ADVANCED MONTE CARLO METHODS FOR PRICING BERMUDAN OPTIONS AND THEIR APPLICATIONS IN REAL OPTIONS ANALYSIS

JIE ZHU

MACQUARIE
University

Principal Supervisor
Professor Pavel Shevchenko
Associate Supervisor
Professor Tak Kuen (Ken) Siu

Submitted to the Department of Applied Finance and Actuarial Studies
in fulfilment of the requirement for the degree of

Master of Research

at the
Faculty of Business and Economics
Macquarie University

October 2017

ABSTRACT

Pricing options with early exercise features is a challenging problem in mathematical finance. There is no general closed-form solutions available. We have implemented three established Monte Carlo methods in R for pricing Bermudan options: the random tree method, the stochastic mesh method and the least-squares Monte Carlo method (LSMC). We have also adopted the expectation-maximization (EM) control algorithm recently proposed in the literature and adapted this method for pricing Bermudan options. The numerical analyses find LSMC has the best performance; and have been extended to the improvements on LSMC via considering regression schemes such as piecewise linear regression and smoothing splines, random number generating process using low-discrepancy sequences and the usage of European options as control variates. The algorithm has also been applied to study a standard real option problem, the option to delay an investment project. The implemented algorithm is a powerful tool to solve many important application problems of decision under uncertainty in realistic settings that can be considered in further research.

Keywords: American option, real option, EM algorithm, Monte Carlo, least-squares Monte Carlo, option pricing, optimal stopping, dynamic programming, stochastic control

ACKNOWLEDGEMENTS

The completion of this thesis would not be possible without the support from many people. Among them, I must express my greatest appreciation to my supervisors Professor Pavel Shevchenko and Professor Ken Siu, whose guidance was invaluable throughout the development of this research project since late 2016. I truly cannot thank enough for their effort and time put into the reviewing and editing of this thesis. I also appreciate department advisor Shauna Ferris, who gave me many advices from the beginning of Master of Research program. I would also like to thank Professor Thomas Smith for his useful comments on my research frontier report. The administration staff at the Department of Applied Finance and Actuarial Studies has always been courteous and helpful. I would like to thank especially to Lin Bai and Agnieszka Baginska. Finally, I must thank my family for all their support, without them, I can hardly imagine this project could be accomplished.

Jie Zhu

CONTENTS

LIST OF FIGURES

LIST OF TABLES

---

1 INTRODUCTION

An option is a contract that gives the right but not an obligation to buy (call) or sell (put) the contract underlying asset at an agreed price at a predetermined future time. It is an essential financial instrument that facilitates the exchange of risks between two parties. Determination of a fair price of an option is a critical and challenging problem, especially when the market is incomplete where an option's price is not uniquely determined.

There are two basic types of options, namely, European options, where the option can only be exercised at the contract expiration date T; and American options, which can be exercised at any time before the expiry. In addition, there are many exotic options such as Barrier, Asian, Parisian and compound options, to name a few, but these will not be considered in this thesis. The classical assumption for pricing options is that the option underlying asset, S_t, follows a risk-neutral process over time $t, 0 \leq t \leq T$,

$$dS_t / S_t = (r - q)dt + \sigma dW_t, \tag{1.1}$$

where r is the annualised continuously compounded risk-free interest rate, q is the annualised continuous dividend yield, σ is the annualised volatility of S_t and $\{W_t\}_{0 \leq t \leq T}$ is a standard Brownian motion under a risk-neutral probability measure $\mathbb{Q}$. Then, the fair price of an option is calculated as risk neutral expectation under the process (1.1) of the discounted option payoff. For more general stochastic processes, the existence of $\mathbb{Q}$ is essentially equivalent to no arbitrage opportunity in the market; moreover, it is unique if the market is complete (Joshi, 2008). This thesis will focus on option pricing problems based on process (1.1).

The fundamental framework in Black and Scholes (1973) and its extension to dividend-paying stocks in Merton (1973) provide closed-form solution for the price of a European call options. However, for American options on a dividend-paying stock, we do not have knowledge about when is the optimal time to exercise the option in advance and a general closed-form solution is absent. In spite of this, there are a number of efficient numerical pricing techniques. The finite difference method applied to *Black-Scholes* partial differential equation (corresponding to the process (1.1)) is the most common approach (Brennan and Schwartz (1978)). To achieve an accurate pricing, a fine enumeration of possible stock values and time intervals is necessary. The second popular approach is the binomial tree due to Cox, Ross, and Rubinstein (1979), where we partition the time into pieces, generate binomial tree till maturity and find the optimal exercise frontier (*i.e.*, level of the underlying stock) backward in time through the tree starting from maturity.

Semi-analytic approximations for American option prices have also been well studied. Roll (1977) approximates American calls on dividend-paying stocks at the ex-dividend date. Johnson (1983) provides a semi-analytical approximation for pricing American puts without dividend. Geske and Johnson (1984) derives an approximation based on the compound option in Geske (1979), where the payoff of an American put is duplicated by an infinite series of risk-free bonds. Although Geske and Johnson (1984) is more efficient compared to the finite difference method, as the number of underlying assets grows, it becomes difficult to evaluate the required cumulative multivariate normal density functions. Carr (1998) estimates American option prices based on the randomization of the maturity date. The method provides accurate price for a single-asset American option by setting the maturity as a random variable following a gamma distribution.

Barone Adesi and Whaley (1987) proposes a quadratic approximation technique, which is proved to be efficient for pricing short dated American options (*i.e.*, the maturity is less than one year), but can provide poor estimates beyond one year.

It is also possible to approximate American option prices via quadrature methods (*e.g.*, Andricopoulos et al. (2003) and Sullivan (2000)). Anson, Thomas, and Luk (2012) shows that quadrature methods are 1000 times faster than using a multinomial tree for a given level of accuracy. In Luo and Shevchenko (2014), authors explored the usage of Gauss-Hermite quadrature for pricing exotic options. The quadrature achieves superior performance by overcoming the distribution error and non-linearity error in the binomial tree method and finite difference method. However, quadrature methods are limited to low-dimensional pricing problems, and require knowledge of the underlying asset transition density in closed-form.

For perpetual American options, several closed-form solutions for option prices are discussed in Mordecki (2002), Zhang and Guo (2004) and Gapeev and Rodosthenous (2014).

Nonetheless, there is more than one way in which asset dynamics can depart from the assumptions of *Black-Scholes*. In particular, stocks tend to have stochastic volatility and interest rate tends to be stochastic. These extensions will not be discussed in this thesis though our aim to find a flexible Monte Carlo method for pricing early exercise options includes flexibility in handling different underlying stochastic models.

In this thesis, we investigate Monte Carlo methods for pricing multi-asset options with early exercise features, where assets follow the risk-neutral process in equation (1.1). We compare different Monte Carlo methods for pricing options up to five assets. We test whether these methods represent a better alternative comparing to the binomial tree method.

For multi-asset options, especially when the number of underlying assets is more than three, it is necessary to use Monte Carlo methods (Kim et al. (2016)), for which the convergence rate is independent of the number of dimensions. The usage of Monte Carlo methods for option pricing can be dated back to Boyle (1977), where the underlying asset paths are simulated forward in time from valuation date till maturity to price a European option. However, the implementation in the context of options with early exercise features is not trivial. The main complication arises from the fact that the optimal exercise strategy is not known in advance. Bossaerts (1989) drafts the framework to use Monte Carlo methods for pricing American options. Nonetheless, not until Tilley (1993), the obstacle was resolved for research into estimating the optimal exercise strategy. Since then, there has been progress and a number of approximation techniques have been developed. The least-squares Monte Carlo method (LSMC) in Longstaff and Schwartz (2001) has been recognized to be efficient and easy to implement, but it is subject to limitations in two directions. First, the choice of regression scheme complexity in Longstaff and Schwartz (2001) is indecisive. Second, and perhaps more importantly, the estimator is biased and a large number of simulations is necessary to reduce the bias. For this reason, using Monte Carlo methods for pricing single-asset American options is still computationally expensive.

Alternative approaches have been developed to overcome these limitations such as the optimal stochastic convex switching system in Carmona, Hinz, and Yap (2007), however, it requires more computation time compared to Longstaff and Schwartz (2001). Recently, Kou, Peng, and Xu (2016) suggested the expectation-maximization control algorithm (EM-C). It converts a stochastic control problem into a parametric form and achieves an optimal solution via a procedure similar to the standard EM algorithm due to Dempster, Laird, and Rubin (1977). In this thesis, we adopt the algorithm for pricing Bermudan options. To our best knowledge, there is no study in the literature regarding using the EM-C algorithm for valuation of options with early exercise features.

When choosing a benchmark for comparing Monte Carlo methods, an important criterion was that we could compare the result with the binomial tree method. Hence, for high dimensional cases, the underlying asset has been assumed to be the geometric average of five stocks, so that the pricing problem can be reduced to a single dimension (Kemna and Vorst (1990)). Another consideration for implementation was the convergence of Monte Carlo methods. Rather than using American options, we chose Bermudan options, where the number of exercise opportunities is finite. Small number of early exercise opportunities allows us to achieve an accurate estimator within seconds using up to 20,000 simulated stock price paths.

Beyond the actual comparison, we are interested in the improvement on LSMC. To investigate this, we study three techniques. The first is a comparison of different regression schemes including defined polynomials, piecewise linear regression and smoothing splines. The second is based on different random number generating processes (*e.g.*, using low-discrepancy sequences). The last technique introduces European options as control variates to correct the simulation error.

From the simulation results, we conclude that LSMC is the most efficient Monte Carlo method for pricing Bermudan options in terms of computation time and accuracy. The use of control variates significantly improves the estimation accuracy. However, there is no optimal choice of regression scheme that provides superior results, nor the usage of low-discrepancy sequences consistently improves the convergence rate of the estimation.

Using LSMC, we extend the analysis to the option to delay a project (Copeland and Antikarov (2001)). Conventionally, an investment project with the option to delay is priced via the discounted net cash flow method (DCF) with the assistance of a binomial decision tree (Brandão, Dyer, and Hahn (2005)). However, the method is not viable in high dimensional problems. A more serious limitation of DCF is that the extra value from the uncertainty about the net cash flows from a project is not captured (Trigeorgis (1996)). The usage of LSMC for pricing projects with real options in the context of research and development (R&D) project is discussed in Biancardi and Villani (2014). Biancardi and Villani (2016) applies LSMC by transforming a capital budgeting problem into a compound American exchange option. Our study, on the other hand, uses LSMC to approximate future project values directly and decide whether delaying a project is worthwhile. We find the LSMC estimator is close to the analytical solution suggested by McDonald and Siegel (1986) in a single dimensional case. Further application to a two dimensional project is also investigated.

The rest of the thesis is organized as follows: the second chapter introduces option pricing theories and Monte Carlo methods for pricing options with early exercise features; Chapter 3 describes the algorithm and implementation details for pricing Bermudan options; Chapter 4 presents the simulation results; Chapter 5 presents and discusses the results for pricing the option to delay; and Chapter 6 concludes with final remarks and brief discussion of future research directions. Appendix A contains the detailed simulation results and Appendix B presents algorithms used in this thesis.

2 MONTE CARLO METHODS FOR OPTION PRICING

This chapter briefly reviews option pricing theories and Monte Carlo methods for pricing options with early exercise features. For a general introduction to mathematical finance see, among others, Wilmott, Dewynne, and Howison (1993), Joshi (2008) or Hull (2016).

2.1 BASIC CONCEPTS FOR PRICING OPTIONS

For an American option whose underlying asset is S_t, and strike is K, the payoff function at time $t, 0 \leq t \leq T$ can be written as $h(S_t) = \max(\varphi(S_t - K), 0)$, where $\varphi = 1$ for a call option and $\varphi = -1$ for a put option. For European options, this payoff can be paid at expiry date T only.

In their seminal work, Black and Scholes (1973) establishes the pricing framework for European options under the assumption that the contract underlying asset price, S_t, movements follow the stochastic differential equation,

$$dS_t / S_t = \mu dt + \sigma dW_t,$$

where μ is the expected instantaneous price change of the asset, σ is the volatility parameter and $\{W_t\}_{0 \leq t \leq T}$ is a Brownian motion. If the asset is a stock, the original framework assumes no dividend on the stock. Merton (1973) extends the framework to allow a constant continuous dividend payout q.

Let $V(S, t)$ be a price of the option at time t when the underlying asset value is S. Then a riskless portfolio Π can be constructed using an option $V(S, t)$ and the number of units, Δ, of the underlying asset S as

$$\Pi = V(S, t) - \Delta S.$$

Then, the price of this European option through time t is governed by the *Black-Scholes* partial differential equation,

$$\frac{\partial V}{\partial t} + (r-q)S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0, \tag{2.1}$$

with final condition, $V(S,T) = h(S)$. The closed-form solution of equation (2.1) gives the option price, $V(S,t)$, at any time t when underlying asset value is S. Equivalently, we can write the European option price as expectation of the discounted payoff at the expiry date,

$$V(S_0, 0) = \mathbb{E}_0^{\mathbb{Q}}[e^{-rT} h(S_T)], \tag{2.2}$$

where the expectation $\mathbb{E}_0^{\mathbb{Q}}[\cdot]$ is taken with respect to the asset price process under the unique risk-neutral probability measure $\mathbb{Q}$,

$$dS_t/S_t = (r-q)dt + \sigma dW_t, \tag{2.3}$$

and information at time $t = 0$.

For American options, the pricing requires the solution of the following form of variational equality (Wilmott, Dewynne, and Howison (1993)),

$$\min\left\{ rV - \frac{\partial V}{\partial t} - (r-q)S\frac{\partial V}{\partial S} - \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}, V - h(S) \right\} = 0.$$

This means, decisions have to be made at every point in time between exercising the option (if $V(S_t, t) < h(S_t)$) and holding it otherwise. Assuming the asset S_t follows the risk-neutral process in equation (2.3), the value, at time $t = 0$, of the American option can be computed as

$$V(S_0, 0) = \sup_{0 \le \tau \le T} \mathbb{E}_0^{\mathbb{Q}}[e^{-r\tau} h(S_\tau)], \tag{2.4}$$

where the maximum is taken over expectation of discounted payoff at every stopping time. It should be noted that a call on non-dividend-paying stock is always worth more than its intrinsic value $\max(S_t - K, 0)$ before the expiry date, and there is no extra value from early exercise opportunities.

2.2 WHY MONTE CARLO?

Monte Carlo methods are preferred in high dimensional problems or when the underlying process or payoff are complicated. Although constructing a multi dimensional binomial tree is possible (Boyle, Evnine, and Gibbs (1989)), it is inefficient. For example, if there are two underlying assets (S_1 and S_2), at each node of a two dimensional tree we will have four combinations of price movements:

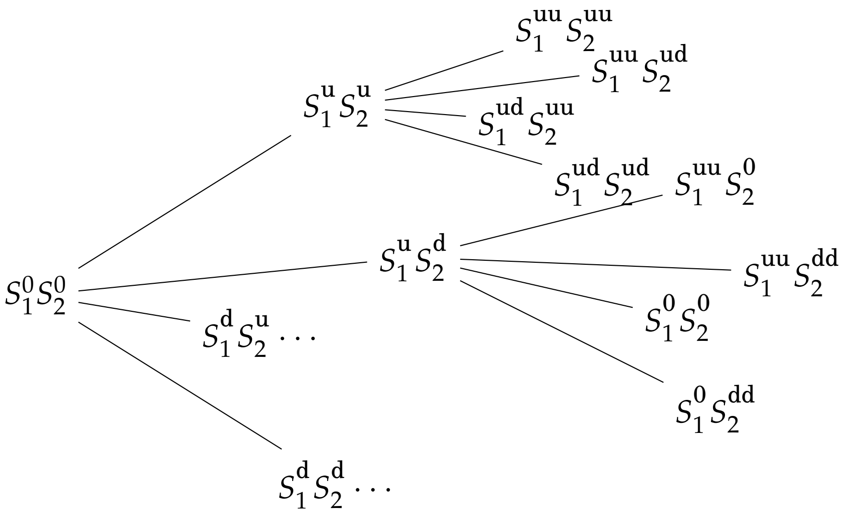


Figure 2.1: Illustration of a two dimensional tree

where S_i^{u} indicates an upward movement, S_i^{d} is a downward movement and S_i^0 is the spot price for the assets.

Figure 2.1 demonstrates that the number of nodes grows exponentially at the rate of $\mathcal{O}[2^{2dn}]$, where n is the number of time steps, $d = 2$ is the dimension of underlying assets. In contrast, for Monte Carlo methods, the computation effort, $\mathcal{O}[dM]$, is linearly dependent on the dimension and the number of simulation paths, M.

The second example demonstrates that Monte Carlo methods are more flexible compared to the binomial tree even in a two dimensional case. Illustrated by Joshi (2008), suppose a Bermudan option on an asset S which has a time-dependent volatility structure (σ_t), that is, the stock price volatility is stochastic. Consider a two-step standard binomial tree constructed for S_t, the asset price at time t, following a geometric Brownian model (Cox, Ross, and Rubinstein (1979)), and the parameters are adjusted so that there is equal probability of an up-move and a down-move in the first time step as well as the second time step.

Then, after an up-move in the first step and a down-move in the second step, we get

$$S^{\text{up}} = S_0 e^{(2r-(\sigma_1^2+\sigma_2^2)/2)\Delta t+(\sigma_1-\sigma_2)\sqrt{\Delta t}},$$

where S_0 is the initial stock price. When a down-move followed by an up-move, we get

$$S^{\text{down}} = S_0 e^{(2r-(\sigma_1^2+\sigma_2^2)/2)\Delta t+(\sigma_2-\sigma_1)\sqrt{\Delta t}}.$$

The two terms will be equal if and only if the volatility is not time-dependent (i.e., $\sigma_1 = \sigma_2$). If the asset S is determined by a single factor, we are able to adjust the time step to equal the two terms in order to recombine the two nodes, whereas for multiple assets, the recombining feature of the binomial tree is destroyed. The limitations in binomial tree methods lead us to consider the more flexible Monte Carlo methods.

2.3 PROBLEM FORMULATION

The application of Monte Carlo methods to European options is straightforward. We wish to find the expectation in equation (2.2) and know the stock prices follow the process in equation (2.3). Denote $S_{t,m}$ as simulated asset price at time t for the m^{th} simulated path. The procedure is to simulate dW_t by drawing random numbers from the standard normal distribution $N(0,1)$, plug these into equation (2.3), take $S_{T,m}$ for m^{th} draw, and then compute $e^{-rT}h(S_{T,m})$. We repeat this, keep the record of all $e^{-rT}h(S_{T,m})$, and the average of them converges to the desired expectation as the number of paths M increases without bias,

$$\mathbb{E}_0^{\mathbb{Q}}[e^{-rT}h(S_T)] \approx \frac{1}{M}\sum_{m=1}^{M} e^{-rT}h(S_{T,m}), \tag{2.5}$$

where $\hat{\mu} = \frac{1}{M}\sum_{m=1}^{M} e^{-rT}h(S_{T,m})$ is the Monte Carlo estimator. The order of error is $\mathcal{O}[M^{-1/2}]$, which means we need large number of simulations to achieve an accurate estimator for $\mu = \mathbb{E}_0^{\mathbb{Q}}[e^{-rT}h(S_T)]$. Under the central limit theorem, as $M \to \infty$, we can show that the $\hat{\mu}$ is distributed approximately from normal distribution as

$$\widehat{\mu} - \mu \sim N\left(0, \sqrt{\frac{s^2}{M}}\right),$$

where s^2 can be estimated as

$$\hat{s}^2 = \frac{1}{M-1} \sum_{m=1}^{M} (e^{-rT} h(S_{T,m}) - \widehat{\mu})^2.$$

We refer the quantity $\sqrt{\hat{s}^2/M}$ as the standard error of the Monte Carlo estimator (Glasserman (2013)). The error bound of the Monte Carlo simulations is random rather than deterministic (Gentle (2013)).

However, for American type contracts, the unknown optimal exercise time complicates the application of Monte Carlo methods. We have to formulate these problems as optimal stopping problems (Bäuerle and Rieder (2011), Chow and Robbins (1963), and Wald and Wolfowitz (1949)). These problems can be solved via dynamic programming techniques (Bellman (1952) and Bellman (1954)) or the martingale stopping theorem (Williams (1991)).

In the context of optimal stopping problems, we assume the option price

$$V(S_0, 0) < \infty.$$

Then, pricing options with early exercise features is equivalent to finding the optimal stopping time τ and the value function $V(S_0, 0)$ in

$$V(S_0, 0) = \sup_{0 \leq \tau \leq T} \mathbb{E}_0^{\mathbb{Q}} [e^{-r\tau} h(S_\tau)],$$

where notations are specified in (2.4).

Bellman (1954) derived an iterative procedure for finding the optimal stopping time τ called *value iteration* in a continuous time framework. Chow and Robbins, 1963 extends this framework to a discrete and finite horizon. Suppose $t_i, i = 0, 1, \ldots, N$, where $t_0 = 0 < t_1 \leq t_2 \leq \cdots \leq t_N = T$, the procedure is outlined as follows:

1. At $i = N$, set the option value function $V(S_{t_N}, t_N) = h(S_{t_N})$ for terminal state S_{t_N}.

2. For $i = N-1, N-2, \ldots, 0$, the value function is calculated as

$$V(S_{t_i}, t_i) = \max\left\{h(S_{t_i}), \mathbb{E}^{\mathbb{Q}}_{t_i}[e^{-r\Delta t}V(S_{t_{i+1}}, t_{i+1})]\right\}, \tag{2.6}$$

   where $\Delta t = t_{i+1} - t_i$, and expectation is taken with respect to information available at time t_i.

3. The solution for optimal stopping time is found by:

$$\tau := \min\{t_i | V(S_{t_i}, t_i) = h(S_{t_i})\}.$$

The challenge in this procedure is the estimation of the expected holding value $H(S_{t_i}, t_i) = \mathbb{E}^{\mathbb{Q}}_{t_i}[e^{-r\Delta t}V(S_{t_{i+1}}, t_{i+1})]$ in equation (2.6) before the expiry. In the next section, we review three major categories of Monte Carlo methods for approximating $H(S_{t_i}, t_i)$ under this backward dynamic programming framework.

2.4 EXISTING MONTE CARLO METHODS FOR PRICING AMERICAN OPTIONS

Following Stentoft (2013), we classify Monte Carlo methods for pricing American options into:

- stratified sampling methods;
- random tree and stochastic mesh methods; and
- regression based methods.

2.4.1 *Stratified Sampling Methods*

Tilley (1993)'s bundling algorithm is the first rigorous application of Monte Carlo method for pricing American options, the algorithm approximates the expected holding value $H(S_t, t)$ by grouping stock paths according to their values. It provides an estimator that is close to the value from the binomial tree method in a single dimension. However, Tilley (1993) does not address how the method can be extended to high dimensional problems. Barraquand

and Martineau (1995)'s stratified sampling method extends Tilley (1993)'s algorithm to multivariate American options. The paper shows results for options on up to 10 underlying assets. These early attempts successfully approximate the expected holding value, nonetheless, the resulting estimators are subject to bias, which is left untreated.

2.4.2 *Random Tree and Stochastic Mesh Methods*

Broadie and Glasserman (1997) argues, although Monte Carlo methods cannot produce an unbiased estimator for American option prices, boundaries can be computed for the estimator. It presents how to use a non-recombining random tree for pricing early exercise opportunities with an upper and a lower bound. Here, the expected holding value is calculated as the average of simulated paths at each node of the tree illustrated in Figure 2.2,

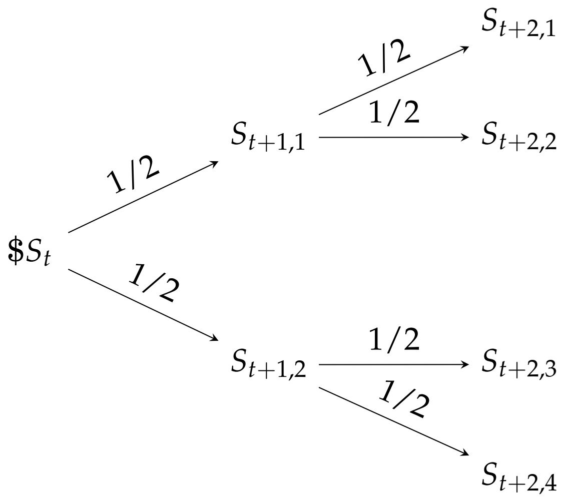


Figure 2.2: Illustration of a random tree

for example, the expected holding value at node $S_{t+1,1}$ are calculated as

$$H(S_{t+1,1}, t+1) \approx \frac{1}{2}\sum_{i=1}^{2} V(S_{t+2,i}, t+2)e^{-r\Delta t},$$

where Δt is the interval between t and $t+1$ times. This approach provides an upper estimator of the option value. The lower estimator, on the other hand, is computed by selecting part of the simulated paths to approximate the holding value. The interval between two boundaries is narrowing as we increasing the number of simulated paths at each node of a random tree.

A related method for calculating the upper bound is the duality method due to Rogers (2002), where a tighter bound can be achieved via correcting the foresight-bias (*i.e.*, knowing all possible paths over the simulation horizon).

The random tree method is more efficient than the non-recombining binomial tree in valuing Bermudan options, since it does not require partitioning the time into fine pieces in the case of simple processes where simulation over finite time interval can be done without discretization error. However, the computation effort still grows exponentially with the number of exercise opportunities.

Inspired by Rust (1997), Broadie and Glasserman (2004) introduce the stochastic mesh method. The algorithm is based on a simulated mesh as illustrated in Figure 2.3 to calculate the expected holding value:

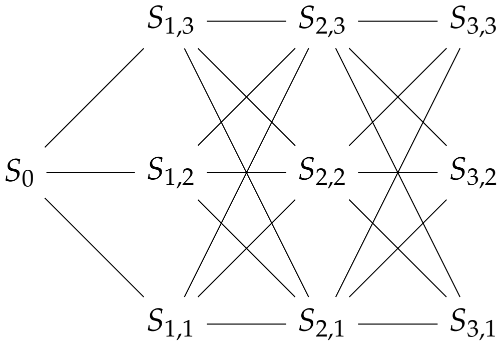


Figure 2.3: Illustration of a stochastic mesh

for example, at $S_{2,i}, i = 1, 2, 3$, the expected holding values are calculated as

$$H(S_{2,i}, 2) \approx \frac{1}{3} \sum_{j=1}^{3} V(S_{3,j}, 3) e^{-r\Delta t} W(S_{2,i}, S_{3,j}), \tag{2.7}$$

where $W(\cdot)$ is the *mesh weight* describing the transition behaviour from nodes $S_{2,i}$ to nodes $S_{3,j}$ (Broadie and Glasserman (2004)).

