H^1(\mathbb{R}) : v \geq g \text{ a.e. } x\} \tag{11}$$

where $H^1(\mathbb{R}^n)$ is the Hilbert space, which is also a special case of Sobolev space [13]. Then the variational form of (9)(10) is given by:

$$\begin{aligned} &\text{Find } u \in L^2([0,T]; H^1(\mathbb{R}^n)) \cap H^1([0,T]; L^2(\mathbb{R}^n)) \text{ such that } u(t,\cdot) \in \mathcal{K} \text{ and} \\ &(\frac{\partial u}{\partial t}, v-u) + a(u, v-u) \geq 0, \forall v \in \mathcal{K}, \text{ a.e. in } [o,T], \\ &u(0) = g \end{aligned} \tag{12}$$

where

$$(u,v) = \int_\Omega u(x)v(x)dx \tag{13}$$

is the inner product in $L^2(\Omega)$, and $a(\cdot,\cdot)$ is defined by:

$$a(u,v) = \frac{1}{2}\int_\Omega \sum_{i,j=1}^{n} \sigma_i\sigma_j\rho_{ij}\frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_i}dx - \int_\Omega \sum_{i=1}^{n}(r - q_i - \frac{1}{2}\sigma_i^2)\frac{\partial u}{\partial x_i}vdx + \int_\Omega ruvdx \tag{14}$$

Since the bilinear form $a(\cdot,\cdot)$ is continuous and satisfies a Gårding inequality: *there exists a nonnegative constant λ and a possitive constant C such that for all $v \in V$,*

$$a(v,v) \geq C|v|_V^2 - \lambda||v||_{L^2(\Omega)}^2 \tag{15}$$

the problem (12) then has a unique solution for every payoff (the proof can be found in [14]).

To simplify the problem a step further, we localize problem (12) to a bounded domain $G = (-R, R), R > 0$, by approximating the value v in Equation (5) by the value of a barrier option:

$$v_R(t,s) = \sup_{\tau \in \mathcal{T}_{t,T}} E[e^{-r(\tau-t)} g(e^{X_\tau}) 1_{\{\tau < \tau_G\}} | X_t = x] \tag{16}$$

Then we obtain the estimate for the localization error: there exist constants $C(T,\sigma), \gamma_1, \gamma_2 > 0$ such that (a detailed proof can be found in [9])

$$|v(t,x) - v_R(t,x)| \leq C(T,\sigma) e^{-\gamma_1 R + \gamma_2 |x|} \tag{17}$$

Using the same idea, we can localize the multi-dimensional space by a so-called parabolic cylinder, which is defined by:

$$Q_r(x_0,t) = \{(x,t) \in \mathbb{R}^n \times \mathbb{R} : |x - x_0| < r, -r^2 < |t - t_0| < r^2\} \tag{18}$$

Similarly, one can prove [15] that the localization error is bounded.

4. Finite Element Approximation

In this section, we develop the spatial discretization of the variational formulation (12) based on finite element methods. For textbook treatments of the basic theories, readers are referred to [16] and [17]. For the application to quantitative finance, one can read [18] and [19].

Consider the bounded domain after the localization: $\overline{\Omega} = [\underline{x}_1, \overline{x}_1] \times \cdots [\underline{x}_n, \overline{x}_n]$. Each of the n variables x_k is discretized as: $x_k^i = \underline{x}_k + i h_k, i = 0, \ldots, m_k + 1$, with $h_k = (\underline{x}_k - \overline{x}_k)/(m_k + 1)$. The next step is to define finite element basis functions only on the inner nodes, the total number of which is $M = m_1 \times \cdots \times m_n$. Define the basis functions $\{\varphi_J(\mathbf{x})\}_{J=1}^M$ associated with the inner nodes as the product of one-dimensional hat functions:

$$\begin{gathered} \varphi_J(\mathbf{x}) = \varphi_{i_1}(x_1)\varphi_{i_2}(x_2)\cdots\varphi_{i_n}(x_n) \\ J = J(i_i, i_2, \ldots, i_n), \quad 1 \leq i_1 \leq m_1, \ldots, 1 \leq i_n \leq m_n, \quad 1 \leq J \leq M \end{gathered} \tag{19}$$

where each one-dimensional hat function $\varphi_{i_k}(x_k)$ associated with the node x_k^i is defined as:

$$\varphi_i(x) = \begin{cases} (x - x_{i-1})/h, & x_{i-1} \le x \le x_i \\ (x_{i+1} - x)/h, & x_i \le x \le x_{i+1} \\ 0, & otherwise \end{cases} \tag{20}$$

Notice that the Jth basis function $\varphi_J(\mathbf{x})$ is a hat function equal to one at the node $\mathbf{x}_J = (x_1^{j_1}, \ldots, x_n^{j_n})$ and zero outside the n-dimensional cube $[x_1^{j_1-1}, x_1^{j_1+1}] \times \cdots \times [x_n^{j_n-1}, x_n^{j_n+1}]$. Then $u_N(t, \mathbf{x})$ can be represented by a linear combination of all the basis functions, such that:

$$u_N(t, \mathbf{x}) = \sum_J u_{N,J}(t)\varphi_j(\mathbf{x}) \tag{21}$$

Then we introduce a partition of the time interval $[0, T]$ into subintervals $[t_{n-1}, t_n], 1 \le n \le N$, with $k_i = t_i - t_{i-1}$. Supposing we set time intervals into equal sizes, such that $k_i = k$, we then have a fully discretized problem. Applying an implicit (backward) Euler scheme [20] together with the spatial discretization, problem (12) can be transformed into:

$$\begin{aligned} &\text{Find } u_N^{m+1} \text{ such that for } m = 0, \ldots, M-1 \\ &(v - u_N^{m+1})^*(\mathbf{M} + k\mathbf{A})u_N^{m+1} \ge (v - u_N^{m+1})^*(k\underline{f} + \mathbf{M}u_N^m) \\ &u_N^0 = 0 \end{aligned} \tag{22}$$

where the so-called *mass matrix*, *stiffness matrix* and the *load vector* with respect to the basis are defined as:

$$\mathbf{M}_{IJ} = (\varphi_J, \varphi_I), \quad \mathbf{A}_{IJ} = a(\varphi_J, \varphi_I), \quad \underline{f}_I = -a(g, \varphi_I) \tag{23}$$

Note that in the general multi-dimensional case, matrix $\mathbf{M}$ and $\mathbf{A}$ are Kronecker products of matrices corresponding to univariate problems. The Kronecker product of matrices $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{p \times q}$ is given by:

$$\mathbf{Z} := \mathbf{X} \otimes \mathbf{Y} = \begin{pmatrix} \mathbf{X}_{11}\mathbf{Y} & \mathbf{X}_{12}\mathbf{Y} & \cdots & \mathbf{X}_{1n}\mathbf{Y} \\ \mathbf{X}_{21}\mathbf{Y} & \mathbf{X}_{22}\mathbf{Y} & \cdots & \mathbf{X}_{2n}\mathbf{Y} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{X}_{m1}\mathbf{Y} & \mathbf{X}_{m2}\mathbf{Y} & \cdots & \mathbf{X}_{mn}\mathbf{Y} \end{pmatrix} \in \mathbb{R}^{mp \times nq} \tag{24}$$

The finite element approximation is an essential part of the solution to the problem.

Thus, we provide some important properties regarding to problem (22).

Theorem 4.1 (**Uniqueness**). *Consider λ such that Gårding inequality holds, and take $\Delta t < \frac{1}{\lambda}$; there exists a unique solution to problem* (22).

Proof. The arguments are mostly based on Stampacchia. First rewrite problem (22) in a more compact form:

$$\forall v \in \mathcal{K}, \quad (u^n - u^{n-1}, v - u^n) + \Delta t_n a(u^n, v - u^n) \geq 0 \tag{25}$$

For infinite-dimensional Hilbert spaces assume that the bilinear form $a(\cdot, \cdot)$ is continuous and satisfies Gårding inequality.

There exists unique $Q^n \in V_h$ such that

$$(Q^n, v)_V = (u^n - u^{n-1}, v) + \Delta t_n a(u^n, v) \quad v \in V_h \tag{26}$$

We then introduce a positive parameter ρ, which will be chosen later. Then problem (25) is equivalent to

$$\forall v \in \mathcal{K}, \quad (u^n, v - u^n)_V - \rho(Q^n, v - u^n) \leq (u^n, v - u^n)_V \tag{27}$$

which implies that u^n is the projection of $u^n - \rho Q^n$ on $\mathcal{K}$, for the scalar product $(\cdot, \cdot)_V$:

$$u^n = \Pi_{\mathcal{K}}(u^n - \rho Q^n) \tag{28}$$

Thus, we have proven problem (22) is equivalent to the *fixed point problem* (28). To use the Banach-Picard fixed point theorem [21], we nned to find ρ such that the operator $u \mapsto \Pi_{\mathcal{K}}(u - \rho Q)$ is a contraction in the norm $|| \cdot ||_V$, which can be easily proved by the continuity of the projector $\Pi_{\mathcal{K}}$.