The key advantage of stochastic mesh method is the computation time grows linearly as the number of exercise opportunities increases. However, the efficiency of the mesh method is drew down by the effort to compute the weight function $W(\cdot)$ in equation (2.7).

Improvements of the stochastic mesh method can be found in Tan and Boyle (2000), where quasi Monte Carlo simulation is adopted to achieve a better convergence. In the review paper by Fu et al. (2001), a comparison has been made among stratified sampling algorithms, the stochastic mesh method and the random tree method. It concludes that the tree method is the most efficient for valuing American options in terms of computation time and accuracy.

2.4.3 *Regression Based Methods*

The idea of Tilley (1993) was well studied after its publication. The most important improvement is Carriere (1995), which uses a nonlinear spline regression to approximate the holding value. Later, Longstaff and Schwartz (2001) develops a new valuation technique for Bermudan options. This new technique uses a least-squares polynomial regression to estimate the expected holding value. About the same time, Tsitsiklis and Van Roy (2001) proposed a parametric approximation scheme that is similar to Longstaff and Schwartz (2001). The key differences between the two are the value function used for approximation and the paths selected for simulation.

In our numerical analysis, we will see that the least-squares Monte Carlo method (LSMC) provides a more accurate estimator than Tsitsiklis and Van Roy (2001). Stentoft (2014) provides a comprehensive review among Carriere (1995), Longstaff and Schwartz (2001), and Tsitsiklis and Van Roy (2001). The study concludes that LSMC has a smaller estimation error.

The performance of LSMC is affected by four factors: (1) the number of simulation paths, M; (2) the number of exercise opportunities, N; (3) the dimension of the problem; and (4) the regression scheme used in the LSMC (*i.e.*, polynomial regression, piecewise linear regression, smoothing splines, etc.). The convergence of LSMC is particularly dependent on the first two factors, we can see as $M \to \infty$ and $N \to \infty$, the LSMC estimator converges to the true American option price implied by the finite difference method or binomial tree method (if regression error can be ignored). In Egloff (2005), author proved the convergence of LSMC in high dimensions by generalizing the problem to a statistical learning problem. Regarding to point (3) and (4), Glasserman and Yu (2004) argues that, for high dimensional options, LSMC requires regression schemes with high-degree polynomials. However, increasing the degree of polynomial without increasing the number of simulation paths may degrade the estimation accuracy. To date, studies find it is unclear which polynomial will provide the most accurate Monte Carlo estimator in LSMC (Moreno and Navas (2003), Rasmussen (2005), Stentoft (2004), and Zhou (2004)).

Improvements on LSMC have also been investigated based on factors other than regression schemes. In terms of path simulation, variance reduction techniques are well studied and applied in practice. Boyle (1977) discusses the application of antithetic variables in Monte Carlo. Lai and Spanier (1998) applies low-discrepancy sequences to simulate stock paths and improves the convergence rate of LSMC [1]. Barraquand and Martineau (1995) applies quadratic re-sampling method. Other techniques such as common random numbers, conditioning and moment matching are discussed in detail in Glasserman (2013).

LSMC estimators are biased, Haugh and Kogan (2004) attempts constructing boundary for LSMC estimators similar to the random tree method: a lower bound can be obtained via applying the LSMC control policy to an independent set of assets paths; and an upper bound is computed via the duality method due to Rogers (2002). Broadie and Cao (2008) tightens the lower bound via a martingale control variate. In Rasmussen (2005), author introduced the European options as the control variates in LSMC procedure. The introduction of control variates shows a significant improved estimation accuracy compared to Longstaff and Schwartz (2001).

Although LSMC's computation time grows linearly with the number of exercise opportunities, the computation time is still an issue in high dimensional simulations. Dimensionality reduction is discussed in Dupire (1998), where we apply *Cholesky decomposition* to the covariance matrix of the simulated stock price movement processes. When a large number of exercise opportunities are required, one technique to reduce the computation time is based on the Brownian bridge in Stentoft (2004), where stock paths are generated from the end of valuation horizon. The usage of the Brownian bridge speeds up the convergence of LSMC (Caflisch and Chaudhary (2004)).

Of course, computation effort can be improved from a computer engineering perspective too. The utilisation of a graphic card in parallel computing is one approach, Abbas-Turki and Lapeyre (2009) confirms superior performance of multi-core GPU over CPU in Monte Carlo simulations. Pagès, Pironneau, and Sall (2016) proposes a two-level parareal

---

[1] The usage of low-discrepancy sequences is a kind of quasi-Monte Carlo methods

algorithm for LSMC simulation. In this algorithm, it allocates blocks of Monte Carlo paths to different processors.

In spite of these efforts, two problems of LSMC cannot be circumvented: 1) the appropriate choice of regression complexity; and 2) the large number of simulation paths required for high dimensional problems.

2.4.4 *Stochastic Convex Switching System*

To overcome the above mentioned limitations, the optimal stochastic convex switching system method was proposed in Carmona, Hinz, and Yap (2007) and its application to optimal stopping problems in Hinz (2014). Suppose in a convex switching system, a Bermudan option pricing problem over discrete times $t_i, i = 0, 1, \dots, N$ is formulated as:

$$V(S_{t_0}, t_0) = \sup_{\tau \in \{t_0, t_1, \dots, t_N\}} \mathbb{E}^{\mathbb{Q}}_{t_0}[e^{-r\tau} h(S_\tau)],$$

where S_{t_i} is the stock price following some random process; and K is the strike.

The corresponding system is defined by two positions $P = \{1, 2\}$ and two actions $A = \{1, 2\}$; where $p = 1$ means the option has been exercised and $p = 2$ means the exercise right is still available; and $a = 1$ denotes the action - exercise and $a = 2$ is the action - hold. Then, a position change matrix can be created by:

$$\{\alpha(p,a)\}_{p,a=1}^{p,a=2} := \begin{bmatrix} \alpha(1,1) & \alpha(2,1) \\ \alpha(1,2) & \alpha(2,2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

The reward function at t_i is:

$$R(S_{t_i}, p, a) = e^{-rt_i} \max(S_{t_i} - K, 0)(p - \alpha(p,a))$$

for all $p \in P$, and as the option can be only exercised once, so $a = 1$. By embedding the stochastic process S_{t_i} into appropriate vector space, the optimization is addressed in terms of value iteration and requires the approximation of the expected holding value:

$$H(S_{t_i}, t_i) = \mathbb{E}^{\mathbb{Q}}_{t_i}(\max(S_{t_{i+1}} - K, 0)(p - \alpha(p,a)))$$

at time t. Hinz (2014) assumes the value function to be a convex function and approximate it by a maximum over a finite set of affine linear functions. In terms of pricing American options, to achieve a given level of accuracy, the computation effort is in par with the trinomial random tree method.

2.4.5 *Expectation-Maximization Control Algorithm*

Kou, Peng, and Xu (2016) proves that a stochastic control problem can be expressed in a parametric form, of which parameters can be estimated via a modified expectation-maximization algorithm (EM algorithm). For instance, consider an utility maximization problem in discrete time:

$$\sup_C \mathbb{E}_0\left[\sum_{t=0}^{T-1} u_{t+1}(S_{t+1}, S_t, C_t)\middle| C_0, \theta_1, \theta_2, \cdots, \theta_{T-1}\right]$$

where S_t is the underlying wealth, and $C_t \in \mathbb{R}$, the amount of consumption at time t, are control variables. $S_0 = 1$ and C_0 are given, but $C_t, t = 1, 2, \ldots, T-1$ are determined by control parameters $\theta_t, t = 1, 2, \ldots, T-1$. At time T, the controller will consume all the wealth, $C_T = S_T$. We assume the utility function to be separable logarithmic utility, such that the problem can be defined as

$$\sup_C \mathbb{E}_0\left[\sum_{t=0}^{T-1} \log C_t + \log S_T\right], \tag{2.8}$$

where the underlying wealth, S_t, is assumed to follow a stochastic process,

$$S_{t+1} = (S_t - \frac{S_t}{1 + e^{C_t}})e^{a+bz_{t+1}}, t = 0, 1, \ldots, T,$$

where z_t is a random process. The aim is to maximize the expected utilities in expression (2.8).

The expectation-maximization control (EM-C) algorithm uses the following iterative procedures to address the maximization problem:

1. At $i = 1$, initialize the control policy parameter $x^i := (C_0^i, \theta_1^i, \theta_2^i, \ldots, \theta_{T-1}^i)$.

2. For i^{th} iteration, $i = 2, 3, \ldots$, start from θ_{T-1}, update x^i such that

$$\begin{aligned}&\mathbb{E}_0\left[\sum_{t=0}^{T-1} u_{t+1}(S_{t+1}, S_t, C_t)\middle| C_0^{i-1}, \theta_1^{i-1}, \theta_2^{i-1}, \cdots, \theta_{T-1}^{i}\right]\\ &\geq \mathbb{E}_0\left[\sum_{t=0}^{T-1} u_{t+1}(S_{t+1}, S_t, C_t)\middle| C_0^{i-1}, \theta_1^{i-1}, \theta_2^{i-1}, \cdots, \theta_{T-1}^{i-1}\right]\end{aligned}$$

3. Update the control parameters, $\theta_{T-2}^{i-1}, \ldots, \theta_2^{i-1}, \theta_1^{i-1}$ in the same fashion.

4. At period 0, update C_0^{i-1} such that

$$\begin{aligned}&\mathbb{E}_0\left[\sum_{t=0}^{T-1} u_{t+1}(S_{t+1}, S_t, C_t)\middle| C_0^{i}, \theta_1^{i}, \theta_2^{i}, \cdots, \theta_{T-1}^{i}\right]\\ &\geq \mathbb{E}_0\left[\sum_{t=0}^{T-1} u_{t+1}(S_{t+1}, S_t, C_t)\middle| C_0^{i-1}, \theta_1^{i}, \theta_2^{i}, \cdots, \theta_{T-1}^{i}\right]\end{aligned}$$

5. Proceed to the $(i+1)^{\text{th}}$ iteration or terminate the iteration if the difference between the expectation at i^{th} iteration and $(i-1)^{\text{th}}$ iteration is smaller than a predefined threshold value.

The major benefit of EM-C is that it does not depend on the iteration of value function and there is no need to approximate the holding value. We adopted EM-C algorithm to price Bermudan options. Suppose a Bermudan call has strike K, on an asset S_t following risk-neutral process in equation (1.1), and can be exercised at N future times, t_i, $i = 0, 1, \ldots, N$. The option value using EM algorithm, $V(S_{t_0}, t_0)$, at time t_0 with optimal control policy C^* can be found as

$$V(S_{t_0}, t_0) = \max_C \mathbb{E}_{t_0}\left[\sum_{i=1}^{N-1}\left(C_{t_i} h(S_{t_i}) \prod_{j=0}^{i-1}(1 - C_{t_j})\right) + h(S_{t_N}) \prod_{i=0}^{N-1}(1 - C_{t_i})\right], \tag{2.9}$$

where C_{t_i} is defined as,

$$C_{t_i} = \begin{cases} 1, & \text{if exercise the option;} \\ 0, & \text{if hold the option.} \end{cases}$$

In our numerical analysis, the exercise boundary is assumed to be a set of discrete points $\theta_i, i = 0, 1, \ldots, N$:

$$C_{t_i} = \begin{cases} 1, & \text{if } S_{t_i} \geq \theta_i; \\ 0, & \text{if } S_{t_i} < \theta_i. \end{cases}$$

Using the EM-C algorithm, we are able to find the parameters $\theta(t_i)$ that maximize objective function (2.9).

2.5 SUMMARY

Compared to numerical approximations such as binomial tree method, Monte Carlo methods make pricing options with early exercise features possible in high dimensions. Nonetheless, the Monte Carlo estimators for American option prices are biased. To reduce bias, significant computation effort is required. Several variance reduction techniques have been developed to speed up the convergence of Monte Carlo methods and confirmed to be efficient. In the next chapter, we use least-squares Monte Carlo method to demonstrate the methodology for pricing Bermudan options using Monte Carlo methods.

3 PRICING BERMUDAN OPTIONS

With all the theories presented above, we are now ready to present the implementation of least-squares Monte Carlo method (LSMC) for pricing Bermudan options. Several ways to improve LSMC such as regression schemes, low discrepancy random number sequences and control variates are discussed.

3.1 PROBLEM FORMULATION

In our numerical analysis, we assume a Bermudan call with a strike K which can be exercised at N times: t_i, $i = 1, 2, \dots, N$, where $t_0 = 0 < t_1 \leq t_2 \leq \cdots \leq t_N = T$. Further, the stock price S_t at time t is assumed to be driven by the risk-neutral process,

$$dS_t / S_t = (r - q)dt + \sigma dW_t,$$

where r is the risk-free rate, q is the continuous dividend payout from S_t, and $\{W_t\}_{0 \leq t \leq T}$ is the Brownian motion defined under a risk-neutral probability measure $\mathbb{Q}$. The option payoff at t_i, if exercised, is $h(S_{t_i}) = \max(S_{t_i} - K, 0)$; and the spot price S_{t_0} is given. The option value, then, can be formulated as

$$V(S_{t_0}, t_0) = \sup_{\tau \in \{t_0, t_1, \dots, t_N\}} \mathbb{E}^{\mathbb{Q}}_{t_0}[e^{-r\tau} h(S_\tau)],$$

where τ is the optimal exercise time for the given Bermudan option.

We exercise a Bermudan option when the immediate payoff $h(S_{t_i})$ exceeds the expected holding value given the current stock price and compute the option value V at t_i,

$$V(S_{t_i}, t_i) = \max(h(S_{t_i}), e^{-r\Delta t} \mathbb{E}^{\mathbb{Q}}_{t_i}[V(S_{t_{i+1}}, t_{i+1})]),$$

where $\Delta t = t_{i+1} - t_i$ is the interval between two consecutive admissible exercise dates. This value function is computed backward in time from t_N to t_0 in order to obtain the option price $V(S_{t_0}, t_0)$ starting from the terminal condition: $V(S_{t_N}, t_N) = h(S_{t_N})$.

3.2 LEAST-SQUARES MONTE CARLO APPROXIMATION

LSMC uses a cross-sectional regression to approximate the expected holding value $H(S_{t_i}, t_i) = \mathbb{E}^{\mathbb{Q}}_{t_i}[V(S_{t_{i+1}}, t_{i+1})]$. The procedure is outlined as follows. Suppose, holding value can be represented as $H(S_{t_i}, t_i) = \sum_{l=1}^{L} \phi_l(S_{t_i})\beta_{l,t_i}$, where $\{\phi_l(\cdot)\}_{l=1}^{l=L}$ is a set of basis functions and $\beta_{l,t_i}, l = 1, 2, \ldots, L$ are coefficients. Then

- simulate M stock paths $S_{t_i,m}, m = 1, 2, \ldots, M$ over times t_i, $i = 0, 1, \ldots, N$. At time t_i, denote the in-the-money paths as $S'_{t_i,q}, q = 1, 2, \ldots, Q \leq M$, and $h(S'_{t_i,q}) > 0$.

- For the in-the-money paths, we use current stock prices, S'_{t_i}, as covariates; and discount corresponding option value at next time step $V(S'_{t_{i+1}}, t_{i+1})$, as the response of LSMC regression, Y_{t_i}.

- Formulate the LSMC regression as

  $$Y_{t_i} = \sum_{l=1}^{L} \phi_l(S_{t_i})\beta_{l,t_i} + \epsilon_{t_i}, \tag{3.1}$$

  where ϵ_{t_i} denotes zero mean random error in the ordinary least-squares estimation of coefficients β_{l,t_i}. We assume that residuals are homoscedastic (*i.e.*, have the same variance).

- Finally, the expected holding value for in-the-money-paths, $H(S'_{t_i,q}, t_i)$, is approximated by

  $$\widehat{H}(S'_{t_i,q}, t_i) = \sum_{l=0}^{L} \phi_l(S'_{t_i,q})\hat{\beta}_{l,t_i},$$

  where $\hat{\beta}_{l,t_i}$ are estimated via minimizing $\sum_{q=1}^{Q}(Y_{t_{i,q}} - \sum_{l=0}^{L} \phi_l(S'_{t_i,q})\hat{\beta}_{l,t_i})^2$ for all in-the-money paths. We summarize the LSMC algorithm as follows.

Standard LSMC algorithm

1. For $t_i = t_0, t_1, \ldots, t_N$, $i = 0, 1, \ldots, N$, simulate M stock paths over N time points: $S_{t_i,m}$, $m = 1, 2, \ldots, M$.

2. Generate a matrix of control parameters, $A_{t_i,m} = 0$, to store information about the optimal exercise time for each path; a matrix of option payoff, $B_{t_i,m} = 0$, and a matrix of discounting factors, $C_{t_i,m} = 0$, at every point in time for each path.

3. At $i = N$, we have:

$$
\begin{aligned}
A_{t_i,m} &= 1; \\
B_{t_i,m} &= h(S_{t_i,m}); \\
C_{t_i,m} &= e^{-rt_N};
\end{aligned}
$$

   for $m = 1, 2, \ldots, M$.

4. Apply backward induction:
   - At $i = N - 1$, denote in-the-money paths at time t_i as $S'_{t_i,q}, q = 1, ..., Q \leq M$ (*i.e.*, corresponding to paths with $h(S_{t_i,m}) \geq 0\ , m = 1, ..., M$).
   - Construct the regression (3.1) and estimate its coefficients.
   - The approximated expected holding value for path q at $i = N - 1$ is

$$\widehat{H}(S'_{t_i,q}, t_i) = \sum_{l=0}^{L} \phi_l(S'_{t_i,q}) \hat{\beta}_{l,t_i}$$

     for $q = 1, 2, \ldots, Q$.
   - The option payoff at $i = N - 1$ is determined by

$$
\begin{cases}
B_{t_i,q} = h(S'_{t_i,q}), \text{ if } h(S'_{t_i,q}) \geq \widehat{H}(S'_{t_{i+1},q}) \\
B_{t_i,q} = V(S'_{t_i,q}, t_i), \text{ otherwise}
\end{cases}
$$

     for $q = 1, 2, \ldots, Q$.

- Update the control parameters in A:
  If $S'_{t_i,q} - K \geq \widehat{H}(S'_{t_i,q})$, find the corresponding path, $S_{t_i,m}$, and set

  $$\begin{aligned} &A_{t_i,m} = 1, \\ &A_{t_{i+1},m} = 0, A_{t_{i+2},m} = 0, \dots, A_{t_N,m} = 0; \end{aligned}$$

  Otherwise, the value of $A_{t_i,m}$ is not changed from the one that was set, for $m = 1, 2, \dots, M$.
- Update the discounting factors in C by

  $$C_{t_i,m} = e^{-rt_i};$$

  for $m = 1, 2, \dots, M$.
- Repeat step 4 for $i = N-2, N-3, \dots, 1$.

5. The optimal exercise time for path m: $\tau_m, m = 1, \dots, M$ can be found in the matrix A by

   $$\tau_m = \min_{i \in \{0,1,\dots,N\}} (t_{i,m} | A_{t_i,m} = 1).$$

6. At $i = 0$, approximate the option price as

   $$\widehat{V}(S_{t_0}, t_0) = \max\left(\sum_{t=t_0}^{t_N} \sum_{m=1}^{M} [A \circ B \circ C] e^{-r\Delta t}, h(S_{t_0}) \right),$$

   where $A \circ B \circ C$ is the Hadamard product[1] of matrices and S_{t_0} is the spot price for stock S_{t_i}.

We face two problems in this algorithm: the selection of regression scheme in equation (3.1); and how to increase the speed of convergence. The computation effort for LSMC is $\mathcal{O}[M]$, which grows linearly with M, the number of paths simulated. However, Glasserman and Yu (2004) finds in high dimensional problems, the number of paths to be required will grow exponentially with the degree of polynomials. Hence, we want to limit the number of simulation paths and, in the meanwhile, achieve an accurate estimator.

---

[1] The operation takes two matrices of the same dimensions, and computes another matrix where each element i, j is the product of elements i, j of the original two matrices.

3.3 IMPROVEMENTS ON THE LEAST-SQUARES MONTE CARLO METHOD

This thesis approaches the above problems in three directions:

- The regression scheme;
- The generator of random numbers; and
- The selection of control variates.

3.3.1 *Regression Schemes*

The choice of the basis $\{\phi_l(\cdot)\}_{l=1}^{l=L}$ can be parametric, semi-parametric polynomials or non-parametric splines. In Longstaff and Schwartz (2001), authors selected weighted Laguerre polynomials. In addition to this, we implement Chebyshev, Hermite and Laguerre polynomials. These polynomials are generated by the following standard recurrence functions. Chebyshev polynomials:

$$\phi_1(x) = 1; \phi_2(x) = 2x; \phi_{n+1}(x) = 2x\phi_n(x) - \phi_{n-1}(x).$$

Hermite polynomials:

$$\phi_1(x) = 1; \phi_2(x) = 2x; \phi_{n+1}(x) = 2x\phi_n(x) - 2n\phi_{n-1}(x).$$

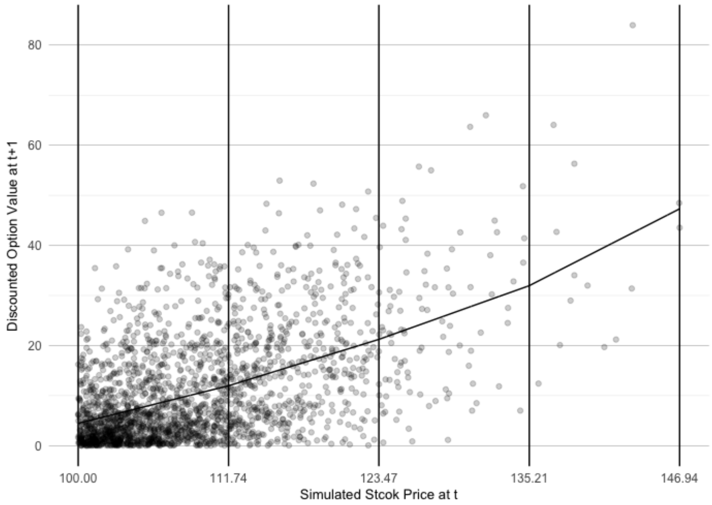
Laguerre polynomials:

$$\phi_1(x) = 1; \phi_2(x) = 1 - x; \phi_{n+1}(x) = \frac{1}{n+1}((2n+1-x)\phi_n(x) - n\phi_{n-1}(x)).$$

Legendre polynomials:

$$\phi_1(x) = 1; \phi_2(x) = 1 - x; \phi_{n+1}(x) = \frac{1}{n+1}((2n+1)\phi_n(x) - n\phi_{n-1}(x)).$$

Notice that the first two terms for these polynomials can only construct a simple linear regression. To compare polynomials of different degrees, our implementation compares the estimation results from using the first 3 to 12 terms in each defined polynomial.



The regression is fitted 4 months before maturity for a one-year Bermudan call option with strike price 100, which can be exercised at the end of every four months. The underlying stock has a spot price 100, annualised volitionality 20%. The risk-free rate is 5%. The number of in-the-money paths are 925 out of 2000 simulated paths.

Figure 3.1: Example of piecewise linear regression

More general nonlinear regression is investigated in two forms: smoothing splines (Green and Silverman (1993)), and piecewise linear regression. A piecewise linear regression simply breaks an ordinary linear regression into several segments, and fits separate simple linear regression within each segment. For example, in Figure 3.1, the linear regression are fitted separately in 4 intervals.

We anticipate these two nonlinear regression will provide better fittings for regression (3.1) than defined polynomials. The better fitting approximates the expected holding values and option prices more accurately.

3.3.2 *Random Number Generation*

The second approach targets the random number generation process in Monte Carlo methods. We implement variance reduction techniques to speed up the convergence of LSMC and reduce the number of simulations. Three techniques implemented are described as follows.

- *Antithetic variables.* Assume $z_i, i = 1, 2, \dots, M$ are independent random numbers from the standard normal distribution, and an estimator of μ is given by

  $$\hat{\mu} = \frac{1}{M} \sum_{i=1}^{M} f(z_i),$$

  for some function $f(\cdot)$. The method of antithetic variables is based on the observation that if z_i is from the standard normal distribution, then so does $-z_i$, and

  $$\tilde{\mu} = \frac{1}{M} \sum_{i=1}^{M} f(-z_i)$$

  is also an estimator of μ. Therefore,

  $$\hat{\mu}_A = \frac{\hat{\mu} + \tilde{\mu}}{2}$$

  is an estimator of μ as well. The $2M$ random numbers obtained from antithetic pairs $(-z_i, z_i)$ are more evenly distributed than a collection of $2M$ independent random numbers. The usage of antithetic variables improves Monte Carlo simulation efficiency if correlation between $\hat{\mu}$ and $\tilde{\mu}$ is negative.

- *Moment matching method.* The moments of random sample $z_i, i = 1, 2, \dots, M$ will not exactly match the moments of standard normal distribution. The Moment matching method attempts to transform z_i to match a finite number of moments of the standard normal distribution. In this thesis, we match the first and second moment of the standard normal distribution by defining

  $$\tilde{z}_i = (z_i - \bar{z}_i) \frac{\sigma_z}{s_z} + \mu_z,$$

  for $i = 1, 2, \dots, M$, where $\bar{z}_i$ is the sample mean of $z_1, \dots, z_M$, s_z is the sample standard deviation of $z_1, \dots, z_M$, $\sigma_z = 1$ is the standard deviation, and $\mu_z = 0$ is the mean of the standard normal distribution.

- *Low-discrepancy sequences.* If paths are generated using low-discrepancy sequences, we can get more evenly distributed asset paths with a smaller number of simulations (Ueberhuber (1997)). Niederreiter and Shiue (2012) demonstrates that the order of error of Monte Carlo simulations using these sequences improves from $\mathcal{O}[M^{-1/2}]$ to $\mathcal{O}[ln(M)^d/M]$, where d is the dimension of the simulation. Notice that the computation effort for low-discrepancy sequences will depend on the dimension. Therefore, for high dimensional options, direct applications of low-discrepancy sequences may require more computation time than a standard Monte Carlo. In spite of this issue, we still expect using these sequences will save computation time for a given level of accuracy.