Consider now λ such that Gårding inequality holds, and take $\Delta t < \frac{1}{\lambda}$. We have:

$$\begin{aligned} ||u - \rho||_V^2 &= ||u||_V^2 - 2\rho(Q, u)_V + \rho^2 ||Q||_V^2 \\ &= ||u||_V^2 - 2\rho((u, u) + \Delta t_n a(u, u)) + \rho^2 ||Q||_V^2 \\ &\leq (1 - 2\rho(1 - \lambda \Delta t_n))||u||_2^2 + (1 - C\rho)||u||_V^2 + \rho^2 C^2 ||u||_V^2 \end{aligned} \tag{29}$$

For $\Delta t < \frac{1}{\lambda}$ and ρ small enough, we have proved that the mapping $u \mapsto u - \rho Q$ is a contraction in V_h and we can apply the fixed point theorem: there exists a unique solution satisfying (28), and therefore (22). □

Theorem 4.2 (**Stability**). *Denote $u_{\Delta t}$ the piecewise affine function in time such that $u_{\Delta t}(t_n) = u^n$. For any $c < 1$, there exists a constant C such that for all Δt with $2\lambda\Delta t < c$,*

$$\sup_{0\le t\le T} ||u_{\Delta t}(t)||^2 + \int_0^T ||u_{\Delta t}||_V^2 dt \le C||u^0||_V^2 \tag{30}$$

Proof. Here we provide a sketch of the proof: according to problem (22) and Gårding inequality, assuming that $2\lambda\Delta t < 1$, we have:

$$(1 - 2\lambda\Delta t)||u^n - u^0||^2 + C_1\Delta t_n||u^n||_V^2 + \Delta t(2\lambda||u^0||^2 + C_2||u^0||_V^2) \tag{31}$$

For $m \le N$, multiplying this by $(1 - 2\lambda\Delta t)^{m-n-1}$ and summing over n from 1 to m:

$$\begin{aligned} &||u^m - u^0||^2 + C_1\sum_{n=1}^{m}(1 - 2\lambda\Delta t)^{m-n-1}\Delta t_n||u^n||_V^2 \\ \le&\Delta t\sum_{n=1}^{m}(1 - 2\lambda\Delta t)^{m-n-1}(2\lambda||u^0||^2 + C_2||u^0||_V^2) \\ =&\frac{1 - (1 - 2\lambda\Delta t)^{m-n-1}}{1 - 2\lambda\Delta t}(||u^0||^2 + C||u^0||_V^2) \end{aligned} \tag{32}$$

The above estimate concludes the proof. □

Theorem 4.3 (**Convergence**). *Assume that the coefficients σ and r are smooth enough so that*

$$\lim_{\Delta t\to 0}\sup_{n=1,\dots,N}\sup_{t\in[t_{n-1},t_n]}\sup_{v,w\in V}\frac{|(a_{t_n} - a)(v, w)|}{||v||_V||w||_V} = 0 \tag{33}$$

then

$$\lim_{h,\Delta t\to 0} ||u - u_{\Delta t}||_{L^2(0,T;V)} + ||u - u_{\Delta t}||_{L^\infty(0,T;L^2(\Omega))} \tag{34}$$

Here we do not provide a theoretical proof, however, the convergence can be shown through experiment results. Readers can find a detailed proof in [18].

In summary, we have shown that problem (22) has unique solution, which is stable and will converge when mesh size gets smaller.

5. Numerical Method

So far, we have already transformed the original obstacle problem (9) (10) into a finite element approximation problem (22). In this section, we present a numerical method to solve the problem. First notice that problem (22) is equivalent to a sequence of linear complementarity problems (LCP). Then at each time step, projected successive overrelaxation (PSOR) method can be implemented to solve the system.

5.1. Linear Complementarity Problem

In optimization theory field, the linear complementarity problem (LCP) arises frequently in computational mechanics and encompasses the well-known quadratic programming as a special case, which was proposed by Cottle and Dantzig in 1968 [22]. The general form of a LCP is given by:

$$\begin{aligned}
&\text{Given a vector } q \in \mathbb{R}^n \text{ and a matrix } M \in \mathbb{R}^{n\times n}\\
&\text{Find a vector } z \in \mathbb{R}^n \text{ such that}\\
&z \geq 0\\
&q + Mz \geq 0\\
&z^*(q + Mz) = 0
\end{aligned} \tag{35}$$

In problem (22), denote $\mathbf{B} := \mathbf{M}+k\mathbf{A}$, $F^m := k\underline{f}+\mathbf{M}u_N^m$. Then, the problem is equivalent to:

$$\begin{aligned}
&\text{Find a vector } u_N^{m+1} \in \mathbb{R}^n \text{ such that for } m = 0, \ldots, M-1\\
&\mathbf{B}u_N^{m+1} \geq F^m\\
&u_N^{m+1} \geq 0\\
&(u_N^{m+1})^*(\mathbf{B}u_N^{m+1} - F^m) = 0
\end{aligned} \tag{36}$$

The equivalence of problem (22) and problem (36) is proved in great detail in [9].