3.3.3 *Control Variates*

The third approach is to use control variates to adjust the estimated option price $\widehat{V}(S_{t_0}, t_0)$. A good control variate X_0 usually has a high correlation with the estimated price. The adjusted estimator is

$$\widehat{V}_{adj}(S_{t_0}, t_0) = \widehat{V}(S_{t_0}, t_0) - b(\widehat{X}_0 - X_0), \tag{3.2}$$

where $\widehat{X}_0$ is the estimator of X_0 and the optimal b (computed via minimizing variance of $\widehat{V}_{adj}(S_{t_0}, t_0)$) is given by

$$b = \frac{\text{cov}(\widehat{V}_0, \widehat{X}_0)}{\text{var}(\widehat{V}_0)}.$$

For a Bermudan option, Rasmussen (2005) proposes the control variate to be the price of a plain vanilla European option whose parameters are the same as the Bermudan option to be valued (except the early exercise features).

For each iteration of LSMC algorithm, Rasmussen (2005) uses the following basis for LSMC regression:

$$\begin{aligned}
\phi_0(S_t) &= K \\
\phi_1(S_t) &= S_t \\
\phi_2(S_t) &= X_t \\
\phi_3(S_t) &= S_t X_t
\end{aligned}$$

where K is the strike of a Bermudan option, S_t are simulated stock prices at time t, and X_t are European option prices with spot price S_t and maturity $T - t$, where T is the the expiry for the Bermudan option, and all other parameters follow the Bermudan option. These basis are confirmed to have better performance than weighted Laguerre polynomials in Longstaff and Schwartz (2001) for pricing a Bermudan using control variates.

Suppose $\widehat{V}(S_{t_0}, t_0)$ is the LSMC estimator before adjustment from using Rasmussen (2005)s' basis, we adjust the LSMC estimator at the end of each iteration by equation (3.2), where $\widehat{X}_0$ is the European option price estimated from Monte Carlo method. The European has a maturity T and same parameters as the Bermudan option; and its exact price is computed using the *Black-Scholes* option pricing formula. We calculate b using the sample covariance and variance of the simulated European and Bermudan option prices using Monte Carlo iterations [2].

Implementing the standard LSMC algorithm is straightforward. However, the performance of LSMC depends on multiple factors, such as regression scheme and random number generating process. The improvements introduced in this chapter will be implemented in our numerical analysis. We are interested in whether these improvements will increase the estimation accuracy and reduce the computation effort of LSMC.

---

[2] Independent iterations of Monte Carlo simulations.

4 SIMULATION RESULTS AND ALGORITHM IMPROVEMENT

The main goal of this thesis is to attempt to find an efficient Monte Carlo method for pricing Bermudan options. We have reviewed three existing Monte Carlo methods: the random tree method, stochastic mesh method and least-squares Monte Carlo method (LSMC); as well as the the new expectation-maximization control (EM-C) algorithm. This chapter compares the estimation results from these methods for Bermudan calls on a single asset and the maximum of two assets.

Previous studies indicate a number of improvements can be done to LSMC. From simulation results, we want to find the most effective improvement. These comparisons are based on Bermudan calls on a single asset and the geometric average of five assets.

To effectively compare different algorithms, in addition to computation time, we select three proxies for measuring accuracy: error, root-mean-square error (RMSE) and relative standard error (Relative SE).

- Error, the difference between a Monte Carlo estimator and the benchmark value.
- RMSE, an aggregate measure of error, is given by:

  $$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\text{Error}_i)^2}$$

  where $N = 25$ is the number of independent repetitions (iterations) of Monte Carlo option price estimation. A smaller value of RMSE indicates a higher accuracy of the estimator. To reduce the bias of Monte Carlo estimators one has to simulate the large number of stock paths in each simulation rather than increasing the number of iterations.
- Relative SE, a percentage calculated via dividing the standard deviation of option price estimator over 25 repetitions by the estimated value. A smaller Relative SE indicates the estimator has proportionately less variation around its mean, therefore more precise.

4.1 SINGLE-ASSET BERMUDAN OPTIONS

We begin the analysis of Bermudan calls specified in Table 4.1 by fixing the spot price $S_0 = 100$, and calculating the benchmark value of option price (5.7299) via binomial tree method using 1,000 time steps. We use antithetic variables to generate random numbers to compare the four Monte Carlo methods.

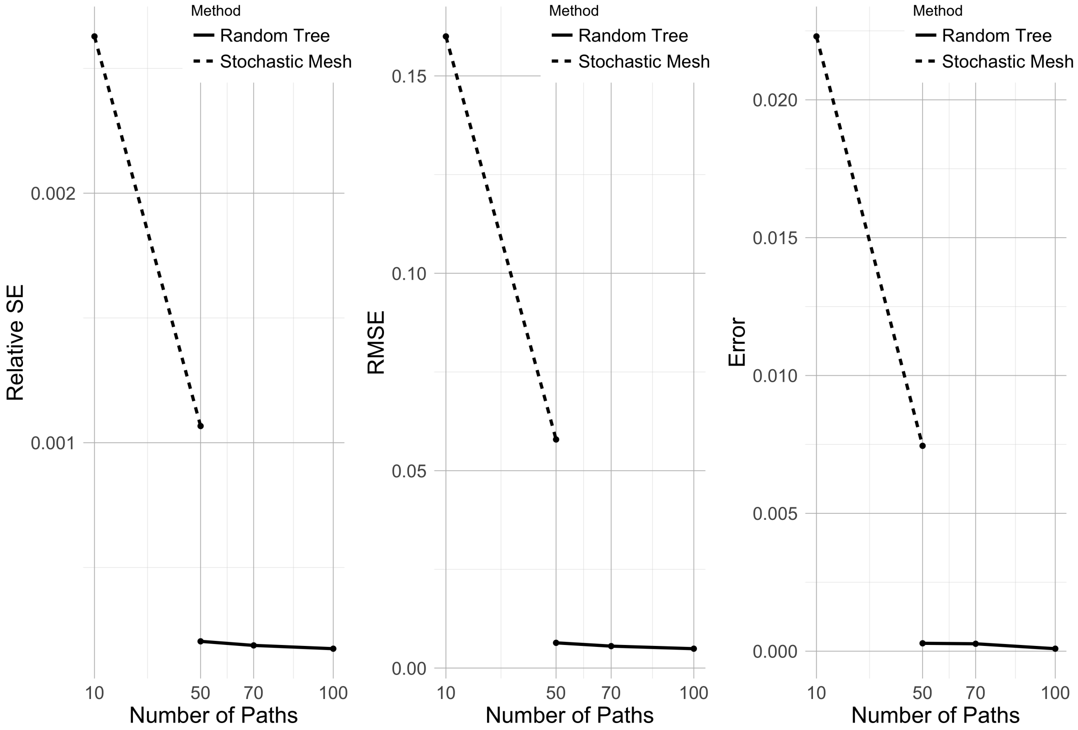
Table 4.1: Specifications for Bermudan call on a single stocks

Option Specifications	
Option Expiry Date (T)	End of 1 year
Strike (K)	100.00
Exercise Opportunity	End of every 4 months
Exercise Opportunities (N)	3
Annualized Risk Free Rate (r)	0.05
Stock Specifications	
Annualized Stock Volatility (σ)	0.2
Annualized Dividend Rate (q)	0.1

4.1.1 *The Random Tree and Stochastic Mesh*

Figure 4.1 contains the results from the random tree and stochastic mesh. Different number of branches are plotted against three assessment criteria. By restricting the computation time to 25 minutes, a much accurate result can be found in the random tree method with 100 branches compared to the stochastic mesh with 50 branches. Although increasing the number of branches improves the accuracy of both methods, the computation time prevents us from using more than 50 branches in the stochastic mesh method. The results find the computation of the weight function in the stochastic mesh method makes it less efficient in this single dimensional problem. This finding is consistent with Stentoft (2013).

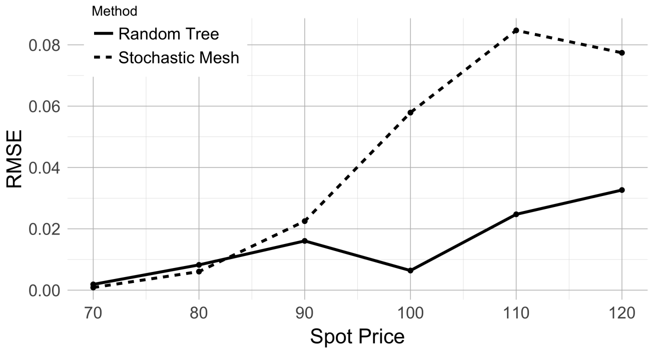
With respect to different spot prices (*i.e.*, 70,80,90,110 and 120) in Figure 4.2, RMSEs for both approaches are the highest when the option is deep-in-the-money (*i.e.*, the spot price is high). A possible explanation for this can be: as the option becomes in-the-money,



(a) Relative SE (b) RMSE (c) Error

Values are computed using R 3.3.1 with an i7-7820HQ CPU (same platform for all calculations in this paper). Computation time for one Monte Carlo Simulation: tree method (50 branches): 1.2 seconds; tree method (70 branches): 6 seconds; tree method (100 branches): 13 seconds; stochastic mesh (10 branches): 10 seconds; stochastic mesh (50 branches): 59 seconds. The detailed result can be found in Appendix A.

Figure 4.1: Comparison of random tree and stochastic mesh for a Bermudan call



Values are computed using 50 branches for both stochastic mesh and random tree.

Figure 4.2: Comparison of the random tree and stochastic mesh method for a Bermudan call with different spot prices

its delta [1] becomes close to one, which means the option will be more sensitive to the stock price movement. To accurately approximate the option value, it requires more simulations compared to an option out-of-the-money, which is less sensitive to the price movement.

In Figure 4.2, it can also be observed that he stochastic mesh method provides marginally better estimators when the spot prices are 70 and 80.

4.1.2 *LSMC and EM-C Algorithm*

Figure 4.3 reports the estimation results from the LSMC and EM-C. Both algorithms are not based on lattice, and are able to use large number of simulated paths within a short period of time. The computation time for these two methods was limited to 30 seconds. LSMC outperforms EM-C in all three criteria assessed. In Appendix A, we can see that the performance of LSMC is better than random tree. To match the benchmark at the third digit, LSMC only requires 20,000 simulated paths and 7.5 seconds. However, it takes 30 minutes for a random tree with 100 branches at each node.

Regarding to different spot prices, LSMC selects more paths to fit the regression as the option becomes more valuable, and requires more computation time (refer to the algorithm in Chapter 3). The computation time for EM-C, on the other hand, is constant across different spot prices but errors are significantly larger than in the case of LSMC. From the error patterns of EM-C, we can see it is not numerically stable and may not be suitable for the optimal stopping problem.

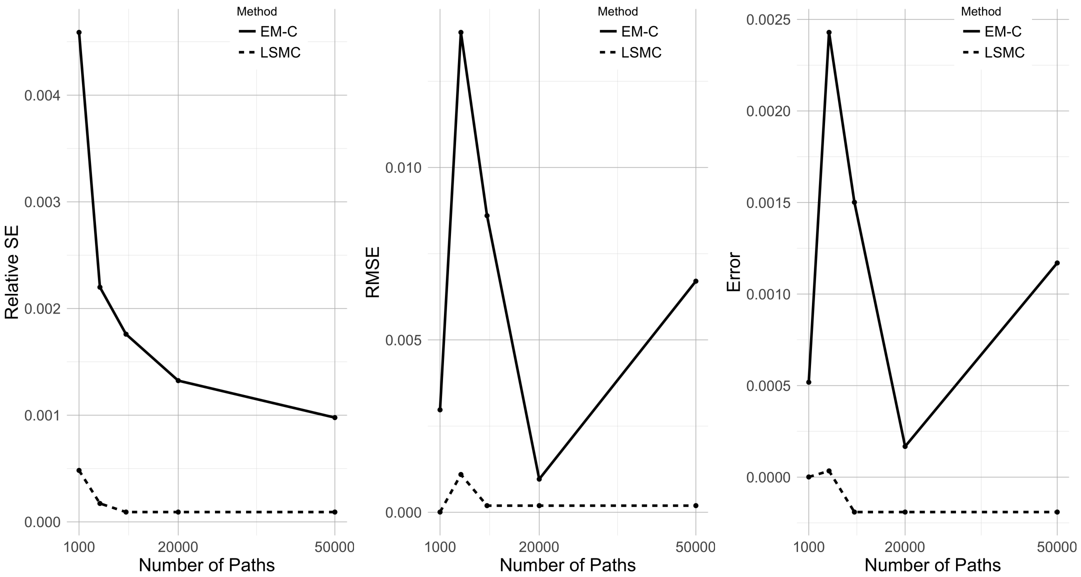
In Figure 4.4, we can see both methods produce precise results when the spot price is 120. This phenomenon could be interpreted by the fact that large number of simulation paths used in LSMC and EM-C help the estimation accuracy for options deep-in-the-money.

4.1.3 *The Accuracy and Convergence of LSMC in a Single Dimension*

We proceed with a discussion of possible performance enhancement to LSMC in a single dimension. The following results are based on the above Bermudan call specification.

---

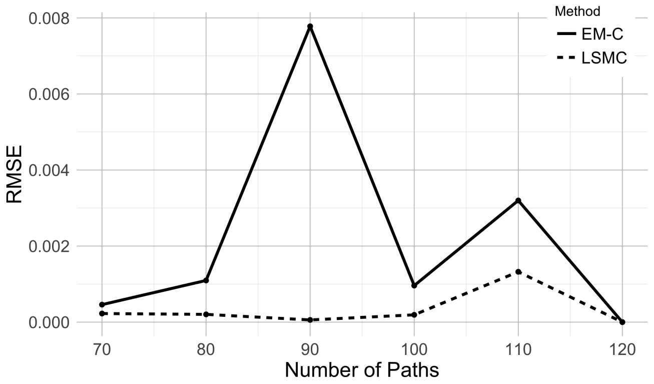
[1] Measures the impact of a change in the price of underlying asset to the option value, a larger delta indicates that the option price is highly sensitive to the underlying asset price movement.



(a) Relative SE (b) RMSE (c) Error

Values are computed using 20,000 paths for LSMC and EM-C. LSMC uses European options as the control variate. Appendix A presents the detailed result.

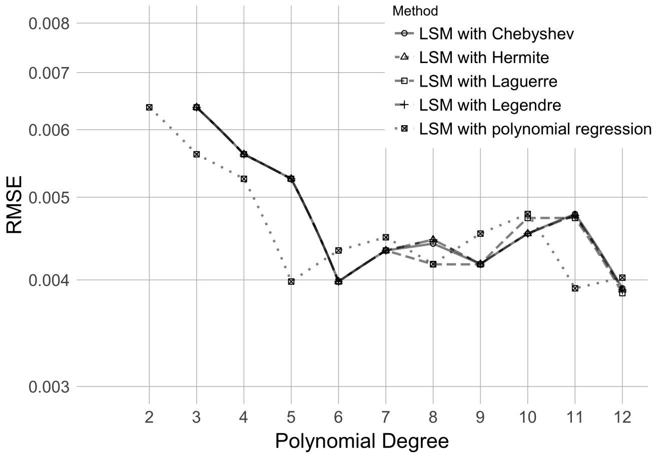
Figure 4.3: Comparison of LSMC and EM-C algorithm for a Bermudan call



Values are computed using 20,000 paths for both estimators.

Figure 4.4: Comparison of LSMC and EM-C algorithm for a Bermudan call with different spot prices

REGRESSION SCHEMES Figure 4.5 plots results from four defined polynomials: Chebyshev, Hermite, Laguerre and Legendre; as well as a simple polynomial: $\sum_{i=1}^{I} X^i$, where I is the highest polynomial degree, and X is the covariates used in the polynomial. The simple polynomial regression occurs to be the most accurate when the highest degree employed is less than or equal to five. The discrepancies in RMSEs are negligible for polynomials of which the highest degree are six or more. This finding indicates that increasing the polynomial degree without increasing the simulation paths does not help to achieve consistently higher accuracy.



Values are computed by replacing the regression component in LSMC by corresponding polynomials with 10,000 paths.

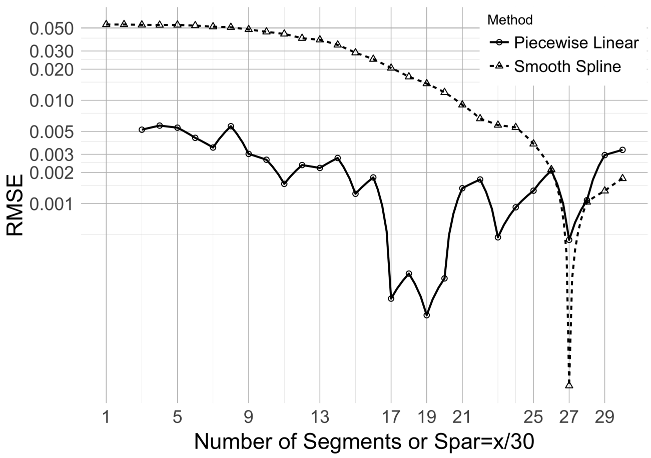
Figure 4.5: Accuracy of LSMC with different regression polynomials

Compared to defined polynomials, nonlinear regression in Figure 4.6 achieves more accurate price. In piecewise linear regression, the best estimator is found at using 19 equal-size segments, whereas in smoothing spline regression, the best occurs at using a smoothing factor spar $= 27/30$ [2]. Both RMSEs are below one tenth of a cent. By employing a smoother spline, the estimation accuracy increases monotonically before reaching the optimal point. However, fluctuations are observed in piecewise linear estimations. Smoothing spline achieves a better estimator via smoother fitting for the cross sectional regression as depicted in Figure 4.7.

These findings tell us the choice of regression scheme is dependent on the fit of LSMC regression. With different simulated paths, we have to try different regression schemes

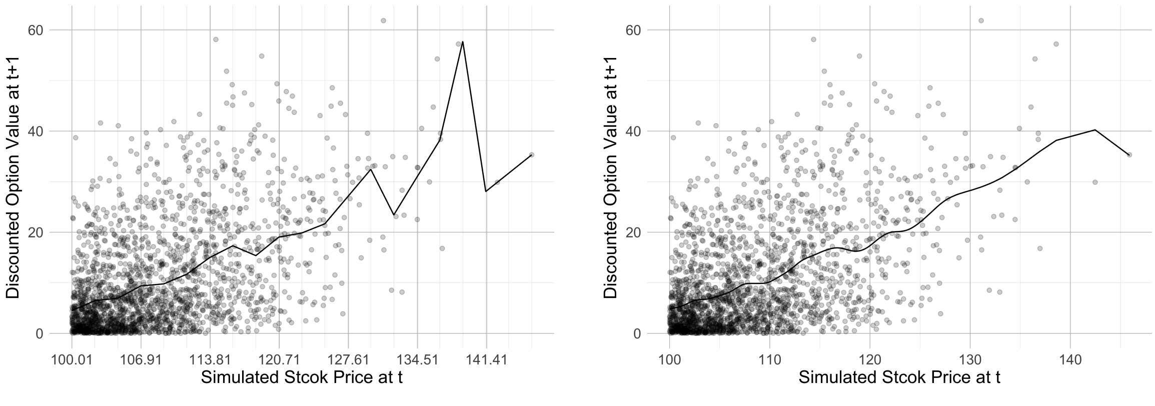
---

[2] *spar* factor in R function *smooth.spline*, indicating the level of smoothness of a spline. Its value is in $(0,1]$. The higher the value of *spar*, the smoother the curve will be.



Values are computed by replacing the regression component in LSMC by respective regression schemes with 10,000 paths. For piecewise linear regression, the x-axis is the number of segments employed; and $x < 3$ is not examined. For smoothing spline regression, it is the x as in spar $= x/30$. The estimation degrees of freedom is 10.

Figure 4.6: Accuracy of LSMC with nonlinear regression



(a) Piecewise linear regression

(b) Smoothing spline

Fittings are depicted using 2097 in-the-money stock paths one time step before the expiry for the option estimated in (4.6).

Figure 4.7: Fitting comparison between nonlinear regression

and examine how well the regression fits. If the fit is good, we can expect a the LSMC estimator will be accurate. Therefore, for further researches, it will be useful to develop algorithms to assess the fit of regression models based a set of simulated paths prior LSMC estimation.

PATH SIMULATION USING LOW-DISCREPANCY SEQUENCES In Figure 4.8, six random number generating processes are compared:

- Halton sequences [3] (with antithetic variables (AV));
- Sobol sequences (with antithetic variables);
- pseudo random number with antithetic variables; and
- pseudo random number matching the second moment of a standard normal distribution.

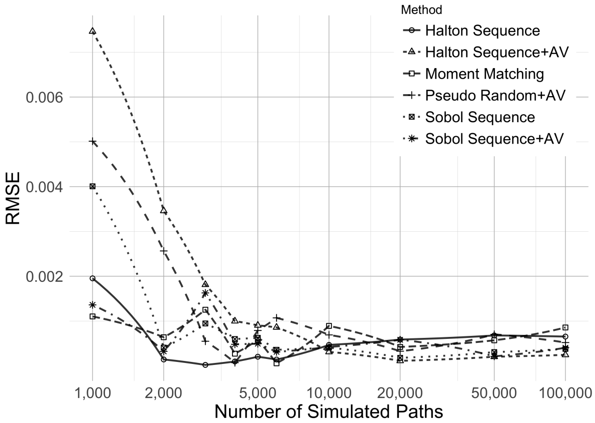
The usage of Halton sequences, Sobol sequences, Sobol sequences with antithetic variables and moment matching method achieve a more accurate result when 1,000 paths are used. However, the results are not converging until using 10,000 paths, which makes the usage of these low-discrepancy sequences the same as the pseudo random number generating process with antithetic variables. In this particular example, we have not seen the usage of low-discrepancy sequence significantly speeds up the convergence of LSMC as Rodrigues and Rocha Armada (2006) [4].

CONTROL VARIATES Figure 4.9 show significant improvement in the convergence rate of LSMC when using control variates. The estimation using control variates converges roughly at 3,000 paths compared to 20,000 paths without the control. Also can be observed is the standard error for LSMC with control variates is much smaller compared to the standard LSMC estimator.

---

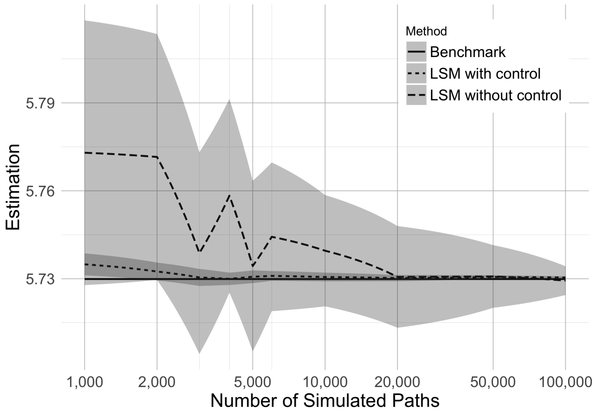
[3] Halton and Sobol sequences were generated in R using *randtoolbox* package. Then, the uniformly distributed sample numbers were transformed using Box and Cox (1964)'s method into a standard normal distribution.

[4] The paper prices a Bermudan call on 2 assets with 10 exercise opportunities using low-discrepancy sequence in this thesis with the assistance of Brownian bridge



Values are computed using LSMC with European options as the control variate as in Rasmussen (2005) and corresponding variance reduction techniques.

Figure 4.8: Convergence of LSMC with low-discrepancy sequences



Controlled values are computed using LSMC with European options as the control variate and regressions in Rasmussen (2005).
Values without control are computed using standard LSMC in Longstaff and Schwartz (2001).
The error bounds are depicted using the standard error of estimations with a 95% confidence interval.

Figure 4.9: Comparison of LSMC with and without control variates

4.1.4 *LSMC and Tsitsiklis and Van Roy (2001)*

In the approximation results presented above, the choices of the covariates and the response follow the standard LSMC in Longstaff and Schwartz (2001). Here, we compare Longstaff and Schwartz (2001) with Tsitsiklis and Van Roy (2001). The key differences between these two regression based methods are listed below:

- The calculation of option value:
    - LSMC:

$$V_t = \begin{cases} V_{t+1}e^{-r\Delta t}, \text{ if } S_t - K \leq \widehat{H}_t \\ S_t - K, \text{ if } S_t - K > \widehat{H}_t \end{cases}$$

    - Tsitsiklis and Van Roy (2001):

$$V_t = max(S_t - K, \widehat{H}_t)$$

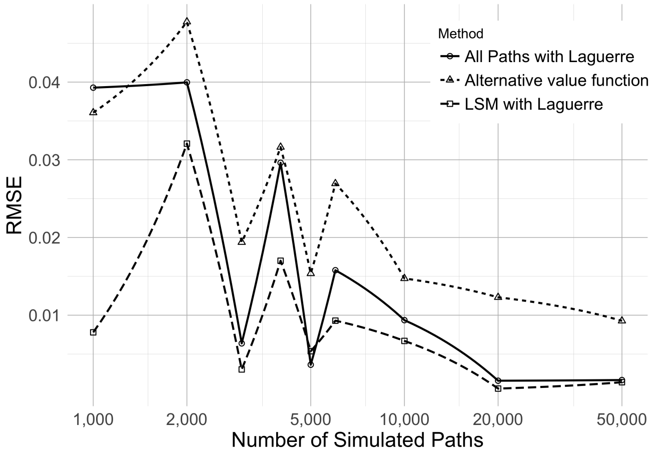
  where V_t denotes the option value at time t, Δt is the interval between t and $t-1$, S_t is the stock price, K is the strike, $\widehat{H}_t$ is the approximation to the expected holding value.

- The selection of paths: Tsitsiklis and Van Roy (2001) uses all paths, but LSMC selects in-the-money paths to fit the regression.

Figure 4.10 depicts the approximation results from:

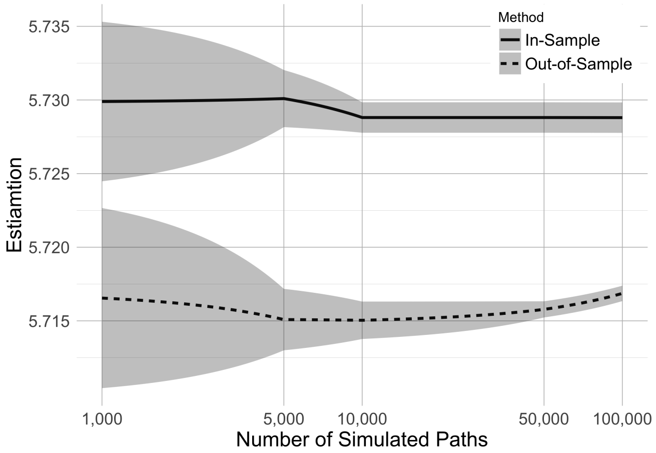
- LSMC using all paths in the regression (denoted as *All Paths with Laguerre*);
- LSMC using the value function in Tsitsiklis and Van Roy (2001) (denoted as *Alternative value function*); and
- LSMC in Longstaff and Schwartz (2001) (denoted as *LSM with Laguerre*).