5.2. The Projected SOR Algorithm

Problem (36) can be solved by a so-called *projected overrelaxation* (PSOR) method which was proposed by Cryer [23]. PSOR is designed to solve the abstract form problem:

$$
\begin{aligned}
&\text{Find a vector } x \in \mathbb{R}^n \text{ such that} \\
&\mathbf{A}x \geq b \\
&x \geq c \\
&(x-c)^*(\mathbf{A}x - b) = 0
\end{aligned}
\tag{37}
$$

The algorithm is then listed below:

Choose an initial guess $x^0 \geq c$
Choose $w \in (0,1]$ and $\varepsilon > 0$
For $k = 0, 1, 2, \ldots$
For $i = 1, \ldots, N$
$\tilde{x}_i^{k+1} = \frac{1}{A_{ii}}(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{N} A_{ij} x_j^k)$
$x_i^{k+1} = \max\{c_i, x_i^k + w(\tilde{x}_i^{k+1} - x_i^k)\}$
Next i
If $\|\|x^{k+1} - x^k\|\|_2 < \varepsilon$ stop else
Next k

Table 1: PSOR algorithm

This construction is a modification of the so-called successive overrelaxation (SOR) method [24] used to solve iteratively systems of linear equations.

The PSOR method has a very good property which can be shown in the following theorem:

Theorem 5.1. *Assume* $\mathbf{A}$ *satisfies: (i) There exist constants* $C_1, C_2 > 0$ *such that* $C_1 v^* v \leq v^* \mathbf{A} v \leq C_2 v^* v, \forall v \in \mathbb{R}^N$*; (ii) It is diagonally dominant, i.e.* $|\mathbf{A}_{ii}| > \sum_{j \neq} |\mathbf{A}_{ij}|, \forall i$*. Also assume* $w \in (0,1]$*. Then the sequence* $\{x^k\}$ *generated by PSOR converges, as* $k \to \infty$*, to the unique solution* x *of problem* (37)*.*

This convergence theorem provides us a stable iterative algorithm. A detailed proof is given by [18].

6. Experiment Results

In this section, we present our numerical experiments. We study the pricing of American-style put options on three assets. The payoff considered is set as the commonly used arithmetic average of the portfolio:

$$g = \left(K - \frac{S_1(t) + S_2(t) + S_3(t)}{3} \right)^+, \quad t \leq T \tag{38}$$

where the asset processes are given in (1) (2).

However, when implementing the algorithm, we consider an American style put option on the geometric average of the three assets, which is proved [25] to be a good approximation to the original problem.

Introduce the geometric average process:

$$I_t := (\Pi_{i=1}^n S_i(t))^{\frac{1}{n}}, \quad t \geq 0 \tag{39}$$

Then the put payoff is transformed into:

$$(K - I_t)^+ \tag{40}$$

Through calculation, we also derive the stochastic differential equation for this geometric Brownian motion process:

$$dI_t = (r - q_I) I_t dt + \sigma_I I_t dB_t, \quad I_0 = (\Pi_{i=1}^n S_i(0))^{\frac{1}{n}} \tag{41}$$

where the geometric volatility and the effective dividend yield are given by:

$$\sigma_I^2 = \frac{1}{n^2} \sum_{i,j=1}^n \rho_{i,j} \sigma_i \sigma_j, \quad q_I = \frac{1}{n} \sum_{i=1}^n (q_i + \frac{1}{2}\sigma_i^2) - \frac{1}{2}\sigma_I^2 \tag{42}$$

And the combined Brownian motion is given by:

$$B_t = \frac{1}{n\sigma_I} \sum_{i=1}^n \sigma_i W_i(t) \tag{43}$$

Notice that B_t is a one-dimensional continuous martingale with quadratic variation t, and so it is a standard Brownian motion. Therefore, the problem of valuing options on multi-asset can be reduced to solving this one-dimensional option pricing problem.