The RMSEs show identical convergence patterns across three methods, while Longstaff and Schwartz (2001) 's method provides the highest accuracy. Further, using the value function suggested by Longstaff and Schwartz (2001) helps to reduce the estimation bias, while using only in-the-money paths for regression helps to speed up the convergence.



Values are computed using the Laguerre polynomial of three degrees.

Figure 4.10: Comparison of LSMC and Tsitsiklis and Van Roy (2001)



Values are computed using LSMC with European options as the control variate and regressions in Rasmussen (2005).

Figure 4.11: Comparison of in-sample and out-of-sample LSMC estimation

4.1.5 *Out-of-Sample Performance*

Before proceeding with a discussion of high dimensional options, it is useful to examine how the LSMC approximation would change for out-of-sample (independent) paths. Specifically, we first run the standard LSMC, record the estimated regression parameters for given paths; then, simulate a new set of paths and predict the expected holding on these new paths using the regression. Figure 4.11 finds that this procedure results in a biased lower estimator, but the convergence pattern is similar to the standard LSMC (*i.e.*, in-sample estimation). Therefore, by using independent paths, we can form a lower bound for the LSMC estimator.

4.2 MULTI-ASSET BERMUDAN OPTIONS

Up to this point, the focus of the discussion has been on the valuation of Bermudan options on a single asset. We expect the performance of LSMC will hold in high dimensional problems. Based on the results of Figure 4.1 and Figure 4.3, it is reasonable to compare the performance of LSMC with the random tree.

4.2.1 *Comparison of LSMC and Random Tree*

The test-bed is a Bermudan call on the maximum of two stocks, $S_{1,t_i} and S_{2,t_i}$. Both stocks follow the same risk-neutral process as the single stock S_{t_i} specified previously. The call option has the payoff:

$$h(S_{1,t_i}, S_{2,t_i}) = \max(\max(S_{1,t_i}, S_{2,t_i}) - K, 0),$$

for $i = 0, 1, \ldots, N$. The parameters of stock prices movement process and the option details are provided in Table 4.2.

Table 4.2: Specifications for Bermudan call on two stocks

Option Specifications	
Option Expiry Date (T)	End of 1 year
Strike Price (K)	100.00
Exercise Opportunity	End of every 4 months
Number Exercise Opportunities (N)	3
Annualized Risk Free Rate (r)	0.05
Stock Specifications	
Annualized Stock Volatility ($\sigma_i, i = 1, 2$)	0.2
Annualized Dividend Rate ($q_i, i = 1, 2$)	0.1
Correlation between log returns of $S_{1,t}$ and $S_{2,t}$	0.3

We vary the spot price for both stocks from 70 to 120, assume $S_{1,t_0} = S_{2,t_0}$, and compute the benchmark values[5] using two dimensional binomial trees with 1,000 steps.

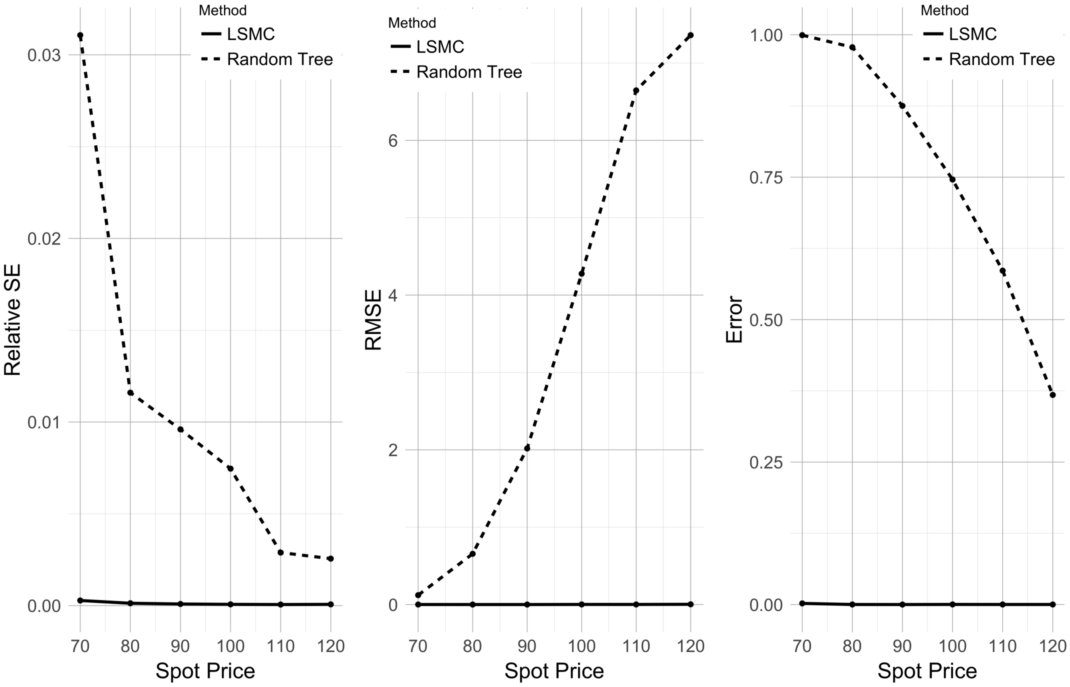
In Figure 4.12, three criteria are compared between LSMC and the random tree. By limiting the computation time to 2 minutes, we are able to use either 20,000 paths in LSMC or 20 branches at each node for the tree. Across six spot prices examined, LSMC has more accurate estimators than the random tree in this two-asset option. For the random tree, the mis-pricing errors and relative SEs improve as the option becomes in-the-money, while the RMSEs become higher for the same spot prices. This is consistent with the single dimensional Bermudan option in Figure 4.1. This two dimensional problem shows that lattice based method is not suitable for pricing high dimensional options.

4.2.2 *The Accuracy and Convergence of LSMC in High Dimensions*

Figure 4.13 through Figure 4.15 examine two enhancement approaches for LSMC: regression schemes and random number generation process. The analyses examine the accuracy and convergence of LSMC for pricing a Bermudan call on the geometric average of five assets. To get the benchmark, this option can be reduced to one dimension and valued accurately via binomial tree method.

---

[5] The benchmark values are: 0.2370 (for a spot price of $S_{t_0,1}$=70), 1.2590 ($S_{t_0,1}$=80), 4.0770 ($S_{t_0,1}$=90), 9.3610 ($S_{t_0,1}$=100), 16.9240 ($S_{t_0,1}$=110) and 25.9800 ($S_{t_0,1}$=120). These values can be found in Appendix A.



(a) Relative SE (b) RMSE (c) Error

Values for LSMC are computed using LSMC with the first three Laguerre polynomials.
Values for the random tree are computed using a two dimensional random tree as suggested by Broadie and Glasserman (1997)
Detailed result can be found in Appendix A.

Figure 4.12: Comparison of the random tree and LSMC for a Bermudan max-call option

4.2.3 *Bermudan Options on the Geometric Average of Five Assets*

Suppose there are five stocks $S_{j,t_i}, j = 1, \ldots, 5, i = 0, 1, \ldots, N$ with the same spot price. Then, a geometric average call with a strike K has the payoff,

$$h(S_{j,t_i}) = \max((\Pi_{j=1}^{5} S_{j,t_i})^{1/5} - K, 0),$$

for $j = 1, \ldots, 5$.

Following Kemna and Vorst (1990), the dimension of option‘s can be reduced to one by introducing a new asset Y_t. Suppose, the log of stock prices, $log(S_{j,t_i})$, follow a multivariate Brownian motion with a constant covariance matrix Σ, and a constant risk-neutral drift rate,

$$\mu_j = r - \frac{1}{2}\sigma_j,$$

where r is the risk-free rate, σ_j are the annualised volatility for stock S_{j,t_i}. Then, Y_t follows the risk-neutral process,

$$dY_t = (\mu_{adj} - q_{adj})dt + \sigma_{adj} dW_t,$$

where $\{W_t\}_{0 \leq t \leq T}$ is a Brownian motion; and

$$\mu_{adj} = r - \frac{1}{10}\sum_{j=1}^{5} \sigma_j^2;$$

$$\sigma_{adj}^2 = \frac{1}{5^2}\sum_{j=1}^{5}\sum_{k=1}^{5} D_{jk};$$

where D is the covariance matrix for $S_j, j = 1, \ldots, 5$, and

$$q_{adj} = \frac{1}{2}\left(\frac{1}{5}\sum_{j=1}^{5}\sigma_j^2 - \sigma_{adj}^2\right).$$

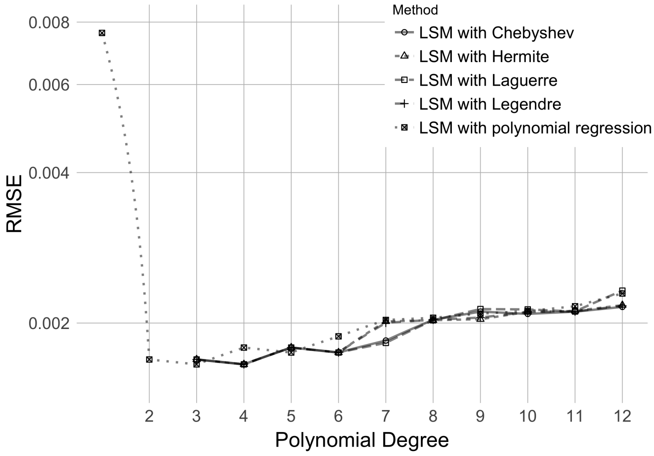
We construct the Bermudan call option based on specifications in Table 4.3. The benchmark value from binomial tree using 1,000 steps is 3.0495.

REGRESSION SCHEMES Figure 4.13 compares the usage of four defined polynomials and one simple polynomial up to 12 degrees in a single dimension. The results report

Table 4.3: Specifications for Bermudan call on five stocks

Option Specifications	
Option Expiry Date (T)	End of 1 year
Strike Price (K)	100.00
Exercise Opportunity	End of every 4 months
Number Exercise Opportunities (N)	3
Annualized Risk Free Rate (r)	0.07
Stock Specifications	
Annualized Stock Volatility ($\sigma_j, j = 1, \ldots, 5$)	0.2
Annualized Dividend Rate ($q_j, j = 1, \ldots, 5$)	0.1
Correlation between log returns of $S_{j,t_i}, j = 1, \ldots, 5$	0.3

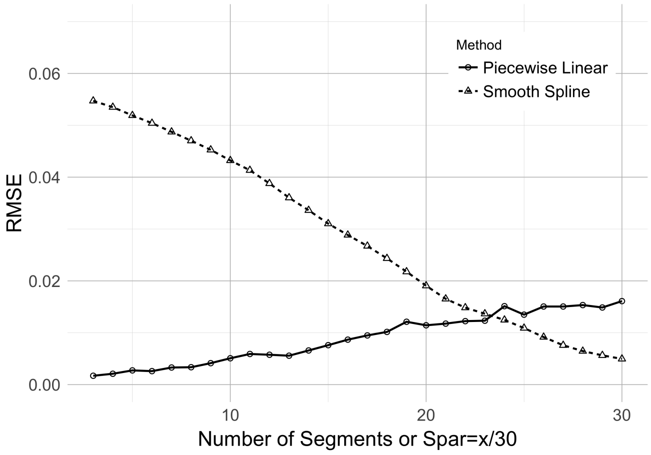
identical performance from the four defined polynomials, and slightly higher RMSEs for the simple polynomial. Across the examined defined polynomials, the RMSEs are trending up with the number of polynomial terms, which provides evidence that using higher degree polynomials without increasing the number of simulated paths deteriorates the estimation Accuracy. The lowest RMSE occurs at using four defined polynomials in this example.



Values are computed by replacing the regression component in LSMC by corresponding polynomials with 10,000 paths.

Figure 4.13: Accuracy of LSMC with different regression polynomials

In Figure 4.14, it contains the results using the piecewise linear regression and smoothing spline. These results show evidence that, in this high dimensional problem, with 10,000 paths, the best choice would be a simple linear regression rather than nonlinear alternatives. In particular, a fine partition of piecewise linear regression does not yield good prediction for expected holding value. This in not the case in a single dimensional option depicted in (4.6), where we found more segments in the regression provide higher accuracy.



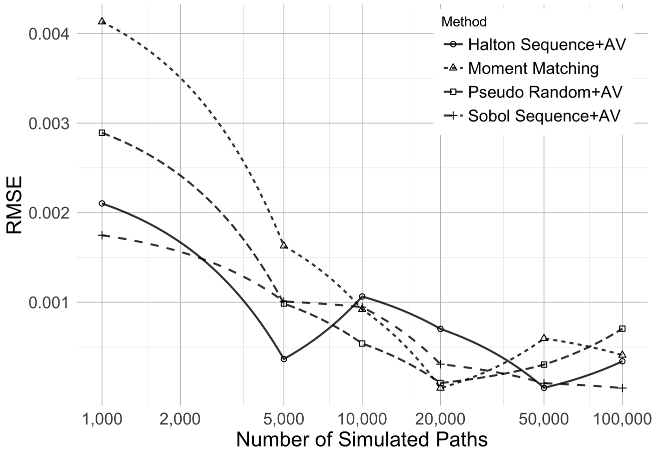
Values are computed by replacing the regression component in LSMC by respective regression schemes with 10,000 paths for each asset. For piecewise linear regression, the x-axis is the number of segments employed; and $x < 3$ is not examined. For smoothing spline regression, it is the x as in spar $= x/30$. The estimation degrees of freedom is 10.

Figure 4.14: Accuracy of LSMC with nonlinear regression

LOW-DISCREPANCY SEQUENCES Figure 4.15 contains the results from using a set of correlated Halton and Sobol sequences. With the assistance of antithetic variables, it is evident that using either Halton or Sobol sequences gives more accurate estimator when small number of simulation paths are deployed. However, the usage of Halton sequences leads to unstable numerical results, while all other methods converge at using around 20,000 paths. Applying Halton sequences is not suitable in this high dimensional problem. Overall, the usage of low-discrepancy sequences helps the convergence of LSMC in high dimensions but not dramatically.

4.3 APPROXIMATE AMERICAN OPTIONS WITH BERMUDAN OPTIONS

As the number of exercises opportunities $N \to \infty$, the value of Bermudan option converges to the American option. Figure 4.16, panel (a), illustrates the convergence of the above Bermudan call on the geometric average of five assets (spot prices are 100 for all stocks) to an American option with 5,000 and 10,000 simulated paths. With 200 opportunities,



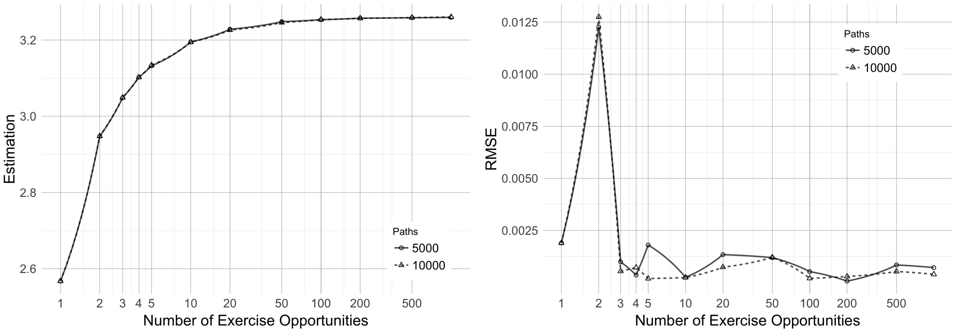
Values are computed using LSMC with control variates in Rasmussen (2005).The random number generation process for each asset follows the one dimensional case. The correlations among assets are included by *Cholesky decomposition*.

Figure 4.15: Convergence of LSMC with low-discrepancy sequences

the estimated price is 3.2570 compared to the benchmark value 3.2601; and with 1,000 opportunities, the estimation error is reduced to less than 0.0001. The computation time grows linearly with the number of exercise opportunities. For example, with 5,000 paths and 500 exercise opportunities the computation time is 7.2 minutes. Panel (b) of Figure 4.16 shows that increasing the number of simulation paths from 5,000 to 10,000 does not improve the accuracy significantly in terms of RMSEs. The LSMC estimation is valid for both the American call and put as illustrated in Figure 4.17. In both cases, we can observe the estimated option prices converge to the benchmark, and the standard error for the estimator (represented by the shaded area) is decreasing with more simulated paths.

Figure 4.18 reports the results for American calls on stocks with different spot prices. The LSMC estimators are plotted against the quadratic approximations from Barone Adesi and Whaley (1987) and the binomial tree method. The mis-pricing errors are narrowing as the the option becomes in-the-money for both LSMC and the quadratic approximation. When the option is out-of-the-money, the quadratic approximation provides biased upper estimators.

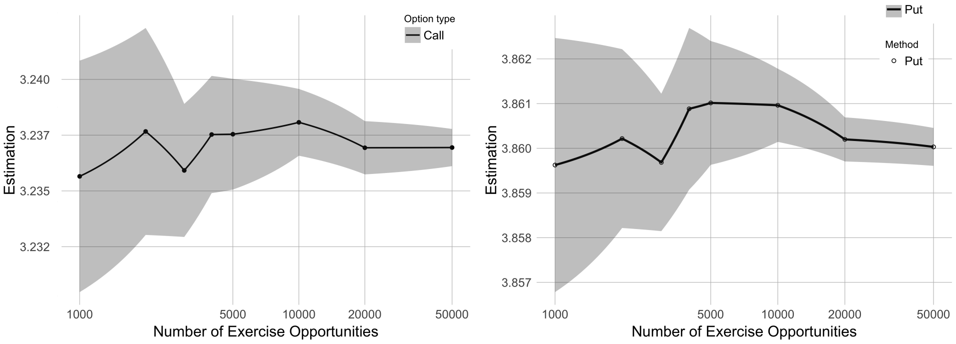
The extension for LSMC from Bermudan to American is simple to implement, and the results are much more accurate compared to the quadratic approximation, and more flexible compared to the binomial tree. One issue arising from increasing number of exer-



(a) Convergence of the estimator (b) RMSE of the estimator

Values are computed using LSMC on a Bermudan call with control variates and regression scheme following Rasmussen (2005);

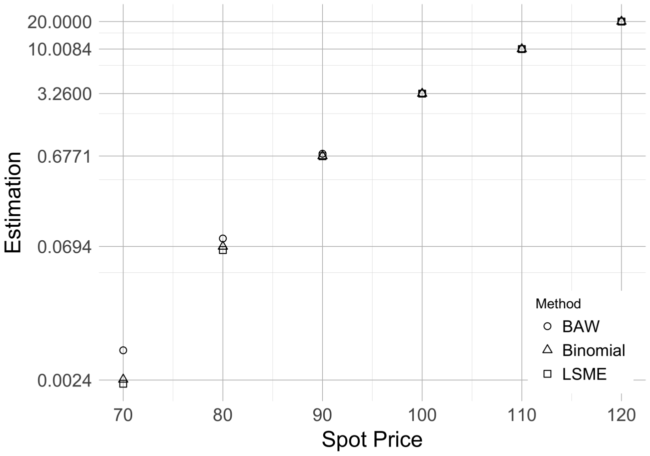
Figure 4.16: Convergence of the Bermuda to American



(a) Call (b) Put

We used antithetic and control variates in LSMC estimators for both options. The control variates and the polynomials for regression are specified in Rasmussen (2005).
The error bounds are depicted using the standard error of each estimator with a 95% confidence interval.

Figure 4.17: Comparison of the convergence of American call and put options



1. Values are computed using LSMC on a Bermudan call with 500 exercise opportunities and 10,000 paths; control variates and regression scheme following Rasmussen (2005);
2.Values are computed using Barone Adesi and Whaley (1987)'s quadratic approximation on American calls of an underlying asset for a given cost-of-carry rate (*i.e.*, dividend yield).
3. Values are computed using binomial tree of 3,000 steps.

Figure 4.18: Comparison of LSMC^1, Barone Adesi and Whaley (1987)^2 and binomial tree^3

cise opportunities is the computation time required. Brownian bridge is proposed to make the simulation of stock paths in American options more efficient (Chaudhary, 2005).

4.4 SUMMARY

The analyses confirmed the least-squares Monte Carlo method is the most efficient Monte Carlo method for pricing options with early exercise opportunities. For the ease of benchmarking with closed-form solutions, the analyses assumed all assets following a risk-neutral process. In future studies, we are able to relax this assumption and price options with underlying assets following other processes such as jump-diffusion process and variance gamma process (Joshi (2008)). Another direction for further investigation is to improve the efficiency of LSMC. First, we have seen regression schemes with high degree polynomials or smoothing splines did not perform well in high dimensional problems, but we do not know whether increasing the number of simulated paths will improve the performance of these regressions schemes. Second, the usage of low-discrepancy sequences could potentially be enhanced via Brownian bridge. Lastly, other control variates can be studied in addition to European options.

5 LSMC FOR REAL OPTIONS ANALYSIS

An analogous of financial options with early exercise provisions in the area of real options is an option to delay a project. We will see the pricing of these options shares similar characteristics. For example, in a capital budgeting problem, a manager will decide whether to invest in a project; the project will be rejected if it has a negative net present value (NPV) based on the conventional discounted cash flow method (DCF):

$$\begin{cases} \text{reject if: NPV} < 0; \\ \text{accept, otherwise.} \end{cases}$$

However, if a manager has the option to invest in a project any time in the future before time T, we will find the NPV can be expressed in terms of expectation of discounted future project value V_τ,

$$\text{NPV} = \sup_{0 \leq \tau \leq T} \mathbb{E}_0[e^{r\tau} V_\tau], \tag{5.1}$$

given information available today. The objective is to find the optimal investment time τ that maximize the NPV in equation (5.1). If this NPV is non-negative, we will invest in the project. Otherwise, the project is rejected. The least-squares Monte Carlo method is applied here to approximate future project values.

5.1 VALUE OF WAITING TO INVEST

McDonald and Siegel (1986) studies the following problem: a project requires initial investment K; the project will generate a stream of cash flows $S_t, t = 1, \ldots, T$ at the end of each year for T years, these cash flows follow a risk-neutral process,

$$dS_t/S_t = (r - \delta)dt + \sigma dW_t,$$

where r is the risk-free rate, δ is the convenience yield [1] from implementing the project, σ is the volatility of the project cash flow, and $\{W_t\}_{0\leq t\leq T}$ is a Brownian process.

At time $t = 0$, we can compute the expected NPV of this project using DCF as

$$V(S_0, 0) = \mathbb{E}_0\left[\sum_{t=1}^{T} S_t e^{-\delta t} - \sum_{t=1}^{T} C_t e^{-rt} - K\right],$$

where $C_t, t = 1, 2, \ldots, T$ are the cost incurred by the project.

Since the decision maker has the option to invest in the project either today or sometimes in the future, the associated future cash flow uncertainty will add value to the project. According to McDonald and Siegel (1986), the project value embedding the option to delay, $V(S_t, t)$, can be computed by the optimal commodity price S_τ, above which we should invest in the project, and corresponding investment time τ as follows:

$$V(S_\tau, \tau) = \left(\frac{S_\tau}{\delta} - \frac{C}{r} - K\right)\left(\frac{S}{S_\tau}\right)^d, \tag{5.2}$$

where

$$\begin{aligned} S_\tau &= \frac{\delta d}{d-1}(C/r + K), \\ m &= 1/2 - (r-\delta)\sigma^2, \\ d &= m + \sqrt{m^2 + 2r/\sigma^2}; \end{aligned}$$

and the NPV is

$$V(S_0, 0) = \mathbb{E}_0[e^{-r\tau} V(S_\tau, \tau)],$$

where S_τ is the price above which it is optimal to make the investment. Note that if we assume constant cash flow S and cost C for the project, for a perpetual project ($T \to \infty$), the optimal project value is

$$V(S_\tau, \tau) = \frac{S}{\delta} - \frac{C}{r} - K,$$

which is the same as the DCF estimation. This solution provides a good benchmark for our Monte Carlo implementation.

---

[1] The convenience yield is the benefit from holding a certain commodity. For example, a electronic manufacturer may benefit from holding physical copper to meet the unexpected shortage.

We apply LSMC to pricing the option to delay. In particular, we are interested in finding the optimal investment time τ in

$$V(S_0, 0) = \sup_{\tau \in \{0,1,\ldots,T\}} \mathbb{E}_0[e^{r\tau} V(S_\tau, \tau)].$$

The estimation is done via value iteration:

1. At the end of investment horizon T, the project NPV is calculated assuming investment starts at time T:

   $$V(S_T, T) = \max(S_T e^{-\delta T} - C_T e^{-rT} - K, 0),$$

   if there is immediate profit and cost cash flow from investing in the project, and set $\tau = T$.

2. For $t = T-1, T-2, \ldots, 0$, we update the project value iteratively via

   $$V(S_t, t) = \max\left(\sum_{i=t}^{T} S_i e^{-\delta i} - \sum_{i=t}^{T} C_i e^{-ri} - K, \mathbb{E}_t[V(S_\tau, \tau)]\right), \tag{5.3}$$

   if $\sum_{i=t}^{T} S_i e^{-\delta i} - \sum_{i=t}^{T} C_i e^{-ri} - K \geq \mathbb{E}_t(V(S_\tau, \tau))$, update $\tau = t$; do not update τ otherwise.

3. At $t = 0$, we estimate the project NPV as

   $$V(S_0, 0) = \max\left(\sum_{i=0}^{T} S_i e^{-\delta i} - \sum_{i=0}^{T} C_i e^{-ri} - K, \mathbb{E}_0[V(S_\tau, \tau)]\right). \tag{5.4}$$

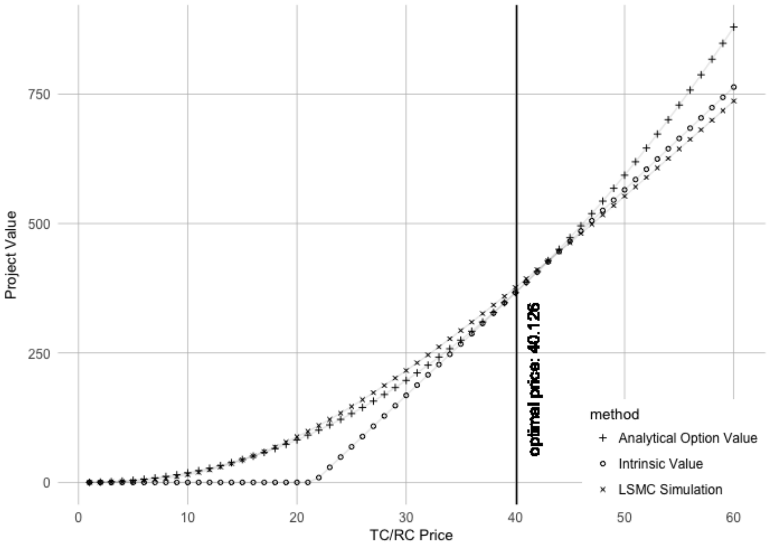
We use LSMC algorithm to approximate the expectation in equations (5.3) and (5.4).