6.1. *Option Price*

The result for option price is shown in the following figure:

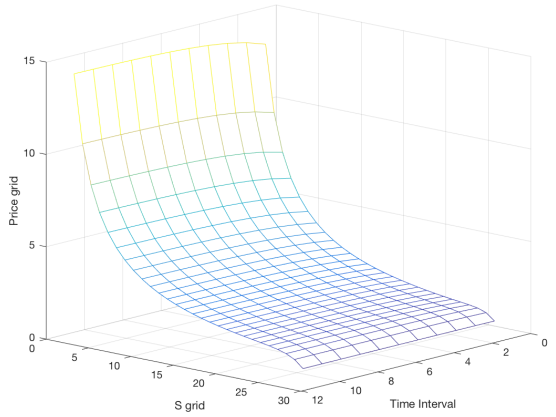

Fig. 2. American put price

In Figure 2, X-axis is the time interval; Y-axis is the stock grid and Z-axis is the option price grid. Note that here we do not provide the exact values, since they may vary with different input. Instead we use the grids to show the relationships among them.

To benchmark our result, we reproduce the result from [26], as is shown in the following figure:

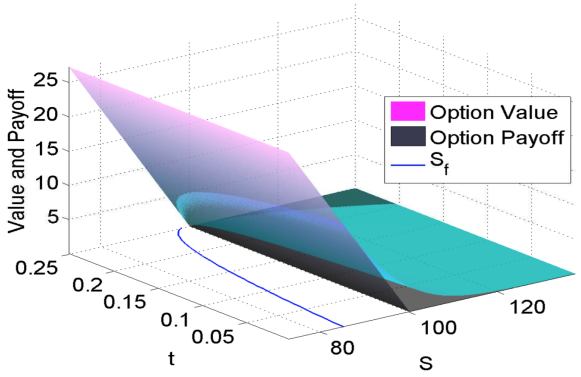

Fig. 3. American put price (benchmark)

Then we fix the time to maturity and check the value change of time. The result is shown in the following figure:

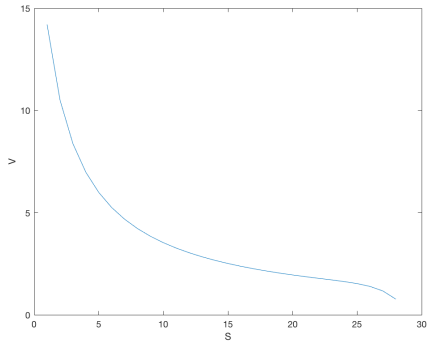

Fig. 4. Price with fixed maturity

We can compare this with Figure 1 to see we get reasonable results.

6.2. Exercise Boundary

According to the theory developed in the previous sections, the exercise boundary should look like the following sketch figure:

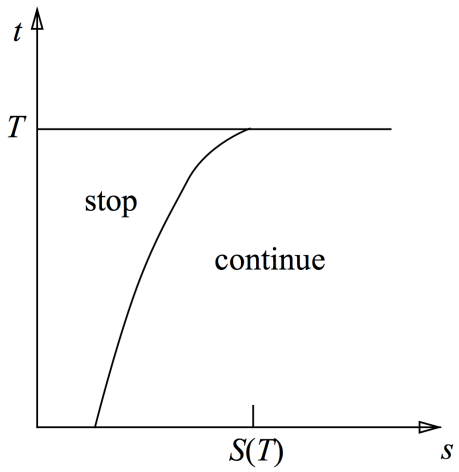

Fig. 5. Exercise boundary for put (theory)

The exercise boundary that we get is:

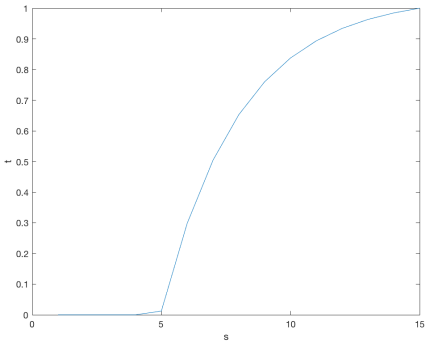

Fig. 6. Exercise boundary for put (this project)

We also find a benchmark from [19]:

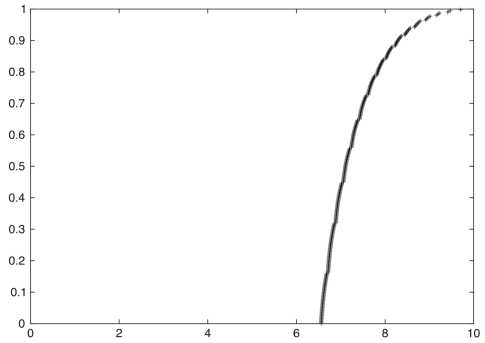

Fig. 7. Exercise boundary for put (benchmark)

Note that this benchmark is calculated by finite difference scheme without interpolation or smoothing. But it is enough to convey the idea and make comparison with our result.

Based on the results presented above, we can draw the conclusion [14] [27]: that the solution has a smooth graph over its domain and the optimal exercise boundary is smooth, despite the presence of many corners in the payoff function.

6.3. Convergence Analysis

We pick one point in Figure 2 and change the mesh size to do convergence analysis. The convergence curves in spatial and time direction are shown below:

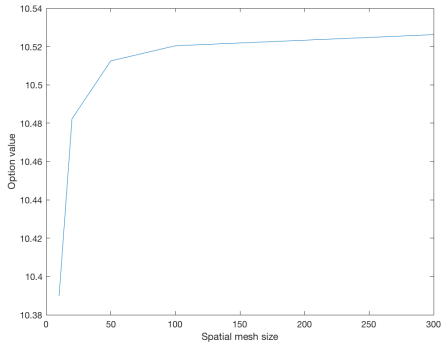

Fig. 8. Spatial convergence curve

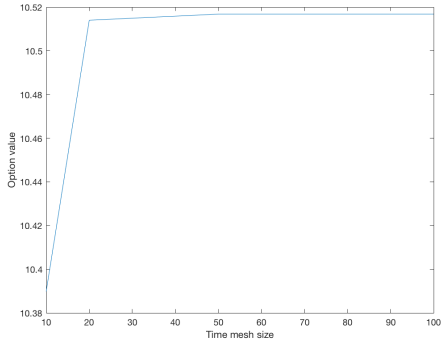

Fig. 9. Time convergence curve

As we can see from Figure 8 and Figure 9, the method converges pretty fast both in spatial and time direction.

6.4. Limiting Value Tests

Another useful technique for model validation is limiting value test. Here we simply apply this to strike and maturity and we observe the following cases:

- Set $K = 0$: we observe all the option values are zero or extremely small. The reason is that stock value cannot be negative, so a zero strike option should have no value.
- Set K very large: our upper bound of stock is programmed as 300. When we set $K = 10000$, we observe that at the mesh points where the stock values are small, the option's values are almost the same as the strike.
- Set u^0: we also test the sensitivity of input in the iteration program. When we use different input, the method converges every time even when we have negative input. This shows good compatibility of our method.

7. Conclusion

In this project, a finite element method approach is applied to multi-asset American put option pricing. We first give a detailed mathematical derivation of the problem together with error analysis for the approximation scheme. Then an example with maximum of three assets is carefully studied. The experiment results are compared with other literature to show the effectiveness of our method.

References

[1] John C Cox, Stephen A Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of financial Economics*, 7(3):229–263, 1979.

[2] Dietmar PJ Leisen and Matthias Reimer. Binomial models for option valuation-examining and improving convergence. *Applied Mathematical Finance*, 3(4):319–346, 1996.

[3] John C Hull and Sankarshan Basu. *Options, futures, and other derivatives*. Pearson Education India, 2016.

[4] Eduardo S Schwartz. The valuation of warrants: implementing a new approach. *Journal of Financial Economics*, 4(1):79–93, 1977.

[5] A Andalaft-Chacur, M Montaz Ali, and J González Salazar. Real options pricing by the finite element method. *Computers & Mathematics with Applications*, 61(9):2863–2873, 2011.

[6] Mark Broadie and Jérôme Detemple. The valuation of american options on multiple assets. *Mathematical Finance*, 7(3):241–286, 1997.

[7] Alain Bensoussan and J-L Lions. *Applications of variational inequalities in stochastic control*, volume 12. Elsevier, 2011.

[8] Patrick Jaillet, Damien Lamberton, and Bernard Lapeyre. Variational inequalities and the pricing of american options. *Acta Applicandae Mathematica*, 21(3):263–289, 1990.

[9] Norbert Hilber, Oleg Reichmann, Christoph Schwab, and Christoph Winter. *Computational methods for quantitative finance: Finite element methods for derivative pricing*. Springer Science & Business Media, 2013.

[10] Damien Lamberton and Bernard Lapeyre. *Introduction to stochastic calculus applied to finance*. Chapman and Hall/CRC, 2011.

[11] Avner Friedman. *Variational principles and free-boundary problems*. Wiley Online Library, 1982.

[12] Regis Monneau. A brief overview on the obstacle problem. In *European Congress of Mathematics*, pages 303–312. Springer, 2001.

[13] Robert A Adams and John JF Fournier. *Sobolev spaces*, volume 140. Academic press, 2003.

[14] Peter Laurence and Sandro Salsa. Regularity of the free boundary of an american option on several assets. *Communications on Pure and Applied Mathematics*, 62(7):969–994, 2009.

[15] Stephane Villeneuve. Exercise regions of american options on several assets. *Finance and Stochastics*, 3(3):295–322, 1999.

[16] Stig Larsson and Vidar Thomée. *Partial differential equations with numerical methods*, volume 45. Springer Science & Business Media, 2008.