5.2 CASE STUDY: MINING TECHNOLOGY

The analysis is based on the case study in Crundwell (2008). A mining company wishes to decide whether they should invest in a new copper concentrate refinement technology. The technology will enable the company to produce high-grade copper. The price of the refined copper concentrate is determined by treatment costs and refining charges, or TC/RC, in US c/lb. The following table specifies the parameters for this project:

Project Specification	
Current copper concentrate price (S_0)	29 US c/lb
Annualised copper concentrate volatility (σ)	0.2
Production cost (C)	17 US c/lb
Annualised convenience yield (δ)	0.05
Annualised risk-free rate (r)	0.05
Initial investment (K)	90 US c/lb

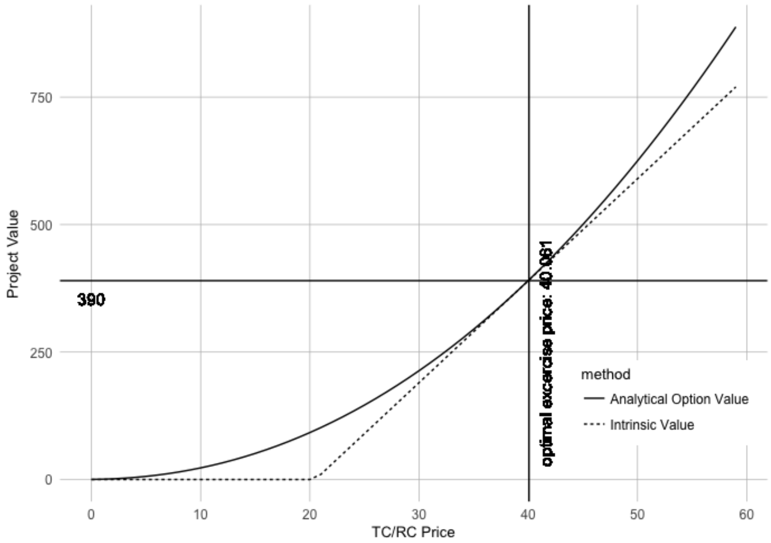
We further assume the manager can make the decision at the end of each year for 100 years. Results in Figure 5.1 shows the option to delay inflates the project value. At the current TC/RC price of 29 US c/lb, the DCF method returns a NPV of 148.175 US c/lb, while the analytical solution gives 183.083 US c/lb and the LSMC returns 187.144 US c/lb. Note that the trajectory of project NPV follows that of a call option, where the spot price is underlying commodity value and the strike is the initial investment K plus discounted production cost.



1. Values are computed using Longstaff and Schwartz (2001), the response is the simulated future project value, and covariates are the current commodity prices. First three Laguerre polynomials are used as the regression basis. 5000 paths are simulated over a 100-year horizon. 10 Monte Carlo simulations were used for 1 estimation.
2. Values are computed using McDonald and Siegel (1986).
3. Intrinsic values are computed using the discounted cash flow method.

Figure 5.1: Comparison of project value between LSMC[1], McDonald and Siegel (1986)[2] and DCF[3]

The optimal investment price given by equation (5.2) is 40 US c/lb. This price is 11 US c/lb higher than the current price, and almost doubles the price at which NPV is zero (*i.e.*, approximately 21.5 US c/lb). Also, at this optimal price, the NPV obtained from the analytical solution is closest to the intrinsic NPV (390.227 US c/lb compared to 389.972 US c/lb, the corresponding LSMC estimation is 395.462 US c/lb). Indeed, if we calculate the analytical solution with an infinite time horizon, this point[2] is the point of tangency between the curve for project NPVs and the line for intrinsic NPVs as in illustrated in Figure 5.2.



Analytical values are computed using McDonald and Siegel (1986)'s method.
Intrinsic values are computed using the discounted cash flow method.

Figure 5.2: The optimal investment price and the point of tangency

An intuitive explanation of choosing this point is given by the delta of an option: if the price is lower than the optimal price, the project value with embedded options will be less sensitive to the movement of the copper price as indicated by the curve slope, and vice versa. Only at the point of tangency, the estimated project value has the same price sensitivity as the DCF estimation (*i.e.*, the same slope). Therefore, the decision maker will be protected from excessive price movement at the optimal investment price, as the project value with real option is the same as the DCF estimation.

---

[2] Corresponding to a project NPV of 390 US c/lb.

In LSMC, we suggest locating this optimal investment price by the difference between the simulated value and the corresponding intrinsic value. The optimal investment price occurs at the the first point when this difference is lower than a predetermined level (we use 0.001).

The performance of LSMC in the above example is not comparable with the analytical solution. With a horizon of 100 years, the LSMC method takes more than a minute to produce an estimation with spot commodity price 29 US c/lb using 5000 paths, but the estimation error is large.

5.3 OPTIONS TO DELAY ON TWO FACTORS

Nonetheless, if the project is dependent on multiple factors, where there is no closed-form solution, LSMC will be useful. We demonstrate this by allowing the cost C in the above example to follow a risk-neutral process:

$$\frac{dC_t}{C_t} = r_c dt + \sigma_c dW_t^c, \tag{5.5}$$

where the drift rate $r_c = 0$, the annualised volatility $\sigma_c = 0.05$ and $\{W_t^c\}_{0 \le t \le T}$ is a Brownian process. The correlation between the income and cost cash flow is $\rho = 0.5$. We limit the investment horizon to 10 years and use 10,000 simulation paths. All other parameters are the same as the previous *mining technology* case study.

The optimal exercise price is 48 US c/lb, the estimated NPV of the project is 159 US c/lb from LSMC and 149 US c/lb from DCF. The inclusion of an additional stochastic factor significantly reduces the project value even with the option being embedded. Figure 5.3 shows that the option premium converges to zero as the cooper concentration price increases.

We determined the the optimal investment price at 74 US c/lb, when the difference between LSMC and DCF estimation is first observed lower than 0.001. With two stochastic processes for the project value, the optimal price to investment is significantly pushed up. Even at this significantly higher price, the realised project value is 350 US c/lb, which is 50 US c/lb lower than the previous example.

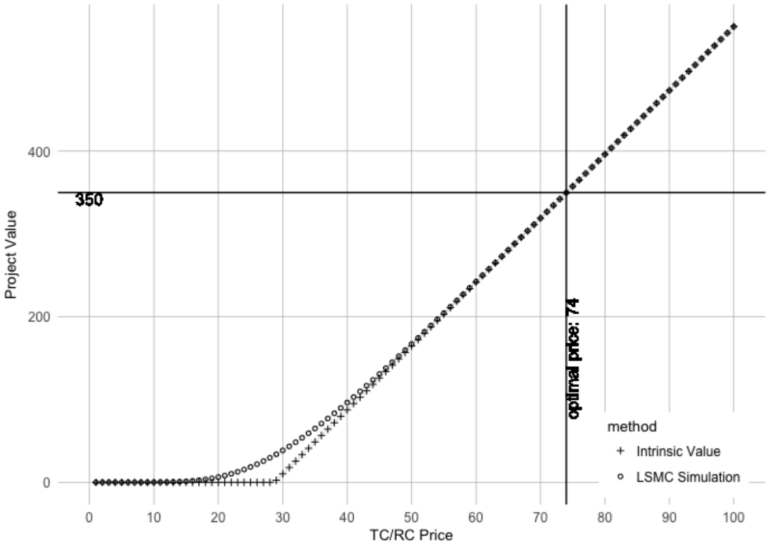


Figure 5.3: LSMC estimation of the optimal investment price

5.4 SUMMARY

The valuation of projects with the options to delay shares similar characteristics as the pricing for a call option with early exercise provisions. The illustrated cases showed significant value could be added by these options. However, it should be noticed that the high project NPV produced via real option analysis alone cannot be used to justify the acceptance of a project that is rejected by DCF. In particular, a decision maker should assess the real-world probability of the underlying assets to reach the optimal investment price. If this probability is low, then, we should not take the project. Another case to consider is whether the new technology will be exclusive to the company. If other companies can develop the technology to produce high-quality copper as well, the project will not realise the estimated NPV. Therefore, although real option analysis helps us uncover the time value embedded in decision making process, for a more prudent result, we should treat it as an upper bound at which the project can worth rather than a lower bound.

6 CONCLUSIONS AND DIRECTIONS OF FUTURE RESEARCH

Options with early exercise provisions are important financial derivatives. Monte Carlo methods provide resilient and efficient solutions for pricing these options. Our results indicated superior performance of the least-squares Monte Carlo method; and found European options are effective control variates for improving the accuracy of the Monte Carlo estimation for Bermudan option prices. The extension to the option to delay demonstrated that Monte Carlo methods are useful in real investment decision problems other than pricing financial derivatives. In light of these critical findings, we believe that our analyses may contribute to the selection of efficient Monte Carlo methods for option valuation as well as the improvements on the least-squares Monte Carlo method.

Monte Carlo methods could be applied quite reliably in other financial decision contexts without a significant degradation in performance. Future work will mainly cover the improvements and additional applications of the least-squares Monte Carlo method, including (1) the choice of regression schemes under different number of simulation paths; (2) the application of the algorithm for options with underlying assets following stochastic processes other than the geometric Brownian motion; and (3) the extension to complex capital budgeting problems with embedded real options such as the option to abandon and expand.

A APPENDIX: SIMULATION RESULTS

A.1 SIMULATION RESULTS FOR BERMUDAN CALLS ON A SINGLE ASSET

Table A.1: Comparison of LSMC and EM-C algorithm for a Bermudan call

Method	Branches	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with control	5000	70	0.12111	0.0001	0.1212	-0.0008	0.0001	0.077%
		80	0.67000	0.0003	0.6699	0.0001	0.0000	0.049%
		90	2.30358	0.0006	2.3030	0.0003	0.0006	0.024%
		100	5.73010	0.0010	5.7299	0.0000	0.0011	0.017%
		110	11.33962	0.0015	11.3410	-0.0001	0.0014	0.014%
		120	20.00000	0.0000	20.0000	0.0000	0.0000	0.000%
	20000	70	0.1212	0.0001	0.1212	0.0002	0.0002	0.046%
		80	0.6698	0.0001	0.6699	-0.0002	0.0002	0.017%
		90	2.3029	0.0003	2.3030	0.0000	0.0001	0.011%
		100	5.7288	0.0005	5.7299	-0.0002	0.0002	0.009%
		110	11.3397	0.0006	11.3410	-0.0001	0.0013	0.005%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%
EM-C	5000	70	0.1180	0.0019	0.1212	0.0265	0.0032	1.636%
		80	0.6714	0.0072	0.6699	0.0022	0.0015	1.065%
		90	2.3055	0.0114	2.3030	0.0011	0.0025	0.495%
		100	5.7438	0.0126	5.7299	0.0024	0.0139	0.220%
		110	11.3304	0.0154	11.3410	0.0009	0.0106	0.136%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%
	20000	70	0.1207	0.0009	0.1212	0.0038	0.0005	0.712%
		80	0.6688	0.0029	0.6699	0.0016	0.0011	0.430%
		90	2.2952	0.0056	2.3030	0.0034	0.0078	0.242%
		100	5.7309	0.0076	5.7299	0.0002	0.0010	0.132%
		110	11.3442	0.0057	11.3410	0.0003	0.0032	0.050%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%

Each estimation was obtained using 25 Monte Carlo estimations. 4 digits are reported for each estimation and accuracy measures. **Computation time (for 1 Monte Carlo Estimation):** LSMC (5000 paths, spot price = 100), 0.1 seconds; LSMC (20000 paths, spot price = 100), 0.3 seconds; For options deep in the money (spot price = 120), the computation time required will increase by around 0.02 seconds for 5000 paths and decrease by around 0.03 seconds if the the option is out of money ((spot price = 70). EM-C algorithm (5000 paths), 0.9 seconds;EM-C algorithm (20000 paths), 2.5 seconds. Stock spot price as indicated in the table. Est. denotes estimator. S.E. denotes standard error of the estimator and a 95%confidence interval was calculated using the low and the high estimator in the tree method. Each estimator was obtained using 25 Monte Carlo estimations.

Table A.2: Comparison of random tree and stochastic mesh for a Bermudan call

Method	**Paths**	**Stock Price**	**Low est.**	**S.E. Low est**	**High est.**	**S.E. Low est**	**Confidence Interval**	**Point est.**	**Benchmark**	**Error**	**RMSE**	**Relative SE**
Stochastic Mesh	10	70	0.1186	0.0013	0.1214	0.0005	[0.1165,0.1222]	0.1200	0.1210	0.0082	0.0038	0.393%
		80	0.6283	0.0055	0.6924	0.0021	[0.6193,0.6959]	0.6603	0.6700	0.0144	0.0179	0.318%
		90	2.1420	0.0169	2.5020	0.0082	[2.114,2.516]	2.3220	2.3030	0.0083	0.0594	0.352%
		100	5.4380	0.0270	6.2790	0.0154	[5.394,6.305]	5.8590	5.7310	0.0223	0.1600	0.263%
		110	11.1800	0.0353	12.1500	0.0239	[11.12,12.18]	11.6600	11.3410	0.0282	0.3490	0.205%
		120	20.3500	0.0149	20.7400	0.0161	[20.33,20.77]	20.5500	20.0000	0.0274	0.5510	0.078%
	50	70	0.1204	0.0003	0.1215	0.0001	[0.12,0.1217]	0.1210	0.1210	0.0002	0.0009	0.098%
		80	0.6510	0.0019	0.6849	0.0008	[0.6479,0.6863]	0.6679	0.6700	0.0031	0.0061	0.127%
		90	2.2340	0.0056	2.3940	0.0053	[2.225,2.399]	2.3140	2.3030	0.0049	0.0225	0.229%
		100	5.5830	0.0113	5.9650	0.0062	[5.564,5.975]	5.7740	5.7310	0.0075	0.0579	0.107%
		110	11.1300	0.0176	11.6400	0.0128	[11.1,11.66]	11.3900	11.3410	0.0040	0.0847	0.112%
		120	20.0500	0.0048	20.1000	0.0051	[20.04,20.11]	20.0700	20.0000	0.0037	0.0774	0.025%
Stochastic Tree	50	70	0.1210	0.0004	0.1210	0.0004	[0.1203,0.1216]	0.1210	0.1210	0.0001	0.0019	0.317%
		80	0.6683	0.0017	0.6686	0.0017	[0.6655,0.6713]	0.6685	0.6700	0.0023	0.0082	0.248%
		90	2.2963	0.0032	2.3009	0.0031	[2.291,2.306]	2.2986	2.3030	0.0019	0.0161	0.136%
		100	5.7187	0.0014	5.7401	0.0012	[5.7164,5.7419]	5.7294	5.7310	0.0003	0.0064	0.020%
		110	11.2796	0.0035	11.3595	0.0020	[11.2738,11.3627]	11.3195	11.3410	0.0019	0.0247	0.018%
		120	19.9933	0.0097	20.0161	0.0082	[19.9773,20.0296]	20.0158	20.0000	0.0008	0.0326	0.041%
	70	70	0.1212	0.0004	0.1212	0.0004	[0.1206,0.1218]	0.1212	0.1210	0.0018	0.0018	0.305%
		80	0.6679	0.0011	0.6681	0.0011	[0.6659,0.6699]	0.6680	0.6700	0.0030	0.0059	0.170%
		90	2.3005	0.0022	2.3038	0.0021	[2.2969,2.3072]	2.3022	2.3030	0.0004	0.0106	0.092%
		100	5.7222	0.0011	5.7367	0.0011	[5.7203,5.7384]	5.7295	5.7310	0.0003	0.0056	0.019%
		110	11.3188	0.0016	11.3547	0.0012	[11.3161,11.3567]	11.3367	11.3410	0.0004	0.0077	0.011%
		120	19.9882	0.0085	19.9983	0.0078	[19.9742,20.0111]	20.0046	20.0000	0.0002	0.0261	0.039%
	100	70	0.1212	0.0003	0.1213	0.0003	[0.1208,0.1216]	0.1212	0.1210	0.0021	0.0013	0.221%
		80	0.6682	0.0012	0.6683	0.0012	[0.6662,0.6702]	0.6683	0.6700	0.0026	0.0059	0.173%
		90	2.3017	0.0015	2.3040	0.0015	[2.2992,2.3063]	2.3028	2.3030	0.0001	0.0073	0.064%
		100	5.7263	0.0010	5.7367	0.0010	[5.7246,5.7383]	5.7315	5.7310	0.0001	0.0049	0.017%
		110	11.3297	0.0014	11.3504	0.0012	[11.3274,11.3524]	11.3401	11.3410	0.0001	0.0065	0.011%
		120	20.0080	0.0066	20.0104	0.0064	[19.9971,20.0209]	20.0141	20.0000	0.0007	0.0278	0.032%

Stock spot price as indicated in the table. Est. denotes estimator. S.E. denotes standard error of the estimator and a 95%confidence interval was calculated using the low and the high estimator. Each estimator was obtained using 25 Monte Carlo estimations. Point estimator was calculated as $(\max(\text{SpotPrice} - \text{K}, LowEstimator) + HighEstimator)/2$. The relative SE is calculated as the estimation divided by its standard error.

Table A.3: Convergence of LSMC with low-discrepancy sequences in a single dimension

Method	Paths	Stock Price	Point est.	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with Halton and AV	1000	100	5.7374	0.0013	100	0.0013	0.0075	0.023%
	2000	100	5.7334	0.0009	100	0.0006	0.0035	0.015%
	3000	100	5.7317	0.0009	100	0.0003	0.0018	0.015%
	4000	100	5.7309	0.0007	100	0.0002	0.0010	0.012%
	5000	100	5.7308	0.0008	100	0.0002	0.0009	0.013%
	6000	100	5.7308	0.0007	100	0.0001	0.0009	0.013%
	10000	100	5.7302	0.0007	100	0.0001	0.0003	0.012%
	20000	100	5.7300	0.0007	100	0.0000	0.0001	0.011%
	50000	100	5.7297	0.0006	100	0.0000	0.0002	0.011%
	100000	100	5.7297	0.0006	100	0.0000	0.0002	0.011%
LSM with Pseudo and AV	1000	100	5.7349	0.0019	100	0.0009	0.0050	0.033%
	2000	100	5.7325	0.0016	100	0.0004	0.0026	0.027%
	3000	100	5.7304	0.0015	100	0.0001	0.0005	0.026%
	4000	100	5.7300	0.0011	100	0.0000	0.0001	0.019%
	5000	100	5.7307	0.0011	100	0.0001	0.0008	0.019%
	6000	100	5.7310	0.0008	100	0.0002	0.0011	0.015%
	10000	100	5.7306	0.0008	100	0.0001	0.0007	0.013%
	20000	100	5.7302	0.0005	100	0.0001	0.0003	0.009%
	50000	100	5.7306	0.0003	100	0.0001	0.0007	0.005%
	100000	100	5.7304	0.0002	100	0.0001	0.0005	0.004%
LSM with Pseudo	1000	100	5.7292	0.0025	100	0.0001	0.0007	0.044%
	2000	100	5.7289	0.0016	100	0.0002	0.0010	0.027%
	3000	100	5.7311	0.0015	100	0.0002	0.0012	0.026%
	4000	100	5.7320	0.0013	100	0.0004	0.0021	0.022%
	5000	100	5.7313	0.0013	100	0.0002	0.0014	0.024%
	6000	100	5.7297	0.0010	100	0.0000	0.0002	0.018%
	10000	100	5.7303	0.0008	100	0.0001	0.0004	0.014%
	20000	100	5.7303	0.0006	100	0.0001	0.0004	0.011%
	50000	100	5.7305	0.0004	100	0.0001	0.0006	0.007%
	100000	100	5.7308	0.0003	100	0.0002	0.0009	0.005%
LSM with Halton	1000	100	5.7319	0.0016	100	0.0003	0.0020	0.028%
	2000	100	5.7300	0.0014	100	0.0000	0.0001	0.024%
	3000	100	5.7299	0.0014	100	0.0000	0.0000	0.025%
	4000	100	5.7300	0.0015	100	0.0000	0.0001	0.025%
	5000	100	5.7297	0.0014	100	0.0000	0.0002	0.024%
	6000	100	5.7298	0.0013	100	0.0000	0.0001	0.023%
	10000	100	5.7294	0.0013	100	0.0001	0.0005	0.023%
	20000	100	5.7293	0.0013	100	0.0001	0.0006	0.022%
	50000	100	5.7292	0.0013	100	0.0001	0.0007	0.022%
	100000	100	5.7292	0.0013	100	0.0001	0.0007	0.022%
LSM with Moment Matching	1000	100	5.7310	0.0026	100	0.0002	0.0011	0.045%

Continued on next page

Table A.3 – continued from previous page

Method	Paths	Stock Price	Point est.	S.E.	Benchmark	Error	RMSE	Relative SE
	2000	100	5.7293	0.0016	100	0.0001	0.0006	0.028%
	3000	100	5.7312	0.0012	100	0.0002	0.0013	0.020%
	4000	100	5.7296	0.0011	100	0.0000	0.0003	0.019%
	5000	100	5.7305	0.0010	100	0.0001	0.0006	0.018%
	6000	100	5.7300	0.0010	100	0.0000	0.0001	0.018%
	10000	100	5.7308	0.0009	100	0.0002	0.0009	0.016%
	20000	100	5.7303	0.0006	100	0.0001	0.0004	0.010%
	50000	100	5.7305	0.0003	100	0.0001	0.0006	0.005%
	100000	100	5.7308	0.0002	100	0.0001	0.0009	0.004%
LSM with Sobol	1000	100	5.7259	0.0019	100	0.0007	0.0040	0.034%
	2000	100	5.7303	0.0010	100	0.0001	0.0004	0.017%
	3000	100	5.7308	0.0012	100	0.0002	0.0009	0.021%
	4000	100	5.7305	0.0008	100	0.0001	0.0006	0.014%
	5000	100	5.7293	0.0006	100	0.0001	0.0006	0.011%
	6000	100	5.7295	0.0007	100	0.0001	0.0004	0.013%
	10000	100	5.7303	0.0006	100	0.0001	0.0004	0.011%
	20000	100	5.7301	0.0003	100	0.0000	0.0002	0.005%
	50000	100	5.7302	0.0002	100	0.0001	0.0003	0.004%
	100000	100	5.7303	0.0002	100	0.0001	0.0004	0.003%
LSM with Sobol and AV	1000	100	5.7313	0.0011	100	0.0002	0.0014	0.019%
	2000	100	5.7302	0.0007	100	0.0001	0.0003	0.012%
	3000	100	5.7315	0.0005	100	0.0003	0.0016	0.010%
	4000	100	5.7304	0.0004	100	0.0001	0.0005	0.007%
	5000	100	5.7294	0.0006	100	0.0001	0.0005	0.010%
	6000	100	5.7302	0.0004	100	0.0001	0.0003	0.007%
	10000	100	5.7303	0.0004	100	0.0001	0.0004	0.007%
	20000	100	5.7305	0.0002	100	0.0001	0.0006	0.004%
	50000	100	5.7301	0.0002	100	0.0000	0.0002	0.003%
	100000	100	5.7303	0.0001	100	0.0001	0.0004	0.001%

Table A.4: Accuracy of LSMC with nonlinear regression in a single dimension

Method	Segments/Spar	Stock Price	Point est.	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with Piecewise Linear	3	100	5.7238	0.0070	5.7299	-0.0011	0.0052	0.122%
	4	100	5.7233	0.0067	5.7299	-0.0011	0.0057	0.117%
	5	100	5.7236	0.0069	5.7299	-0.0011	0.0054	0.121%
	6	100	5.7247	0.0067	5.7299	-0.0009	0.0043	0.117%
	7	100	5.7255	0.0069	5.7299	-0.0008	0.0035	0.120%
	8	100	5.7234	0.0066	5.7299	-0.0011	0.0056	0.115%
	9	100	5.7260	0.0064	5.7299	-0.0007	0.0030	0.111%
	10	100	5.7263	0.0064	5.7299	-0.0006	0.0027	0.112%
	11	100	5.7274	0.0067	5.7299	-0.0004	0.0016	0.117%
	12	100	5.7266	0.0065	5.7299	-0.0006	0.0024	0.114%
	13	100	5.7268	0.0067	5.7299	-0.0005	0.0022	0.116%
	14	100	5.7262	0.0063	5.7299	-0.0006	0.0028	0.109%
	15	100	5.7278	0.0064	5.7299	-0.0004	0.0012	0.111%
	16	100	5.7272	0.0064	5.7299	-0.0005	0.0018	0.111%
	17	100	5.7291	0.0064	5.7299	-0.0001	0.0001	0.111%
	18	100	5.7288	0.0064	5.7299	-0.0002	0.0002	0.111%
	19	100	5.7289	0.0064	5.7299	-0.0002	0.0001	0.112%
	20	100	5.7288	0.0066	5.7299	-0.0002	0.0002	0.115%
	21	100	5.7304	0.0066	5.7299	0.0001	0.0014	0.115%
	22	100	5.7307	0.0065	5.7299	0.0001	0.0017	0.113%
	23	100	5.7295	0.0066	5.7299	-0.0001	0.0005	0.115%
	24	100	5.7299	0.0065	5.7299	0.0000	0.0009	0.113%
	25	100	5.7303	0.0066	5.7299	0.0001	0.0013	0.116%
	26	100	5.7311	0.0065	5.7299	0.0002	0.0021	0.114%
	27	100	5.7294	0.0067	5.7299	-0.0001	0.0004	0.118%
	28	100	5.7301	0.0064	5.7299	0.0000	0.0011	0.112%
	29	100	5.7319	0.0066	5.7299	0.0004	0.0029	0.115%
	30	100	5.7323	0.0064	5.7299	0.0004	0.0033	0.113%
LSM with Smoothing Spline	0.033	100	5.7831	0.0065	5.7299	0.0093	0.0541	0.113%
	0.067	100	5.7832	0.0066	5.7299	0.0093	0.0542	0.114%
	0.100	100	5.7827	0.0065	5.7299	0.0092	0.0537	0.113%
	0.133	100	5.7825	0.0065	5.7299	0.0092	0.0535	0.112%
	0.167	100	5.7826	0.0065	5.7299	0.0092	0.0536	0.113%
	0.200	100	5.7819	0.0065	5.7299	0.0091	0.0529	0.112%
	0.233	100	5.7806	0.0065	5.7299	0.0088	0.0516	0.112%
	0.267	100	5.7799	0.0065	5.7299	0.0087	0.0509	0.113%
	0.300	100	5.7775	0.0066	5.7299	0.0083	0.0485	0.114%
	0.333	100	5.7750	0.0065	5.7299	0.0079	0.0460	0.113%
	0.367	100	5.7729	0.0065	5.7299	0.0075	0.0439	0.113%
	0.400	100	5.7688	0.0065	5.7299	0.0068	0.0398	0.113%
	0.433	100	5.7675	0.0066	5.7299	0.0066	0.0385	0.114%
	0.467	100	5.7633	0.0065	5.7299	0.0058	0.0343	0.113%
	0.500	100	5.7578	0.0067	5.7299	0.0049	0.0288	0.117%