[17] Vidar Thomée. *Galerkin finite element methods for parabolic problems*, volume 1054. Springer, 1984.

[18] Yves Achdou and Olivier Pironneau. *Computational methods for option pricing*, volume 30. Siam, 2005.

[19] Rüdiger Seydel and Rudiger Seydel. *Tools for computational finance*, volume 3. Springer, 2006.

[20] John C Butcher. Numerical methods for ordinary differential equations in the 20th century. In *Numerical analysis: Historical developments in the 20th century*, pages 449–477. Elsevier, 2001.

[21] Abdul Latif. Banach contraction principle and its generalizations. In *Topics in Fixed Point Theory*, pages 33–64. Springer, 2014.

[22] Richard W Cottle and George B Dantzig. Complementary pivot theory of mathematical programming. *Linear algebra and its applications*, 1(1):103–125, 1968.

[23] Colin W Cryer. The solution of a quadratic programming problem using systematic overrelaxation. *SIAM Journal on Control*, 9(3):385–392, 1971.

[24] David Young. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1):92–111, 1954.

[25] Pavlo Kovalov, Vadim Linetsky, and Michael Marcozzi. Pricing multi-asset american options: A finite element method-of-lines with smooth penalty. *Journal of Scientific Computing*, 33(3):209–237, 2007.

[26] Deniz Kaya. Pricing a multi-asset american option in a parallel environment by a finite element method approach, 2011.

[27] Kaj Nyström. Free boundary regularity for multi-dimensional american options through blow-ups and global solutions. *Journal of Computational Mathematics and Optimization*, 3:39–76, 2007.

Appendix

main_KaiChen.cpp

```
1  //============================================================
2  // Name        : main_KaiChen.cpp
3  // Author      : Kai Chen
4  // RUID        : 183001550
5  // Date        : 05/04/2018
6  // Description : Final Project for Computational Finance
7  //============================================================
8
9  #include <iostream>
10 #include <math.h>
11 #include <cmath>
12 #include <vector>
13 #include <algorithm>
14 #include "PSOR.h"
15
16 using namespace std;
17
18 double putPayoff(double spot, double strike);
19
20 int main()
21 {
22     int num = 30; // size for mesh size
23     vector<double> S_grid(num);
24     vector<double> t_grid(num);
25     vector<double> F(num);
26     vector<double> u(num,0.0); // set as u0 first
27     vector<vector<double>> A(num,vector<double>(num));
28     vector<vector<double>> M(num,vector<double>(num));
29     vector<vector<double>> B(num,vector<double>(num));
30     vector<double> Y, Z;
31
32     /* some variables that will be used in the program */
33     int i=0;
34     int j=0;
35     double dt = 0.1;
```

```
36      double T = 1; //maturity
37      double K = 100; // strike
38      double S_max = 300; // max for stock grid
39      double S_min = 0;   // min for stock grid
40      double r = 0.02;    // interest rate
41      double R = 300; // bound on spatial domain
42      double h = (S_max-S_min)/S_grid.size(); // dS
43
44      double sigma1 = 0.3;
45      double sigma2 = 0.3;
46      double sigma3 = 0.3;
47      double rho12 = 0.2;
48      double rho13 = 0.2;
49      double rho23 = 0.2;
50      double n = 3;
51      double sigma = sigma1*sigma1+2*rho12*sigma1*sigma2+2*rho13*↩
           sigma1*sigma3+sigma2*sigma2+2*rho23*sigma2*sigma3+sigma3*↩
           sigma3;
52
53      for(int i=0; i<S_grid.size(); i++)
54          S_grid[i] = (S_max-S_min)/S_grid.size() * (i+1);
55
56      /* build matrix A */
57      A[0][0] = 0.5*r*h;
58      A[0][1] = -(S_grid[0]*S_grid[0] * sigma*sigma) / (2*h) - 0.5*(↩
           r*S_grid[0]);
59
60      for(i=1; i<S_grid.size()-1; i++)
61      {
62          A[i][i] = 0.5*(S_grid[i]*S_grid[i] * sigma*sigma)*(1./h↩
               +1./h) + r/2*(h+h);
63          A[i][i-1] = -(S_grid[i]*S_grid[i] * sigma*sigma)/(2.*h) + ↩
               0.5*r*S_grid[i];
64          A[i][i+1] = -(S_grid[i]*S_grid[i] * sigma*sigma)/(2.*h) - ↩
               0.5*r*S_grid[i];
65      }
66
67      double last = S_grid.size()-1;
```

```
68      A[last][last] = 0.5*(S_grid[last]*S_grid[last] * sigma*sigma)↩
           *(1./h+1./h) + r/2*(h+h);
69      A[last][last-1] = -(S_grid[last]*S_grid[last] * sigma*sigma)↩
           /(2.*h) + 0.5*r*S_grid[last];
70
71      /* build matrix M */
72      double temp = h/6.0;
73      M[0][0] = 4*temp;
74      M[0][1] = temp;
75      for(i=1; i<S_grid.size()-1; i++)
76      {
77          M[i][i] = 4*temp;
78          M[i][i-1] = temp;
79          M[i][i+1] = temp;
80      }
81      M[last][last] = 4*temp;
82      M[last][last-1] = temp;
83
84      /* build matrix B */
85      for(i=0; i<S_grid.size(); i++)
86          for(j=0; j<S_grid.size(); j++)
87              B[i][j] = M[i][j] + dt*A[i][j];
88
89      /* build vector f */
90      vector<double> f(num,0.2);
91      f[0] = putPayoff(exp(-R),K)*(sigma*sigma/(2.*h*h) + (sigma*↩
           sigma    /2-r)/(2.*h));
92      f.back() = putPayoff(exp(R),K)*(sigma*sigma/(2.*h*h) - (sigma*↩
           sigma /2-r)/(2.*h));
93
94      /* build vector F */
95      for(i=0; i< F.size(); i++)
96      {
97          for(j=0; j<u.size(); j++)
98              F[i] = M[i][j]*u[j];
99          F[i] += dt*f[i];
100     }
101
```

```
102
103     /* Use Projected SOR to solve the LCP */
104     PSOR myLCP(A,F,u,0.5); // input lhs(A), rhs(F), initial guess(↩
            u), and the parameter for PSOR(w)
105     double t = 0;
106     while(t<T)
107     {
108         u = myLCP.solve(A,F,u,0.5);
109         //for(i=0; i<u.size(); i++)   // print out the result for ↩
                each iteration
110         //  cout << u[i] * S_grid[i]*100 << endl;
111         for(i=0; i< F.size(); i++)     // update F
112         {
113             for(j=0; j<u.size(); j++)
114                 F[i] = M[i][j]*u[j];
115             F[i] += dt*f[i];
116         }
117         t += dt;
118     }
119     return 0;
120 }
121
122 /* payoff of a put option */
123 double putPayoff(double spot, double strike)
124 {
125     double result = strike-spot > 0? strike-spot:0;
126     return result;
127 }
```