Continued on next page

Table A.4 – continued from previous page

Method	Segments/Spar	Stock Price	Point est.	S.E.	Benchmark	Error	RMSE	Relative SE
	0.533	100	5.7540	0.0070	5.7299	0.0042	0.0250	0.122%
	0.567	100	5.7495	0.0070	5.7299	0.0034	0.0205	0.121%
	0.600	100	5.7459	0.0067	5.7299	0.0028	0.0169	0.117%
	0.633	100	5.7435	0.0067	5.7299	0.0024	0.0145	0.117%
	0.667	100	5.7409	0.0067	5.7299	0.0019	0.0119	0.117%
	0.700	100	5.7380	0.0069	5.7299	0.0014	0.0090	0.120%
	0.733	100	5.7356	0.0069	5.7299	0.0010	0.0066	0.120%
	0.767	100	5.7348	0.0071	5.7299	0.0008	0.0058	0.123%
	0.800	100	5.7345	0.0072	5.7299	0.0008	0.0055	0.125%
	0.833	100	5.7328	0.0067	5.7299	0.0005	0.0038	0.117%
	0.867	100	5.7311	0.0066	5.7299	0.0002	0.0021	0.114%
	0.900	100	5.7290	0.0064	5.7299	-0.0002	0.0000	0.111%
	0.933	100	5.7280	0.0065	5.7299	-0.0003	0.0010	0.114%
	0.967	100	5.7277	0.0065	5.7299	-0.0004	0.0013	0.114%
	1.000	100	5.7273	0.0066	5.7299	-0.0005	0.0017	0.116%

Table A.5: Accuracy of LSMC with different regression polynomials in a single dimension

Method	Degrees	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with polynomial regression	1	100	5.7119	0.0069	5.7299	-0.0031	0.0171	0.121%
	2	100	5.7226	0.0064	5.7299	-0.0013	0.0064	0.111%
	3	100	5.7234	0.0067	5.7299	-0.0011	0.0056	0.118%
	4	100	5.7237	0.0069	5.7299	-0.0011	0.0053	0.120%
	5	100	5.7250	0.0068	5.7299	-0.0009	0.0040	0.119%
	6	100	5.7247	0.0066	5.7299	-0.0009	0.0043	0.115%
	7	100	5.7245	0.0068	5.7299	-0.0009	0.0045	0.119%
	8	100	5.7248	0.0068	5.7299	-0.0009	0.0042	0.119%
	9	100	5.7245	0.0067	5.7299	-0.0009	0.0045	0.116%
	10	100	5.7242	0.0067	5.7299	-0.0010	0.0048	0.117%
	11	100	5.7251	0.0066	5.7299	-0.0008	0.0039	0.115%
	12	100	5.7250	0.0065	5.7299	-0.0009	0.0040	0.113%
	13	100	5.7253	0.0064	5.7299	-0.0008	0.0037	0.112%
	14	100	5.7249	0.0064	5.7299	-0.0009	0.0041	0.112%
	15	100	5.7248	0.0064	5.7299	-0.0009	0.0042	0.112%
LSM with Laguerre	3	100	5.7226	0.0064	5.7299	-0.0013	0.0064	0.111%
	4	100	5.7234	0.0067	5.7299	-0.0011	0.0056	0.118%
	5	100	5.7237	0.0069	5.7299	-0.0011	0.0053	0.120%
	6	100	5.7250	0.0068	5.7299	-0.0009	0.0040	0.119%
	7	100	5.7247	0.0066	5.7299	-0.0009	0.0043	0.115%
	8	100	5.7248	0.0068	5.7299	-0.0009	0.0042	0.119%
	9	100	5.7248	0.0068	5.7299	-0.0009	0.0042	0.119%

Continued on next page

Table A.5 – continued from previous page

Method	Degrees	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
	10	100	5.7243	0.0067	5.7299	-0.0010	0.0047	0.117%
	11	100	5.7243	0.0067	5.7299	-0.0010	0.0047	0.117%
	12	100	5.7251	0.0064	5.7299	-0.0008	0.0039	0.112%
LSM with Hermite	3	100	5.7226	0.0064	5.7299	-0.0013	0.0064	0.111%
	4	100	5.7234	0.0067	5.7299	-0.0011	0.0056	0.118%
	5	100	5.7237	0.0069	5.7299	-0.0011	0.0053	0.120%
	6	100	5.7250	0.0068	5.7299	-0.0009	0.0040	0.119%
	7	100	5.7247	0.0066	5.7299	-0.0009	0.0043	0.115%
	8	100	5.7245	0.0068	5.7299	-0.0009	0.0045	0.119%
	9	100	5.7248	0.0068	5.7299	-0.0009	0.0042	0.119%
	10	100	5.7245	0.0067	5.7299	-0.0009	0.0045	0.116%
	11	100	5.7242	0.0067	5.7299	-0.0010	0.0048	0.117%
	12	100	5.7251	0.0066	5.7299	-0.0008	0.0039	0.115%
LSM with Legendre	3	100	5.7226	0.0064	5.7299	-0.0013	0.0064	0.111%
	4	100	5.7234	0.0067	5.7299	-0.0011	0.0056	0.118%
	5	100	5.7237	0.0069	5.7299	-0.0011	0.0053	0.120%
	6	100	5.7250	0.0068	5.7299	-0.0009	0.0040	0.119%
	7	100	5.7247	0.0066	5.7299	-0.0009	0.0043	0.115%
	8	100	5.7245	0.0068	5.7299	-0.0009	0.0045	0.119%
	9	100	5.7248	0.0068	5.7299	-0.0009	0.0042	0.119%
	10	100	5.7245	0.0067	5.7299	-0.0009	0.0045	0.116%
	11	100	5.7242	0.0067	5.7299	-0.0010	0.0048	0.117%
	12	100	5.7251	0.0066	5.7299	-0.0008	0.0039	0.115%
LSM with Chebyshev	3	100	5.7226	0.0064	5.7299	-0.0013	0.0064	0.111%
	4	100	5.7234	0.0067	5.7299	-0.0011	0.0056	0.118%
	5	100	5.7237	0.0069	5.7299	-0.0011	0.0053	0.120%
	6	100	5.7250	0.0068	5.7299	-0.0009	0.0040	0.119%
	7	100	5.7247	0.0066	5.7299	-0.0009	0.0043	0.115%
	8	100	5.7246	0.0068	5.7299	-0.0009	0.0044	0.119%
	9	100	5.7248	0.0068	5.7299	-0.0009	0.0042	0.119%
	10	100	5.7245	0.0067	5.7299	-0.0009	0.0045	0.116%
	11	100	5.7242	0.0067	5.7299	-0.0010	0.0048	0.117%
	12	100	5.7251	0.0066	5.7299	-0.0008	0.0039	0.115%

Table A.6: Comparison of LSMC with and without control variates

Method	Branches	Stock Price	Point est.	S.E.	Benchmark	Error Error	RMSE	Relative SE
LSM with control	**5000**	70	0.1211	0.0001	0.1212	-0.0008	0.0001	0.077%
		80	0.6700	0.0003	0.6699	0.0001	0.0000	0.049%
		90	2.3036	0.0006	2.3030	0.0003	0.0006	0.024%

Continued on next page

Table A.6 – continued from previous page

		100	5.7301	0.0010	5.7299	0.0000	0.0011	0.017%
		110	11.3396	0.0015	11.3410	-0.0001	0.0014	0.014%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%
	20000	70	0.1212	0.0001	0.1212	0.0002	0.0002	0.046%
		80	0.6698	0.0001	0.6699	-0.0002	0.0002	0.017%
		90	2.3029	0.0003	2.3030	0.0000	0.0001	0.011%
		100	5.7288	0.0005	5.7299	-0.0002	0.0002	0.009%
		110	11.3397	0.0006	11.3410	-0.0001	0.0013	0.005%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%
LSM without control	**5000**	70	0.1203	0.0017	0.1212	-0.0076	0.0007	1.433%
		80	0.6709	0.0050	0.6699	0.0016	0.0009	0.751%
		90	2.3053	0.0111	2.3030	0.0010	0.0023	0.483%
		100	5.7252	0.0156	5.7299	-0.0008	0.0038	0.273%
		110	11.3286	0.0105	11.3410	-0.0011	0.0124	0.093%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%
	20000	70	0.1197	0.0008	0.1212	-0.0122	0.0013	0.709%
		80	0.6667	0.0026	0.6699	-0.0047	0.0033	0.388%
		90	2.2944	0.0051	2.3030	-0.0037	0.0086	0.223%
		100	5.7229	0.0068	5.7299	-0.0012	0.0061	0.119%
		110	11.3279	0.0061	11.3410	-0.0012	0.0131	0.054%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%

Table A.7: Comparison of LSMC with restricted and unrestricted sample paths

Method	**Sample Paths**	**Stock Price**	**Estimation**	**S.E.**	**Benchmark**	**Error**	**RMSE**	**Relative SE**
LSM with control	**Unrestricted**	70	0.1212	0.0001	0.1212	0.0002	0.0002	0.046%
		80	0.6693	0.0001	0.6699	-0.0009	0.0007	0.019%
		90	2.2888	0.0002	2.3030	-0.0062	0.0142	0.010%
		100	5.6086	0.0004	5.7299	-0.0212	0.1204	0.007%
		110	10.8598	0.0005	11.3410	-0.0424	0.4812	0.004%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%
	Restricted	70	0.1212	0.0001	0.1212	0.0002	0.0002	0.046%
		80	0.6698	0.0001	0.6699	-0.0002	0.0002	0.017%
		90	2.3029	0.0003	2.3030	0.0000	0.0001	0.011%
		100	5.7288	0.0005	5.7299	-0.0002	0.0002	0.009%
		110	11.3397	0.0006	11.3410	-0.0001	0.0013	0.005%
		120	20.0000	0.0000	20.0000	0.0000	0.0000	0.000%

A.2 SIMULATION RESULTS FOR BERMUDAN CALLS ON MULTIPLE ASSETS

Table A.8: Accuracy of LSMC with different regression polynomials for pricing a Bermudan call on the geometric average of five assets

Method	Degrees	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with polynomial regression	1	100	3.0419	0.0013	3.0495	0.0025	0.0076	0.043%
	2	100	3.0478	0.0009	3.0495	0.0006	0.0017	0.031%
	3	100	3.0478	0.0009	3.0495	0.0005	0.0017	0.029%
	4	100	3.0477	0.0009	3.0495	0.0006	0.0018	0.031%
	5	100	3.0478	0.0010	3.0495	0.0006	0.0017	0.031%
	6	100	3.0476	0.0009	3.0495	0.0006	0.0019	0.031%
	7	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.032%
	8	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.033%
	9	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	10	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	11	100	3.0473	0.0009	3.0495	0.0007	0.0022	0.031%
	12	100	3.0472	0.0009	3.0495	0.0008	0.0023	0.030%
LSM with Laguerre	3	100	3.0478	0.0009	3.0495	0.0006	0.0017	0.031%
	4	100	3.0478	0.0009	3.0495	0.0005	0.0017	0.029%
	5	100	3.0477	0.0009	3.0495	0.0006	0.0018	0.031%
	6	100	3.0478	0.0010	3.0495	0.0006	0.0017	0.031%
	7	100	3.0477	0.0009	3.0495	0.0006	0.0018	0.030%
	8	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.032%
	9	100	3.0474	0.0010	3.0495	0.0007	0.0021	0.032%
	10	100	3.0474	0.0010	3.0495	0.0007	0.0021	0.031%
	11	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	12	100	3.0472	0.0009	3.0495	0.0008	0.0023	0.031%
LSM with Hermite	3	100	3.0478	0.0009	3.0495	0.0006	0.0017	0.031%
	4	100	3.0478	0.0009	3.0495	0.0005	0.0017	0.029%
	5	100	3.0477	0.0009	3.0495	0.0006	0.0018	0.031%
	6	100	3.0478	0.0010	3.0495	0.0006	0.0017	0.031%
	7	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.032%
	8	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.032%
	9	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.033%
	10	100	3.0474	0.0010	3.0495	0.0007	0.0021	0.032%
	11	100	3.0474	0.0010	3.0495	0.0007	0.0021	0.031%
	12	100	3.0473	0.0009	3.0495	0.0007	0.0022	0.031%
LSM with Legendre	3	100	3.0478	0.0009	3.0495	0.0006	0.0017	0.031%
	4	100	3.0478	0.0009	3.0495	0.0005	0.0017	0.029%
	5	100	3.0477	0.0009	3.0495	0.0006	0.0018	0.031%
	6	100	3.0478	0.0010	3.0495	0.0006	0.0017	0.031%
	7	100	3.0475	0.0009	3.0495	0.0007	0.0020	0.031%

Continued on next page

Table A.8 – continued from previous page

Method	Degrees	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
	8	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.032%
	9	100	3.0474	0.0010	3.0495	0.0007	0.0021	0.033%
	10	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	11	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	12	100	3.0473	0.0009	3.0495	0.0007	0.0022	0.031%
LSM with Chebysheve	3	100	3.0478	0.0009	3.0495	0.0006	0.0017	0.031%
	4	100	3.0478	0.0009	3.0495	0.0005	0.0017	0.029%
	5	100	3.0477	0.0009	3.0495	0.0006	0.0018	0.031%
	6	100	3.0478	0.0010	3.0495	0.0006	0.0017	0.031%
	7	100	3.0477	0.0009	3.0495	0.0006	0.0018	0.031%
	8	100	3.0475	0.0010	3.0495	0.0007	0.0020	0.032%
	9	100	3.0474	0.0010	3.0495	0.0007	0.0021	0.032%
	10	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	11	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	12	100	3.0473	0.0009	3.0495	0.0007	0.0022	0.031%

Table A.9: Comparison of the random tree and LSMC for a Bermudan max-call option

Method	Paths	Stock Price	Low est.	S.E. Low est	High est.	S.E. Low est	Confidence Interval	Point est.	Benchmark	Error	RMSE	Relative SE
Tree Method	10	70	0.2087	0.0078	0.2139	0.0079	[0.1958,0.2269]	0.2113	0.2370	0.1084	0.0463	3.751%
		80	1.1262	0.0179	1.1711	0.0194	[1.0966,1.2031]	1.1487	1.2590	0.0876	0.1433	1.693%
		90	3.9808	0.0305	4.2067	0.0311	[3.9306,4.2578]	4.0938	4.0770	0.0041	0.1511	0.759%
		100	9.3016	0.0664	9.9218	0.0702	[9.1924,10.0373]	9.6117	9.3610	0.0268	0.4172	0.731%
		110	16.8377	0.1026	18.0734	0.0874	[16.6689,18.2172]	17.4556	16.9240	0.0314	0.7042	0.501%
		120	23.6829	0.1685	27.7125	0.1013	[23.4057,27.8791]	25.6977	25.9800	0.0109	0.6972	0.394%
	20	70	0.2405	0.0037	0.2433	0.0038	[0.2343,0.2494]	0.1210	0.2370	0.9993	0.1214	3.107%
		80	1.3131	0.0077	1.3376	0.0077	[1.3005,1.3503]	0.6679	1.2590	0.9781	0.6557	1.160%
		90	4.2688	0.0216	4.3720	0.0222	[4.2332,4.4085]	2.3140	4.0770	0.8752	2.0174	0.960%
		100	9.8643	0.0424	10.1470	0.0431	[9.7945,10.2179]	5.7740	9.3610	0.7459	4.2766	0.746%
		110	17.7144	0.0348	18.2585	0.0330	[17.6572,18.3127]	11.3900	16.9240	0.5860	6.6462	0.289%
		120	26.9297	0.0493	27.7873	0.0514	[26.8486,27.8717]	20.0700	25.9800	0.3679	7.3601	0.256%
LSMC	5000	70			0.2363	0.0001			0.2370	0.0029	0.0007	0.048%
		80			1.2582	0.0004			1.2590	0.0006	0.0008	0.035%
		90			4.0760	0.0008			4.0770	0.0003	0.0010	0.020%
		100			9.3587	0.0016			9.3610	0.0002	0.0023	0.018%
		110			16.9253	0.0028			16.9240	0.0001	0.0013	0.017%
		120			25.9795	0.0042			25.9800	0.0000	0.0005	0.016%
	20000	70			0.2365	0.0001			0.2370	0.0020	0.0005	0.028%
		80			1.2591	0.0002			1.2590	0.0001	0.0001	0.013%
		90			4.0771	0.0004			4.0770	0.0000	0.0001	0.009%
		100			9.3596	0.0007			9.3610	0.0002	0.0014	0.007%
		110			16.9227	0.0011			16.9240	0.0001	0.0013	0.006%
		120			25.9769	0.0017			25.9800	0.0001	0.0031	0.007%

Table A.10: Accuracy of LSMC with nonlinear regression for pricing a Bermudan call on the geometric average of five assets

Method	Segments/Spar	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with Piecewise Linear	3	100	3.0478	0.0009	3.0495	0.0006	0.0017	0.029%
	4	100	3.0474	0.0009	3.0495	0.0007	0.0021	0.031%
	5	100	3.0468	0.0010	3.0495	0.0009	0.0027	0.031%
	6	100	3.0469	0.0009	3.0495	0.0009	0.0026	0.030%
	7	100	3.0462	0.0011	3.0495	0.0011	0.0033	0.037%
	8	100	3.0461	0.0008	3.0495	0.0011	0.0034	0.027%
	9	100	3.0454	0.0011	3.0495	0.0014	0.0041	0.035%
	10	100	3.0444	0.0015	3.0495	0.0017	0.0051	0.048%
	11	100	3.0436	0.0017	3.0495	0.0019	0.0059	0.056%
	12	100	3.0437	0.0015	3.0495	0.0019	0.0058	0.048%
	13	100	3.0439	0.0011	3.0495	0.0018	0.0056	0.038%
	14	100	3.0429	0.0014	3.0495	0.0022	0.0066	0.047%
	15	100	3.0419	0.0017	3.0495	0.0025	0.0076	0.055%
	16	100	3.0408	0.0017	3.0495	0.0028	0.0087	0.054%
	17	100	3.0400	0.0021	3.0495	0.0031	0.0095	0.070%
	18	100	3.0393	0.0019	3.0495	0.0033	0.0102	0.063%
	19	100	3.0374	0.0023	3.0495	0.0040	0.0121	0.077%
	20	100	3.0381	0.0020	3.0495	0.0038	0.0114	0.067%
	21	100	3.0377	0.0021	3.0495	0.0039	0.0118	0.068%
	22	100	3.0373	0.0022	3.0495	0.0040	0.0122	0.074%
	23	100	3.0372	0.0022	3.0495	0.0040	0.0123	0.073%
	24	100	3.0344	0.0027	3.0495	0.0050	0.0151	0.091%
	25	100	3.0360	0.0023	3.0495	0.0044	0.0135	0.077%
	26	100	3.0344	0.0025	3.0495	0.0049	0.0151	0.083%
	27	100	3.0344	0.0024	3.0495	0.0049	0.0151	0.080%
	28	100	3.0342	0.0025	3.0495	0.0050	0.0153	0.082%
	29	100	3.0346	0.0025	3.0495	0.0049	0.0149	0.083%
	30	100	3.0334	0.0023	3.0495	0.0053	0.0161	0.074%
LSM with Smoothing Spline	0.100	100	2.9948	0.0021	3.0495	0.0179	0.0547	0.071%
	0.133	100	2.9960	0.0022	3.0495	0.0175	0.0535	0.072%
	0.167	100	2.9976	0.0022	3.0495	0.0170	0.0519	0.072%
	0.200	100	2.9991	0.0022	3.0495	0.0165	0.0504	0.073%
	0.233	100	3.0008	0.0022	3.0495	0.0160	0.0487	0.074%
	0.267	100	3.0025	0.0023	3.0495	0.0154	0.0470	0.075%
	0.300	100	3.0042	0.0023	3.0495	0.0148	0.0453	0.075%
	0.333	100	3.0063	0.0023	3.0495	0.0142	0.0432	0.075%
	0.367	100	3.0082	0.0023	3.0495	0.0136	0.0413	0.077%
	0.400	100	3.0107	0.0024	3.0495	0.0127	0.0388	0.078%
	0.433	100	3.0135	0.0025	3.0495	0.0118	0.0360	0.083%
	0.467	100	3.0159	0.0026	3.0495	0.0110	0.0336	0.085%
	0.500	100	3.0185	0.0027	3.0495	0.0102	0.0310	0.088%

Continued on next page

Table A.10 – continued from previous page

Method	Segments/Spar	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
	0.533	100	3.0207	0.0027	3.0495	0.0095	0.0288	0.088%
	0.567	100	3.0228	0.0027	3.0495	0.0088	0.0267	0.090%
	0.600	100	3.0252	0.0026	3.0495	0.0080	0.0243	0.085%
	0.633	100	3.0277	0.0025	3.0495	0.0071	0.0218	0.083%
	0.667	100	3.0304	0.0025	3.0495	0.0063	0.0191	0.082%
	0.700	100	3.0330	0.0026	3.0495	0.0054	0.0165	0.085%
	0.733	100	3.0347	0.0026	3.0495	0.0049	0.0148	0.086%
	0.767	100	3.0358	0.0025	3.0495	0.0045	0.0137	0.083%
	0.800	100	3.0370	0.0025	3.0495	0.0041	0.0125	0.081%
	0.833	100	3.0386	0.0023	3.0495	0.0036	0.0109	0.074%
	0.867	100	3.0403	0.0021	3.0495	0.0030	0.0092	0.070%
	0.900	100	3.0419	0.0019	3.0495	0.0025	0.0076	0.064%
	0.933	100	3.0430	0.0018	3.0495	0.0021	0.0065	0.059%
	0.967	100	3.0439	0.0016	3.0495	0.0018	0.0056	0.054%
	1.000	100	3.0445	0.0015	3.0495	0.0016	0.0050	0.049%

Table A.11: Convergence of LSMC with low-discrepancy sequences for pricing a Bermudan call on the geometric average of five assets

Method	Paths	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with Halton and AV	1000	100	3.0516	0.0020	3.0495	0.0007	0.0021	0.067%
	2000	100	3.0486	0.0015	3.0495	0.0003	0.0009	0.050%
	3000	100	3.0523	0.0010	3.0495	0.0009	0.0028	0.034%
	4000	100	3.0492	0.0014	3.0495	0.0001	0.0003	0.045%
	5000	100	3.0499	0.0009	3.0495	0.0001	0.0004	0.029%
	6000	100	3.0499	0.0010	3.0495	0.0001	0.0004	0.034%
	10000	100	3.0484	0.0008	3.0495	0.0003	0.0011	0.025%
	20000	100	3.0502	0.0006	3.0495	0.0002	0.0007	0.021%
	50000	100	3.0495	0.0003	3.0495	0.0000	0.0000	0.010%
	100000	100	3.0498	0.0003	3.0495	0.0001	0.0003	0.009%
LSM with Pseudo and AV	1000	100	3.0466	0.0023	3.0495	0.0009	0.0029	0.076%
	2000	100	3.0522	0.0021	3.0495	0.0009	0.0027	0.069%
	3000	100	3.0511	0.0011	3.0495	0.0005	0.0016	0.035%
	4000	100	3.0495	0.0011	3.0495	0.0000	0.0000	0.038%
	5000	100	3.0485	0.0013	3.0495	0.0003	0.0010	0.043%
	6000	100	3.0502	0.0008	3.0495	0.0002	0.0007	0.026%
	10000	100	3.0490	0.0008	3.0495	0.0002	0.0005	0.027%
	20000	100	3.0496	0.0004	3.0495	0.0000	0.0001	0.013%
	50000	100	3.0498	0.0004	3.0495	0.0001	0.0003	0.012%
	100000	100	3.0502	0.0002	3.0495	0.0002	0.0007	0.008%

Continued on next page

Table A.11 – continued from previous page

Method	Paths	Stock Price	Estimation	S.E.	Benchmark	Error	RMSE	Relative SE
LSM with Moment Matching and AV	1000	100	3.0454	0.0021	3.0495	0.0014	0.0041	0.069%
	2000	100	3.0486	0.0020	3.0495	0.0003	0.0009	0.065%
	3000	100	3.0507	0.0015	3.0495	0.0004	0.0012	0.048%
	4000	100	3.0498	0.0013	3.0495	0.0001	0.0003	0.043%
	5000	100	3.0479	0.0009	3.0495	0.0005	0.0016	0.030%
	6000	100	3.0511	0.0010	3.0495	0.0005	0.0016	0.033%
	10000	100	3.0504	0.0009	3.0495	0.0003	0.0009	0.030%
	20000	100	3.0495	0.0005	3.0495	0.0000	0.0000	0.017%
	50000	100	3.0501	0.0004	3.0495	0.0002	0.0006	0.014%
	100000	100	3.0499	0.0002	3.0495	0.0001	0.0004	0.008%
LSM with Sobol and AV	1000	100	3.0512	0.0024	3.0495	0.0006	0.0017	0.080%
	2000	100	3.0481	0.0013	3.0495	0.0004	0.0014	0.043%
	3000	100	3.0507	0.0014	3.0495	0.0004	0.0012	0.047%
	4000	100	3.0520	0.0011	3.0495	0.0008	0.0025	0.037%
	5000	100	3.0485	0.0012	3.0495	0.0003	0.0010	0.038%
	6000	100	3.0522	0.0008	3.0495	0.0009	0.0027	0.027%
	10000	100	3.0486	0.0007	3.0495	0.0003	0.0009	0.024%
	20000	100	3.0492	0.0004	3.0495	0.0001	0.0003	0.014%
	50000	100	3.0494	0.0003	3.0495	0.0000	0.0001	0.009%
	100000	100	3.0495	0.0003	3.0495	0.0000	0.0000	0.009%

Table A.12: Convergence of the Bermuda to American for pricing a Bermudan call on the geometric average of five assets

Paths	**StockPrice**	**Estimation**	**S.E.**	**True Value**	**RMSE**	**Relative SE**	**Time (Seconds)**
5000	100	2.5674	0.0000	2.5655	0.0019	0.074%	1.89
	100	2.9474	0.0009	2.9351	0.0123	0.418%	2.35
	100	3.0485	0.0013	3.0495	0.0010	0.032%	2.49
	100	3.1025	0.0012	3.1029	0.0004	0.011%	3.33
	100	3.1321	0.0012	3.1339	0.0018	0.057%	5.02
	100	3.1948	0.0013	3.1946	0.0002	0.008%	8.01
	100	3.2282	0.0012	3.2269	0.0013	0.041%	16.67
	100	3.2482	0.0013	3.2470	0.0012	0.037%	42.30
	100	3.2533	0.0010	3.2538	0.0005	0.016%	85.65
	100	3.2574	0.0012	3.2573	0.0001	0.002%	173.22
	100	3.2585	0.0010	3.2593	0.0008	0.026%	434.34
10000	100	3.2594	0.0011	3.2601	0.0007	0.022%	880.95
	100	2.5674	0.0000	2.5655	0.0019	0.074%	3.56
	100	2.9478	0.0006	2.9351	0.0127	0.434%	4.42
	100	3.0490	0.0008	3.0495	0.0005	0.018%	5.06
	100	3.1022	0.0010	3.1029	0.0007	0.023%	6.78
	100	3.1341	0.0008	3.1339	0.0002	0.006%	8.55
	100	3.1944	0.0007	3.1946	0.0002	0.007%	17.14
	100	3.2262	0.0008	3.2269	0.0007	0.023%	32.49
	100	3.2458	0.0009	3.2470	0.0012	0.037%	84.29
	100	3.2536	0.0008	3.2538	0.0002	0.006%	158.82
	100	3.2570	0.0008	3.2573	0.0003	0.009%	307.99
	100	3.2588	0.0007	3.2593	0.0005	0.016%	797.06
	100	3.2605	0.0009	3.2601	0.0004	0.012%	1611.54

B APPENDIX: PRICING ALGORITHMS

Algorithms presented in this appendix are written in R 3.4.1 (R Core Team (2017)), and R packages (Analytics and Weston (2015), Azzalini and Genz (2016), Christophe and Petr (2015), Izrailev (2014), Team et al. (2015), Warnes, Bolker, and Lumley (2015), and Wickham (2009)).

B.1 PRICING ALGORITHMS ON A SINGLE ASSET

Algorithm 1: Pricing Bermudan option on a single asset using stochastic tree

```
library("fOptions");
library("tictoc");
library("foreach"); #optional: enable parallel computation

K = 100;              # strike price
delta = 0.1;          # dividend Yield
mT= 1;                # time to maturity
sigma = 0.2;          # asset annual volatility
t = c(0,1/3,2/3,1); # exercise opportunities for the Bermudan option
tstep=1/3;
r = 0.05;             # risk-free rate
b = 100;              # branching parameter
iteration=100; # number of Monte Carlo iterations
repitition=10; # number of Monte Carlo estimators

#pricing initialisation
S = c(70,80,90,100,110,120); # asset prices
L = length(S)
emptyr=matrix(NA, nrow=repitition , ncol=L);
Ehighest     =    emptyr; # option price high estimator
Ehighestc    =    emptyr; # option price high estimator using control variate
Elowest      =    emptyr; # option price low estimator
Elowestc     =    emptyr; # option price low estimator using control variate
pointestimateE= emptyr; # option price point estimator
pointestimateEc=emptyr; # option price point estimator using control variate
empty1=matrix(NA, nrow=1, ncol=L);
# define expected estimators
pointestimator= empty1;
highestimator=empty1;
lowestimator= empty1;
pointestimatorc=empty1;
highestimatorc= empty1;
lowestimatorc=empty1;
# define standard errors
sep=empty1;
```

```
sel=empty1;
seh=empty1;
sepc= empty1;
selc= empty1;
sehc= empty1;
# define confidence intervals
CILow   = empty1;
CIHigh  = empty1;
CILowc  = empty1;
CIHighc= empty1;

#create stohastic tree
createtree=function(S,q,tstep,r,delta,sigma){
  Spath=matrix(NA, nrow=b^(1/tstep-1), ncol=(1/tstep-1));
  Scontrol=matrix(NA, nrow=b^(1/tstep-1), ncol=(1/tstep-1));
  for (p in 1:(1/tstep-1))
  {
    for (j in 1:b^(p-1))
    {
      for (i in 1:(b/2))
      {
        if (p-1==0){
          Sp=S[q];
        }else{
          Sp=Spath[j,(p-1)];
        }
        x=rnorm(1, mean=0, sd=1);
        Spath[b*(j-1)+i,p]=Sp*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*x);
        Spath[b*(j-1)+i+b/2,p]=Sp*exp((r-delta-sigma^2/2)*tstep-sigma*sqrt(tstep)*x);
      }
    }
  }
  Scontrol=Spath;
  return(cbind(Spath,Scontrol));
}

# option pricing
foreach (q=  1:length(S)) %dopar%
{
  for (M in 1:repitition)
  {
    tic();
    lowestc=matrix(NA, nrow=iteration, ncol=1);
    lowest=matrix(NA, nrow=iteration, ncol=1);
    highestc=matrix(NA, nrow=iteration, ncol=1);
    highest=matrix(NA, nrow=iteration, ncol=1);
    for (f in 1:iteration)
    {
      Tree=createtree(S,q,tstep,r,delta,sigma);
      Spath=Tree[,1:(ncol(Tree)/2)];
      Scontrol=Tree[,(ncol(Tree)/2+1):ncol(Tree)];
    # high estimator
    End=(1/tstep-1);
    est=matrix(NA, nrow=b^End, ncol=End);
    estcontrol=est;
    EUVlaue=  GBSOption('c',as.numeric(Spath[,End]),K,tstep,r,(r-delta),sigma)@price;
    est[,End]=pmax(Spath[,End]-K,EUVlaue);
    estcontrol[,End]=pmax(Spath[,End]-K,0);
      for (Step in (1/tstep-2):1)
      {
        index=seq(1,b^(Step+1),b);
        for (i in 1:(b^Step))
        {
          continue=(sum(est[index[i]:(index[i]+b-1),Step+1])*exp(-r*tstep))/b;
        est[i,Step]=max(Spath[i,Step]-K,continue);
        estcontrol[i,Step]=max(mean(estcontrol[index[i]:(index[i]+b-1),Step+1]),0)*
                           exp(-r*tstep);
      }
    }
    contnc=mean(est[,1],na.rm=  TRUE)*exp(-r*tstep);
    highest[f]=max(S[q]-K,contnc);
  # high estimator with control
    control=GBSOption('c',S[q],K,(mT-tstep),r,(r-delta),sigma)@price;
    contc=mean(estcontrol[,1],na.rm=TRUE)*exp(-r*tstep);
    highestc[f]=highest[f]+(control-contc);
  # low estimator
    estL=est;
    estcontrolL=matrix(NA, nrow=b^End, ncol=End);
    estL[,End]=pmax(Spath[,End]-K,EUVlaue);
    for (Step in (1/tstep-1):1)
    {
      index=seq(1,b^(Step),b);
      for (i in 1:(b^(Step-1)))
      {
        tempest=0;
        for (j in 1:(b/2))
        {
         substract=estL[index[i]+(j-1),Step]+estL[index[i]+(j-1)+b/2,Step]
         continueL=(sum(estL[index[i]:(index[i]+b-1),Step])-substract)*exp(-r*tstep)/b;
         spot=ifelse(Step-1==0,S[q],Spath[i,Step-1]);
         tempest=c(tempest,ifelse((spot-K)<continueL,substract*exp(-r*tstep)/2,
                             max(spot-K,0)));
        }
        if (Step!=1){
           estL[i,Step-1]=mean(tempest[2:length(tempest)]);
        }else{
           lowest[f]=mean(tempest[2:length(tempest)]);
        }
        }
      }
    #low estimator with control
    lowestc[f]=lowest[f]+(control-contc);
  }
  #calculated the expected estimators
  Ehighest[M,q]=mean(highest);
  Elowest[M,q]=mean(lowest);
  Ehighestc[M,q]=mean(highestc);
  Elowestc[M,q]=mean(lowestc);
  pointestimateE[M,q]= (max(S[q]-K,Elowest[M,q])+Ehighest[M,q])/2;
  pointestimateEc[M,q]= (max(S[q]-K,Elowestc[M,q])+Ehighestc[M,q])/2;
  toc();
}
#preparing the outputs
lowestimator[q]=mean(Elowest[,q]);
highestimator[q]=mean(Ehighest[,q]);
pointestimator[q]= mean(pointestimateE[,q]);

lowestimatorc[q]=mean(Elowestc[,q]);
highestimatorc[q]=mean(Ehighestc[,q]);
pointestimatorc[q]= mean(pointestimateEc[,q]);

sep[q]=sd(pointestimateE[,q])/sqrt(repitition);
sel[q]=sd(Elowest[,q])/sqrt(repitition);
seh[q]=sd(Ehighest[,q])/sqrt(repitition);

sepc[q]=sd(pointestimateEc[,q])/sqrt(repitition);
selc[q]=sd(Elowest[,q])/sqrt(repitition);
sehc[q]=sd(Ehighestc[,q])/sqrt(repitition);
```

```

  CILow[q]= lowestimator[q] - qnorm(0.95)*sel[q];
  CIHigh[q]= highestimator[q]+ qnorm(0.95)*seh[q];

  CILowc[q]= lowestimator[q] - qnorm(0.95)*sel[q];
  CIHighc[q]= highestimator[q]+ qnorm(0.95)*seh[q];
}
#output
Output=data.frame(StockPrice=S,
                  Lowest=t(lowestimator),
                  StderrL=t(sel),
                  Highest=t(highestimator),
                  StderrH=t(seh),
                  Pointest=t(pointestimator),
                  ConfidenceILow=t(CILow),
                  ConfidentceHigh=t(CIHigh),
                  Pointest=t(pointestimator),
                  )
Outputc=data.frame(StockPrice=S,
                  Lowest=t(lowestimatorc),
                  StderrL=t(selc),
                  Highest=t(highestimatorc),
                  StderrH=t(sehc),
                  Pointest=t(pointestimatorc),
                  ConfidenceILow=t(CILowc),
                  ConfidentceHigh=t(CIHighc),
                  Pointest=t(pointestimatorc),
                  )
print(Output)   #Bemudan option prices
print(Outputc)  #Bemudan option prices using control variate
```

Algorithm 2: Pricing Bermudan option on a single asset using stochastic mesh

```
library("fOptions");
library("tictoc");
library("foreach"); #enable parallel computation

K = 100;            # strike price
delta = 0.1;        # dividend Yield
mT= 1;              # time to maturity
sigma = 0.2;        # asset annual volatility
t = c(0,1/3,2/3,1); # exercise opportunities for the Bermudan option
tstep=1/3;
r = 0.05;           # risk-free rate
b = 20;             # branching parameter
n=100;              # number of Monte Carlo iterations
g=25;               # number of Monte Carlo estimators

#pricing initialisation
S=c(70,80,90,100,110,120); # asset prices
L=length(S);

lowestimator1=matrix(NA, nrow=g, ncol=L);       # option price low estimator
highestimator=matrix(NA, nrow=g, ncol=L);      # option price high estimator
pointestimateouter=matrix(NA, nrow=g, ncol=L); # option price point estimator
# define expected estimators
estimator1=matrix(NA, nrow=1, ncol=L);
highest=matrix(NA, nrow=1, ncol=L);
lowest=matrix(NA, nrow=1, ncol=L);
# define standard errors
serrl=matrix(NA, nrow=1, ncol=L);
```

```
serrh=matrix(NA, nrow=1, ncol=L);
serr1=matrix(NA, nrow=1, ncol=L);
# define confidence intervals
conflow=matrix(NA, nrow=1, ncol=L);
confhigh=matrix(NA, nrow=1, ncol=L);
time=matrix(NA, nrow=1, ncol=L);

foreach (q = 1:length(S)) %dopar%
{
  tic()
  for (h in 1:g)
  {
    #Generate the mesh
    Q0=matrix(NA, nrow=1, ncol=n); #payoff matrix at time 0
    L0=matrix(NA, nrow=1, ncol=n);
    Call=matrix(NA, nrow=1, ncol=n);
    for (f in 1:n)
    {
      S1=matrix(NA, nrow=b, ncol=1);
      S2=matrix(NA, nrow=b, ncol=1);
      for (i in 1:b/2){
        x=rnorm(1);
        S1[i]=S[q]*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*x);
        S1[i+b/2]=S[q]*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*(-x));
        x=rnorm(1);
        S2[i]=S1[i]*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*x);
        S2[i+b/2]=S1[i+b/2]*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*(-x));
      }
#high estimator
      #get the option price at the second last exercise oppotunity
      Q2=matrix(NA, nrow=b, ncol=1); #payoff matrix at time T-1
      Call2=GBSOption('c',as.numeric(S2),K,tstep,r,(r-delta),sigma)@price*
                        exp(-r*(1-tstep))
      for (i in 1:b)
      {
       Q2[i]=max((S2[i]-K)*exp(-r*(1-tstep)),Call2[i]);
      }
      #calculate the weight at time T-2
      density =matrix(NA, nrow=b, ncol=b);
      for (k in 1:b)
      { for (i in 1:b)
        {
          density[i,k]=1/(sigma*sqrt(tstep)*S2[k])*pnorm((log(S2[k]/S1[i])-(r-delta-sigma^2/2)*tstep)/(sigma* sqrt(tstep)));
        }
      }
      w=matrix(NA, nrow=b, ncol=b);
      for (k in 1:b)
      { for (i in 1:b)
        {
          w[i,k]=(density[k,i])/mean(density[,i]);
        }
      }
#calculate the high estimator at time T-2
      Q1=matrix(NA, nrow=b, ncol=1);
      est2=matrix(NA, nrow=b, ncol=1);
      cont2=matrix(NA, nrow=1, ncol=b);
      Call1=matrix(NA, nrow=b, ncol=1);
      for (i in 1:b)
      {
        cont2[i]= mean(Q2*w[,i]); #Mesh estimator
        Q1[i]=max(max(S1[i]-K,0)*exp(-r*tstep),cont2[i]); #Control variate at T-2
        Call1[i]=mean(Call2*w[,i]);
      }
      cont0= mean(Q1);
      Q0[f]=max(S[q]-K,cont0); #the high estimator at T0
      Call[f]=mean(Call1); #control variate at T0
      truecall=GBSOption('c',S[q],K,1,r,(r-delta),sigma)@price;
#low estimator
      lowestimator=Q2;
      #define weightings
      cont=matrix(NA, nrow=b, ncol=b/2);
      exercise=matrix(NA, nrow=b, ncol=b/2);
      weightcont=matrix(NA, nrow=2, ncol=b);
      weights3=matrix(NA, nrow=b-2, ncol=b);
      weightcont2=matrix(NA, nrow=2, ncol=b);
      weights4=matrix(NA, nrow=b-2, ncol=b);
      weightcont3=matrix(NA, nrow=2, ncol=b);
      weights5=matrix(NA, nrow=b-2, ncol=b);
      weights6=matrix(NA, nrow=b-2, ncol=b);
      weights7=matrix(NA, nrow=b-2, ncol=b);
#calculate the low estimator at time T-2
    for (j in 1:(b/2))
    {
      if (j==1){
        for (k in 1:b)
        {
         weightcont[1,k]=density[k,1]/mean(density[,1]);
         weightcont[2,k]=density[k,b/2+1]/mean(density[,b/2+1]);
         for (i in 1:((b-2)/2))
         {
          weights3[i,k]=density[k,i+1]/mean(density[,i+1]);
          weights3[i+(b/2)-1,k]=density[k,i+b/2-1]/mean(density[,i+1+b/2]);
         }
          cont[k,j]= (sum(lowestimator[2:(b/2)]*weights3[(1:(b-2)/2),k])
                      +sum(lowestimator[(b/2+2):b]*weights3[((b-2)/2+1):(b-2),k]))/
                      (b-2);
          exercise[k,j]=((lowestimator[1]*weightcont[1,k])+
                        (lowestimator[1+b/2]*weightcont[2,k]))/2;
        }
      }else if (j==b/2){
        for (k in 1:b)
        {
         weightcont2[1,k]=density[k,(b/2)]/mean(density[,(b/2)]);
         weightcont2[2,k]=density[k,b]/mean(density[,b]);
         for (i in 1:((b-2)/2))
         {
          weights4[i,k]=density[k,i]/mean(density[,i]);
          weights4[i+(b/2)-1,k]=density[k,i+b/2]/mean(density[,i+b/2]);
         }
         cont[k,j]= (sum(lowestimator[2:((b/2)-1)]*weights4[(1:((b-2)/2)),k])
                    +sum(lowestimator[(b/2+1):(b-1)]*weights3[((b-2)/2+1):(b-2),k]))/
                    (b-2);
         exercise[k,j]=((lowestimator[b/2]*weightcont2[1,k])
                           +(lowestimator[b]*weightcont2[2,k]))/2;
        }
      }else{
       for (k in 1:b)
        {
         weightcont3[1,k]=density[k,j]/mean(density[,j]);
         weightcont3[2,k]=density[k,(j+b/2)]/mean(density[,(j+b/2)]);
         for (i in 1:(j-1))          weights5[i,k]=density[k,i]/mean(density[,i]);
         for (i in (j+1):(j+b/2-1))  weights6[i-j,k]=density[k,i]/mean(density[,i]);
         for (i in (j+b/2):b)  weights7[i-(j+b/2),k]=density[k,i]/mean(density[,i]);
        }
        cont[k,j]= (sum(lowestimator[1:(j-1)]*weights5[1:(j-1),k])
                   +sum(lowestimator[(j+1):(j+b/2-1)]*weights6[1:((b/2-1)),k])
                   +sum(lowestimator[(j+b/2+1):b]*weights7[1:(b/2-j),k]))/(b-2);
        exercise[k,j]=((lowestimator[j]*weightcont3[1,k])
                           +(lowestimator[j+b/2]*weightcont3[2,k]))/2;
      }
     }
    }
    lowestimator2=matrix(NA, nrow=1, ncol=b);
    est=matrix(NA, nrow=b, ncol=b/2);
    for (i in 1:b)
    {
     for (k in 1:(b/2))
     {
      if ((max(S1[i]-K,0)*exp(-r*tstep))>=cont[i,k]){
         est[i,k]=max(S1[i]-K,0)*exp(-r*tstep);
      }else{
         est[i,k]=exercise[i,k];
      }
     }
      lowestimator2[i]=mean(est[i,1:(b/2)]*2);
    }
      L0[f]=max(S[q]-K,mean(lowestimator2));
  }
#expected value of the high and low controled estimators
      F=matrix(NA, nrow=n, ncol=1);
      G=matrix(NA, nrow=n, ncol=2);
      H=matrix(1, nrow=n, ncol=1);
      for (k in 1:n)
      {
        F[k]= Q0[k];
        G[k,1:2]= Call[k];
      }
      p = lm(F ~ G)$coefficients
      beta=p[2]
      highestimator[h,q]=mean(Q0)-beta*(mean(Call)-truecall);
      #corrected low estimator
      F1=matrix(NA, nrow=n, ncol=1);
      G1=matrix(NA, nrow=n, ncol=2);
      H1=matrix(1, nrow=n, ncol=1);
      for (k in 1:n){
        F1[k]= (L0[k]);
        G1[k,1:2]= Call[k]
      }
      p = lm(F1 ~ G1)$coefficients
      beta=p[2]
      lowestimator1[h,q]=mean(L0)-beta*(mean(Call)-truecall);
      #point estimator
      pointestimateouter[h,q]=(highestimator[h,q]+lowestimator1[h,q])/2;
  toc()
  }
#preparing the the outputs
  estimator1[q]=mean(pointestimateouter[,q]);
  highest[q]=mean(highestimator[,q]);
  lowest[q]=mean(lowestimator1[,q]);
  serrh[q]=sd(highestimator[,q])/sqrt(g);
  serrl[q]=sd(lowestimator1[,q])/sqrt(g);
  serr1[q]=sd(pointestimateouter[,q])/sqrt(g);
  x = qnorm(0.95);
  conflow[q]=lowest[q]- x*serrl[q];
  confhigh[q]= highest[q]+ x*serrh[q];
}
Output=data.frame(StockPrice=S,
                  t(lowest),
                  t(serrl),
                  t(highest),
                  t(serrh),
                  t(conflow),
                  t(confhigh),
                  t(estimator1)
                 )
print(Output) #Bemudan option prices
```

Algorithm 3: Function for generating asset price paths from geometric Browinian motion

```
#generate the stock path from GBM
PathGeneration=function(r, delta,sigma,s,tstep,b,num_execution,z,assetnum){
  set.seed(100);
  if(is.null(assetnum)) assetnum=1;
  z=matrix(z, nrow=b/2, ncol=(num_execution+1));
  Stockpath=matrix(s, nrow=b, ncol=(num_execution+1));
  for (step in 2:(num_execution+1))
  {
    x=z[,step];
    x=c(x,-x)
    Stockpath[,step]=Stockpath[,step-1]*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*x);
  }
  return(Stockpath);
}
```

Algorithm 4: Pricing Bermudan option on a single asset using EM-Control

```
library("tictoc");
library("fOptions");
K=100;           # strike price
delta=0.1;       # dividend
sigma=0.2;       # annual volatility
r=0.05;          # risk-free rate
T=1;             # maturity
num_execution=3; # exercise opportunities for the option
tstep=1/(num_execution);
S=c(70,80,90,100,110,120); # initial stock price
L=length(S);

#pricing initialisation
b =50000;      #number of asset price paths
g=25;          #number of estimates
Estimation=matrix(0, nrow=1, ncol=L);  #option price
SE=matrix(0, nrow=1, ncol=L);          #option price standard error
prev.theta=theta.maxexp=matrix(300, nrow=1, ncol=num_execution+1); #EM parameters
theta.maxexp[4]=K; #optimal EM parameter

#option pricing
for (q in 1:L){
  tic();
  value=0;
  diff=5; #difference of option price between two subsequent EM estimations
  cp = 0.001; #difference threshold value
  #estimation step
  while (diff!=0)
  {
    simulatedpath=list();
    simulatedeupath=list();
    Estimationtemp=0;
```

```
   prev.theta=theta.maxexp;
   for (iti in 1:g)
   {
     EUpath=Stockpath=PathGeneration(r, delta,sigma,S[q],tstep,b,num_execution);
     simulatedpath[[iti]]=Stockpath;
     for (step in (num_execution+1):1)  EUpath[,step]=GBSOption('c',Stockpath[,step],
                                        K,T-tstep*(step-1),r,(r-delta),sigma)@price;
     simulatedeupath[[iti]]=EUpath;
   }
 #maximization step
 for (stop in (num_execution-1):2)
 {
   Optionsimu=function(x)
   {
     Estimationtemp=0;
     temp.theta=prev.theta;
     temp.theta[stop]=x;
     for (iti in 1:g)
     {
       Stockpath=simulatedpath[[iti]];
       EUpath=simulatedeupath[[iti]];
       CF=pmax(Stockpath-K,0); #Generate the cawshflow table
       Continuation=St=Et=EUpath[,num_execution];
       St=pmax(St,CF[,num_execution]);
       temp.theta[num_execution]=min(Stockpath[CF[,num_execution]>Et,num_execution]);
       for (step in (num_execution-1):2)
       {
         St=St*exp(-r*tstep);
         index=which(CF[,step]>0);
         if (length(index)>=1){
           Continuation[1:b]=0;
           Continuation[index]=temp.theta[step]-K;
           exerciseID=which(CF[,step]>Continuation);
           St[exerciseID]=CF[exerciseID,step];
         }
       }
       St=St*exp(-r*tstep);
       Estimationtemp=c(Estimationtemp,max(mean(St),CF[1,1]));
     }
  return(mean(Estimationtemp[2:length(Estimationtemp)]));
 }
 optimization=optimize(Optionsimu, interval=c(K, max(Stockpath[,stop])), maximum=TRUE)
 optimization
 theta.maxexp[stop]=optimization$maximum;
 value=c(value,optimization$objective);
 }
 diff=round(value[length(value)],cp)-round(value[length(value)-1],cp);
}
#option price estimation
 simulatedpath=list();
 simulatedeupath=list();
 Estimationtemp=0;
 prev.theta=theta.maxexp;
 for (iti in 1:g)
 {
   EUpath=Stockpath=PathGeneration(r, delta,sigma,S[q],tstep,b,num_execution);
   simulatedpath[[iti]]=Stockpath;
   for (step in (num_execution+1):1)
   {
     EUpath[,step]=GBSOption('c',Stockpath[,step],K,T-tstep*(step-1),r,(r-delta),sigma)
   @price;
   }
   simulatedeupath[[iti]]=EUpath;
 }
```

```
  for (iti in 1:g)
  {
    Stockpath=simulatedpath[[iti]];
    EUpath=simulatedeupath[[iti]];
    CF=pmax(Stockpath-K,0); #Generate the cawshflow table
    Continuation=St=Et=EUpath[,num_execution];
    St=pmax(St,CF[,num_execution]);
    prev.theta[num_execution]=min(Stockpath[CF[,num_execution]>Et,num_execution]);
    for (step in (num_execution-1):2)
    {
      St=St*exp(-r*tstep);
      index=which(CF[,step]>0);
      if (length(index)>=1){
        Continuation[1:b]=0;
        Continuation[index]=prev.theta[step]-K;
        exerciseID=which(CF[,step]>Continuation);
        St[exerciseID]=CF[exerciseID,step];
      }
    }
    St=St*exp(-r*tstep);
    Estimationtemp=c(Estimationtemp,max(mean(St),CF[1,1]));
  }
  Estimation[q]=mean(Estimationtemp[2:length(Estimationtemp)]);
  SE[q]=sd(Estimationtemp[2:length(Estimationtemp)])/sqrt(g);
  toc()
}
print(Estimation)  #Bemudan option price
```