PSOR.h

```
1  using namespace std;
2  #include <vector>
3
4  #ifndef PSOR_H
5  #define PSOR_H
6
7  class PSOR
```

```
8  {
9  private:
10     double w; // control parameter for PSOR algorithm
11     vector<vector<double>> A; // lhs of system
12     vector<double> F; // rhs of system
13     vector<double> u; // solution to the system
14     double epsilon = 10e-5; // error tolerance for a single step
15 public:
16     PSOR(vector<vector<double>>, vector<double>, vector<double>, ↩
          double);
17
18     vector<double> solve(vector<vector<double>> A, vector<double> ↩
          F, vector<double> u, double w);
19     void print();
20
21 };
22
23 #endif
```

PSOR.cpp

```
1  // class functions for PSOR.h
2
3  #include "PSOR.h"
4  #include <iostream>
5  #include <math.h>
6  #include <cmath>
7  #include <vector>
8  using namespace std;
9
10 PSOR::PSOR(vector<vector<double>> A_, vector<double> F_, vector<↩
      double> u_, double w_)
11             :
12             A(A_), F(F_), u(u_), w(w_)
13 {
14 }
15
16 vector<double> PSOR::solve(vector<vector<double>> A, vector<double↩
```

```
    > F, vector<double> u, double w)
{
    double sumTemp = 0;
    //double w = 0.5; // parameter for PSOR
    vector<double> u_prev;
    u_prev = u;

    double error = 1;

    while(error > epsilon)
    {
        int i=0;
        int j=0;
        error = 0;
        u_prev = u;
        for(i=0; i<u.size(); i++)
        {
            sumTemp = 0;
            for(j=0; j<i-1; j++)
                sumTemp += A[i][j]*u[j];
            for(j=i+1; j<u.size(); j++)
                sumTemp += A[i][j]*u_prev[j];
            u[i] = 1/A[i][i] * (F[i]-sumTemp);
            u[i] = u_prev[i]+ w*(u[i]-u_prev[i]) > 0? u_prev[i]+ w↩
               *(u[i]-u_prev[i]):0;
        }

        for(i=0; i<u.size(); i++)
            error += (u[i] - u_prev[i])*(u[i] - u_prev[i]);
    }
    return u;
}

void PSOR::print()
{
    for(int i=0; i<u.size(); i++)
        cout << u[i] << endl;
}
```