Algorithm 5: Functions used for least-squares Monte Carlo method

```
#functions for LSMC continuation value estimation
#LSMC with control variates
LSMEsitmation=function(Y,index,r,step,Stockpath,EUpath){
  X1=rep(1,length(index));
  X2=Stockpath[index,step];
  X3=EUpath[index,step];
  X4=EUpath[index,step]*Stockpath[index,step];
  Beta=lm(Y~0+X1+X2+X3+X4)$coefficients;
  Prediction=Beta[1]*X1+Beta[2]*X2+Beta[3]*X3+Beta[4]*X4;
  return(Prediction);
}

#LSMC with standard polynomial
Poly=function(Y,index,r,step,Stockpath,Degree){
  X=list();
  for (i in 1:Degree)
    X[[i]]=Stockpath[index,step]^i;
  Est=lm(as.formula(paste("Y ~",paste(X, collapse='+'))));
  return(predict(Est));
}

#LSMC with Laguerre polynomial
Laguerre=function(Y,index,r,step,Stockpath,Degree){
  X=list();
  x=Stockpath[index,step];
  X[[1]]=rep(1,length(index));
  X[[2]]=1-x;
  for (i in 3:Degree)
    X[[i]]=((2*i+1-x)*X[[i-1]]-i*X[[i-2]])/(i+1);
  Est=lm(as.formula(paste("Y ~",paste(X, collapse='+'))));
  return(predict(Est));
}

#LSMC with Hermite polynomial
Hermite=function(Y,index,r,step,Stockpath,Degree){
  X=list();
  x=Stockpath[index,step];
  X[[1]]=rep(1,length(index));
  X[[2]]=2*x;
  for (i in 3:Degree)
    X[[i]]=(2*x)*X[[i-1]]-2*i*X[[i-2]];
  Est=lm(as.formula(paste("Y ~",paste(X, collapse='+'))));
  return(predict(Est));
}

#LSMC with Legendre polynomial
Legendre=function(Y,index,r,step,Stockpath,Degree){
  X=list();
  x=Stockpath[index,step];
  X[[1]]=rep(1,length(index));
  X[[2]]=x;
  for (i in 3:Degree)
    X[[i]]=((2*i+1)*x*X[[i-1]]-i*X[[i-2]])/(i+1);
  Est=lm(as.formula(paste("Y ~",paste(X, collapse='+'))));
  return(predict(Est));
}

#LSMC with Chebysheve polynomial
Chebysheve=function(Y,index,r,step,Stockpath,Degree){
  X=list();
  x=Stockpath[index,step];
  X[[1]]=rep(1,length(index));
  X[[2]]=x;
  for (i in 3:Degree)
    X[[i]]=2*x*X[[i-1]]-X[[i-2]];
  Est=lm(as.formula(paste("Y ~",paste(X, collapse='+'))));
  return(predict(Est));
}


#LSMC with piece-wise linear regression
piece.formula <- function(var.name, knots) {
  formula.sign <- rep(" - ", length(knots))
  formula.sign[knots < 0] <- " + "
  paste(var.name, "+",
        paste("I(pmax(", var.name, formula.sign, abs(knots), ", 0))",
              collapse = " + ", sep=""))
}

#LSMC with spline regression
Piece.Spline=function(Y,index,r,step,Stockpath,Knot){
  x=Stockpath[index,step];
  knots=seq(min(x), max(x), len=Knot+2);
  Est=lm(as.formula(paste("Y ~",piece.formula("x",knots) )));
  return(predict(Est));
}
```

Algorithm 6: Functions for generating asset price paths using quasi random sequences

```
#functions for generating quasi Monte Carlo sequences
#Halton sequences
GetHalton=function(b,base){
  hs=matrix(0,b,1);
  numbits=1+round(log(b)/log(base));
  BaseVec=base^(-(1:numbits));
  WorkVec=matrix(0,numbits,1);
  for (i in 1:b){
    j = 1; done = 0;
    while (done!=1){
      WorkVec[j]=WorkVec[j]+1;
      if (WorkVec[j]<base){
        done=1;
      }else{
        WorkVec[j]=0;
        j=j+1;
      }
    }
    hs[i]=sum(WorkVec*BaseVec);
  }
  return(hs)
}

QuasiMCSample.halton = function(b,B1,B2) {
  H1 = GetHalton(round(b/2),B1);
  H2 = GetHalton(round(b/2),B2);
  Vlog=sqrt(-2*log(H1));
  samp=matrix(0,b,1);
  samp[seq(1,b,2)]=Vlog*cos(2*pi*H2);
  samp[seq(2,b,2)]=Vlog*sin(2*pi*H2);
  return(samp)
}


#Sobol sequences
QuasiMCSample.sobol = function(b) {

  SS= sobol(b/2,2,init = TRUE,scrambling = 1);
  H1 =sample(SS[,1],b/2);
  H2 =sample(SS[,2],b/2);
  Vlog=sqrt(-2*log(H1));
  samp=matrix(0,b,1);
  samp[seq(1,b,2)]=Vlog*cos(2*pi*H2);
  samp[seq(2,b,2)]=Vlog*sin(2*pi*H2);
  return(samp)
}

#moment matching
QuasiMCSample.MM2 = function(b) {
  H1 = runif(b/2,0,1);
  H2 = runif(b/2,0,2);
  Vlog=sqrt(-2*log(H1));
  samp=matrix(0,b,1);
  samp[seq(1,b,2)]=Vlog*cos(2*pi*H2);
  samp[seq(2,b,2)]=Vlog*sin(2*pi*H2);
  samp = samp-mean(samp)*(1/sd(samp));
  return(samp)
}

#generate the stock path using different sequences
PathGeneration=function(r, delta,sigma,s,tstep,b,num_execution,B1,B2){
Stockpath=matrix(0, nrow=b, ncol=(num_execution+1));
Stockpath[,1]=s;
for (step in 2:(num_execution+1)){
  #Sample=QuasiMCSample.halton(b,step+B1,step+5+B2);
  #Sample=QuasiMCSample.sobol(b/2);
  #Sample=QuasiMCSample.MM2(b);
  Sample=rnorm(b/2); #pseudo random
  Sample=c(-Sample,Sample)
  Stockpath[,step]=Stockpath[,step-1]*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*
    Sample);
}
return(Stockpath);
}
```

Algorithm 7: Pricing Bermudan option on a single asset using least-squares Monte Carlo

```
library("tictoc");
library("fOptions");
K=100;           # strike price
delta=0.1;       # dividend
sigma=0.2;       # annual volatility
r=0.05;          # risk-free rate
T=1;             # maturity
num_execution=3; # exercise opportunities for the option
tstep=1/(num_execution);
S=c(70,80,90,100,110,120); # initial stock price
L=length(S);

#pricing initialisation
b =50000;      #number of asset price paths
g=25;          #number of estimates
L=length(S);
Value=matrix(0, nrow=g, ncol=L);
Estimation=matrix(0, nrow=1, ncol=L);
SE=matrix(0, nrow=1, ncol=L);

#option pricing
for (q in 1:L)
{
  for (f in 1:g)
  {
  tic();
  #pruning for the second last exercise opportunities
  EUpath=Stockpath=PathGeneration(r, delta,sigma,S[q],tstep,b,num_execution);
  for (step in (num_execution+1):1)
  {
   EUpath[,step]=GBSOption('c',Stockpath[,step],K,T-tstep*(step-1),r,
                        (r-delta),sigma)@price;
  }
  CF=pmax(Stockpath-K,0);                    #generate the cawshflow matrix
  Continuation=St=Et=EUpath[,num_execution];  #generate the continuation value matrix
  St=pmax(St,CF[,num_execution]);             #generate the option value matrix
  for (step in (num_execution-1):2)
  {
    St=St*exp(-r*tstep);
    Et=Et*exp(-r*tstep);
    index=which(CF[,step]>0);
    if (length(index)>=1){
    #estimate continuation value with control variate
    #substitute function 'LSMEsitmation' with functions listed in Algorithm 5
    #for different LSMC method
    #Continuation value estimation
    Ft=LSMEsitmation(St[index],index,r,step,Stockpath,EUpath);
    #Control variable estimation
    Etest=LSMEsitmation(Et[index],index,r,step,Stockpath,EUpath);
    Et2est=LSMEsitmation(Et[index]^2*exp(r*tstep),index,r,step,Stockpath,EUpath);
    EtCest=LSMEsitmation(St[index]*Et[index]*exp(r*tstep),index,r,step,Stockpath,EUpath);
    coeff=-(EtCest-Ft*Etest)/(Et2est-Etest^2);#Control adjustment coefficient
    Ftc=Ft+coeff*(Etest-EUpath[index,step]);

    Continuation[1:b]=0;
    Continuation[index]=Ftc;
    exerciseID=which(CF[,step]>Continuation);
    St[exerciseID]=CF[exerciseID,step];
    Et[exerciseID]=EUpath[exerciseID,step];
    }
  }
  St=St*exp(-r*tstep);
  Et=Et*exp(-r*tstep);
  St0=mean(St);
  Etest=mean(Et);
  Et2est=mean(Et^2)*exp(r*tstep);
  EtCest=mean(St*Et)*exp(r*tstep);
  coeff=-(EtCest-St0*Etest)/(Et2est-Etest^2);
  Stc=St0+coeff*(Etest-EUpath[1,1]);
  Value[f,q]=max(Stc,CF[1,1]);
  toc()
  }
  Estimation[q]=mean(Value[,q]);
  SE[q]=sd(Value[,q])/sqrt(g);
}
print(Estimation) #Bemudan option price
```

B.2 PRICING ALGORITHMS ON TWO ASSETS

Algorithm 8: Pricing Bermudan option on two assets using Black and Scholes' formula

```
#pricing Bermudan option on two assets using black-scholes method
EUMax.2 = function(s1 , s2 ,K, sigma1 , sigma2 , b1 , b2 , r ,T, rho){
  sigma = sqrt ( sigma1^2+sigma2^2-2*rho*sigma1*sigma2 );
  d = ( log ( s1/s2 )+(b1-b2+sigma^2/2)*T)/( sigma*sqrt (T));
  y1 = ( log ( s1/K)+(b1+sigma1 ^2/2)*T)/( sigma1*sqrt (T));
  y2 = ( log ( s2/K)+(b2+sigma2 ^2/2)*T)/( sigma2*sqrt (T));
  rho1 = ( sigma1-rho*sigma2 )/sigma;
  rho2 = ( sigma2-rho*sigma1 )/sigma;
  varcov = matrix( c (1 , rep( rho , 2 ) , 1 ) ,nrow = 2);
  varcov1 = matrix( c (1 , rep( rho1 , 2 ) , 1 ) ,nrow = 2);
  varcov2 = matrix( c (1 , rep( rho2 , 2 ) , 1 ) ,nrow = 2);
  value =
  (s1*exp((b1-r)*T)*pmnorm( cbind(y1 , d) , rep (0 ,2) , varcov1 )+
  s2*exp((b2-r)*T)*pmnorm( cbind(y2,-d+sigma*sqrt (T)) , rep (0 ,2), varcov2 )-
  K*exp(-r*T)*(1-pmnorm(cbind(-y1+sigma1*sqrt (T),-y2+sigma2*sqrt (T)),rep(0,2),
  varcov)));
  return(value);
}
```

Algorithm 9: Pricing Bermudan option on two assets using stochastic tree

```
library("tictoc");
library("foreach"); #optional: enable parallel computation
```

```

K      = 100;            # strike price
delta  = 0.1;            # dividend yield
mT     = 1;              # time to maturity
sigma  = 0.2;            # assets annual volatility
rho = rbind(c(1,0.3),c(0.3,1));
t      = c(0,1/3,2/3,1); # exercise opportunities for the Bermudan option
tstep  = 1/3;
r      = 0.05;           # risk-free rate
b      =  10;            # branching parameter
iteration=20;            # number of Monte Carlo iterations
repitition=10;           # number of Monte Carlo estimators

#pricing initialisation
S      = c(70,80,90,100,110,120); # asset prices
L      = length(S);
emptyr = matrix(NA, nrow = repitition, ncol = L);
Ehighest         = emptyr; # option price high estimator
Elowest          = emptyr; # option price low estimator
pointestimateE   = emptyr; # option price point estimator
empty1 = matrix(NA, nrow = 1, ncol = L);
# define expected estimators
pointestimator   = empty1;
highestimator    = empty1;
lowestimator     = empty1;
# define standard errors
sep    = empty1;
sel    = empty1;
seh    = empty1;
# define confidence intervals
CILow       = empty1;
CIHigh      = empty1;
RealError   = empty1;

#create stohastic tree
createtree = function(S,q,tstep,r,delta,sigma,rho,assetnum,shock){
  if (shock == NULL) shock = 0;
  Spath = matrix(NA, nrow = b^(1/tstep), ncol = (1/tstep));
  for (p in 1:(1/tstep))
  {
   for (j in 1:(b^(p-1)))
    {
      for (i in 1:(b))
      {
        if (p-1 == 0){
          Sp = S[q];
        }else{
          Sp = Spath[j,(p-1)];
        }
        x =  sample(shock,1);
        Spath[b*(j-1)+i,p] = Sp*exp((r-delta-sigma^2/2)*tstep+sigma*sqrt(tstep)*x);
      }
    }
  }
  return(Spath);
}


# option pricing
foreach (q  =  1:length(S)) %dopar% {
  tic();
  for (M in 1:repitition)
  {
    lowestc = matrix(NA, nrow = iteration, ncol = 1);
    lowest = matrix(NA, nrow = iteration , ncol = 1);
    highestc = matrix(NA, nrow = iteration , ncol = 1);
    highest = matrix(NA, nrow = iteration , ncol = 1);
    for (f in 1:iteration)
    {
    randomnumber = matrix(rnorm((b+b^2+b^3)*2),ncol = 2);
    randomnumber = randomnumber%*%chol(rho);
    Spath1 = createtree(S,q,tstep,r,delta,sigma,rho,1,randomnumber[,1]);
    Spath2 = createtree(S,q,tstep,r,delta,sigma,rho,2,randomnumber[,2]);
    # High estimator
    est = matrix(NA, nrow = b^(1/tstep), ncol = 1/tstep);
    est[,(1/tstep)] = pmax(Spath2[,(1/tstep)]-K,Spath1[,(1/tstep)]-K,0);
      for (Step in (1/tstep):2)
      {
        index = seq(1,b^(Step),b);
        for (i in 1:(b^(Step-1)))
        {
          continue = (sum(est[index[i]:(index[i]+b-1),Step])*exp(-r*tstep))/b;
          est[i,Step-1] = max(max(Spath1[i,Step-1],Spath2[i,Step-1])-K,continue);
        }
      }
    contnc = mean(est[,1],na.rm = TRUE)*exp(-r*tstep);
    highest[f] = max(S[q]-K,contnc);

# Low estimator
    estL = matrix(NA, nrow = b^(1/tstep), ncol = 1/tstep+1);
    estL[,(1/tstep)] = pmax(Spath2[,(1/tstep)]-K,Spath1[,(1/tstep)]-K,0);
    for (Step in (1/tstep):1)
    {
      index = seq(1,b^(Step),b);
      for (i in 1:(b^(Step-1)))
      {
        tempest = 0;
        for (j in 1:(b/2))
        {
          substract = estL[index[i]+(j-1),Step]+estL[index[i]+(j-1)+b/2,Step];
          continueL = (sum(estL[index[i]:(index[i]+b-1),Step])-substract)*exp(-r*tstep)/b;
          spot = ifelse(Step-1 == 0,S[q],max(Spath1[i,Step-1],Spath2[i,Step-1]));
          tempest = c(tempest,ifelse((spot-K)<continueL,substract*exp(-r*tstep)/2,
                    max(spot-K,0)));
        }
        if (Step!= 1){
          estL[i,Step-1] = mean(tempest[2:length(tempest)]);
        }else{
          lowest[f] = mean(tempest[2:length(tempest)]);
        }
      }
     }
    }
  #calculated the expected estimators
  Ehighest[M,q] = mean(highest);
  Elowest[M,q] = mean(lowest);
    pointestimateE[M,q] = (max(S[q]-K,Elowest[M,q])+Ehighest[M,q])/2;

  }
  #preparing the the outputs
  lowestimator[q] = mean(Elowest[,q]);
  highestimator[q] = mean(Ehighest[,q]);
  pointestimator[q] = mean(pointestimateE[,q]);

  sep[q] = sd(pointestimateE[,q])/sqrt(repitition);
  sel[q] = sd(Elowest[,q])/sqrt(repitition);
  seh[q] = sd(Ehighest[,q])/sqrt(repitition);

  CILow[q]  = lowestimator[q] - qnorm(0.95)*sel[q];
  CIHigh[q] =  highestimator[q]+ qnorm(0.95)*seh[q];
  toc();
}
Output = data.frame(StockPrice = S,
                  Lowest = t(lowestimator),
                  StderrL = t(sel),
                  Highest = t(highestimator),
                  StderrH = t(seh),
                  Pointest = t(pointestimator),
                  ConfidenceILow = t(CILow),
                  ConfidentceHigh = t(CIHigh),
                  Pointest = t(pointestimator),
                  )
print(Output) #Bemudan option prices
```

Algorithm 10: Pricing Bermudan option on two assets using least-squares Monte Carlo

```
library("tictoc");
library("mnormt");
K = 100;   # Strike Price
assetnum =2;
delta = 0.1; # Dividend Yield
sigma = 0.2; # Volatility
rho = rbind(c(1,0.3),c(0.3,1));
r = 0.05; # Risk-free rate
T = 1;       # Time to maturity
num_execution = 3; # Exercise opportunities for the Bermudan PointEstimator
tstep = 1/(num_execution);
S = c(70,80,90,100,110,120); # Initial stock price


#pricing initialisation
b =20000;      #number of asset price paths
g=25;          #number of estimates
L = length(S);
Value = matrix(0, nrow = g, ncol = L);
Estimation = matrix(0, nrow = 1, ncol = L);
SE = matrix(0, nrow = 1, ncol = L);

for (q in 1:L){
  tic();
  for (f in 1:g)
  {
    simulatedpath = list();
    randomnumber = matrix(rnorm(b*(num_execution+1)/2*assetnum),ncol = assetnum);
    randomnumber = randomnumber%*%chol(rho);
    for (assetnum in 1:2)
    {
      z = randomnumber[,assetnum];
      simulatedpath[[assetnum]] =
          PathGeneration(r, delta,sigma,S[q],tstep,b,num_execution,z,assetnum);
    }
    simulatedeupath = simulatedpath;
    for (i in 1:2)
    {
      string = paste("asset",i, " = matrix(unlist(simulatedpath[",i,"]),ncol = num_execution+1)",sep = "");
                eval(parse(text = string));
    }
    EUpath = Stockpath = matrix(0, nrow = b, ncol = (num_execution+1));
    Stockpath = pmax(asset1,asset2);
    EUpath[,num_execution+1] = pmax(pmax(asset1[,num_execution+1],
                                    asset2[,num_execution+1])-K,0);
    for (step in (num_execution):1)
      EUpath[,step] = EUMax.2(asset1[,step],asset2[,step],K,
                              sigma,sigma,r-delta,r-delta,r,T-tstep*(step-1),0.3);
    CF = pmax(pmax(asset1,asset2)-K,0);
    Continuation = St = Et = EUpath[,num_execution];
    St = pmax(St,CF[,num_execution]);
    for (step in (num_execution-1):2)
    {
      St = St*exp(-r*tstep);
      Et = Et*exp(-r*tstep);
      index = which(CF[,step]>0);
      if (length(index)>= 1){
      #Continuation value estimation
      Ft = LSMEsitmation(St[index],index,r,step,Stockpath,EUpath);
      #Control variable estimation
      Etest = LSMEsitmation(Et[index],index,r,step,Stockpath,EUpath);
      Et2est = LSMEsitmation(Et[index]^2*exp(r*tstep),index,r,step,Stockpath,EUpath);
      EtCest = LSMEsitmation(St[index]*Et[index]*exp(r*tstep),index,r,step,Stockpath,EUpath);

      coeff = -(EtCest-Ft*Etest)/(Et2est-Etest^2);#Control adjustment coefficient
      Ftc = Ft+coeff*(Etest-EUpath[index,step]);

      Continuation[1:b] = 0;
      Continuation[index] = Ftc;
      exerciseID = which(CF[,step]>Continuation);
      St[exerciseID] = CF[exerciseID,step];
      Et[exerciseID] = EUpath[exerciseID,step];
      }
    }
    St = St*exp(-r*tstep);
    Et = Et*exp(-r*tstep);
    St0 = mean(St);
    Etest = mean(Et);
    Et2est = mean(Et^2)*exp(r*tstep);
    EtCest = mean(St*Et)*exp(r*tstep);
    coeff = -(EtCest-St0*Etest)/(Et2est-Etest^2);
    Stc = St0+coeff*(Etest-EUpath[1,1]);
    Value[f,q] = max(Stc,CF[1,1]);
  }
  Estimation[q] = mean(Value[,q]);
  SE[q] = sd(Value[,q])/sqrt(g);
  toc();
}
print(Estimation) #Bemudan option price
```

B.3 PRICING ALGORITHMS ON FIVE ASSETS

Algorithm 11: Pricing European option on five assets using multinomial tree

```
library("gtools")
num_execution=3;    #number of executions allowed
                    #assuming equal interval between 2 executions
```

```
asset.num=5;        #number of assets
m=10;               #number of steps between two executions
mT=1;               #maturity
sigma=0.2;          #annual volatility of stock price
delta=0.1;          #dividend
r=0.05;             #mean rate of return
S1=S2=S3=S4=S5=100;#initial price of the stock
K=100 ;             #strike price
corrmatrix=matrix(0.3,5,5);
diag(corrmatrix)=1 #assets correlation matrix

#option pricing setup
mu=matrix(r-delta-0.5*sigma^2,nrow=asset.num)*mT;
ini.price=matrix(c(S1,S2,S3,S4,S5),nrow=asset.num);
sdm=c(sigma,sigma,sigma,sigma,sigma);
covmatrix.annual=sdm%*%t(sdm)*corrmatrix;
covmatrix=sqrt(mT/m)*covmatrix.annual;
chol.covmatrix.annual=t(chol(covmatrix.annual));
A.matrix=2*sqrt(mT/m)*chol.covmatrix.annual;
b.vector=mu*mT-sqrt(mT*m)*chol.covmatrix.annual%*%matrix(1,nrow=asset.num,ncol=1);

#create tree
tree=list();
tree.matrix=permutations(n=m+1,r=asset.num,v=0:m,repeats.allowed=TRUE);
for (i in 1:(m+1)^asset.num){
  tree[[i]]=tree.matrix[i,]
}
x.vector=list();
w.vector=list();
value.vector=list();
payoff.vector=list();
prob.vector=list();
exp.vector=matrix(0,nrow=(m+1)^asset.num,ncol=1);

for (i in 1:(m+1)^asset.num){
x.vector[[i]]=A.matrix%*%unlist(tree[[i]])+b.vector;
w.vector[[i]]=exp(x.vector[[i]]);
value.vector[[i]]=max(w.vector[[i]]*ini.price);
payoff.vector[[i]]=max(value.vector[[i]]-K,0);
choose.element=unlist(tree[[i]]);
prob.vector[[i]]=choose(m,choose.element[1])*choose(m,choose.element[2])*
                 choose(m,choose.element[3])*choose(m,choose.element[4])*
                 choose(m,choose.element[5])*(0.5)^(asset.num*m)
exp.vector[i]=prob.vector[[i]]*payoff.vector[[i]];
}


print(sum(exp.vector)*exp(-r)) #return European option price
```

Algorithm 12: Pricing Bermudan option on the geometric average of five assets using binomial tree

```
call = 1;             #1 for call option, -1 for put option
num_execution = 4;    #number of executions allowed, assuming equal
                      #interval between 2 executions
asset.num = 5;        #number of assets
n = 3;                #number of steps between two executions
N = num_execution*n;  #total number of steps
mT = 1;               #maturity
s = 0.2;              #annual volatality of stock price
sigma = c(s,s,s,s,s);
r = 0.05;             #mean rate of return
```

```
S = 120 ;              #initial price of the stocks
K = 100 ;              #strike price
rho = matrix(0.3,5,5);
diag(rho)=1;           #assets correlation matrix

#convert assets to a single asset following Kemna (1990)
covmatrix.annual = sigma%*%t(sigma)*rho;
sigma.2 = sum(sum(covmatrix.annual))/asset.num^2;
delta = 1/2*(sum(sigma^2)/asset.num-sigma.2)+0.1;

#option pricing setup
dT = mT/N;
up = exp(sqrt(sigma.2*dT));
dw = 1/up;
prob=(exp((r-delta)*dT)-dw)/(up-dw);

#create tree
Stockpath = matrix(0, nrow = N+1, ncol = N+1);
Value = matrix(0, nrow = N+1, ncol = N+1);
for (i in 1:(N+1))
{
  for(j in 1:i){
    Stockpath[j,i]=S*(up^(i-j))*(dw^(j-1));
    Value[j,i]=max(call*(Stockpath[j,i]-K),0);
  }
}
BS = matrix(0, nrow = N+1, ncol = N+1);
BS[,N+1]=Value[,N+1];
a = 0:(N);
index = a[seq(1,N+1,n)];
boundB = 0;
for (g in (num_execution+1):2)
{
  start = index[g-1]+1;
  end = index[g];
  for (j in end:start)
  {
    if(j==index[g-1]+1)
    {
      exercise = 0;
      for (i in 1:j){
        BS[i,j] = max(exp(-r/N)*(prob*BS[i,j+1]+(1-prob)*
                      BS[i+1,j+1]),Value[i,j]);
        exercise = c(exercise,ifelse(exp(-r/N)*(prob*BS[i,j+1]+
                       (1-prob)*BS[i+1,j+1])<Value[i,j],i,0));}
        boundB = plotboundprep(call,exercise,boundB,Stockpath,j,N);
    }else{
      for (i in 1:j) BS[i,j]=exp(-r/N)*(prob*BS[i,j+1]+(1-prob)*
                                BS[i+1,j+1]);
    }
  }
}
BValue = BS[1,1];
print(BValue); #return Bermudan option price
```

Algorithm 13: Pricing Bermudan option on the geometric average of five assets using least-squares Monte Carlo

```
library("fOptions");
library("tictoc");
library("randtoolbox");
```

```r
K=100;            # strike price
assetnum=5;       # number of assets
delta=0.1;        # dividend yield
sigma=0.2;        # annual assets volatility
rho=matrix(0.3,5,5);
diag(rho)=1       # assets correlation matrix
r=0.05;           # risk-free rate
T=1;              # time to maturity
num_execution=3; # exercise opportunities for the Bermudan PointEstimator
tstep=1/(num_execution);
S=c(70,80,90,100,110,120); # Initial stock price

#convert assets to a single asset following Kemna (1990)
#to calculate the control variate
covmatrix.annual=rep(sigma,assetnum)%*%t(rep(sigma,assetnum))*rho;
sigma.2=sum(sum(covmatrix.annual))/assetnum^2;
delta.adj=delta+1/2*(sum(rep(sigma,assetnum)^2)/assetnum-sigma.2);

#pricing initialisation
b =10000 ;   #number of asset price paths
g=25     ;   #number of estimates
L=length(S);
Value=matrix(0, nrow=g, ncol=L);
Estimation=matrix(0, nrow=1, ncol=L);
SE=matrix(0, nrow=1, ncol=L);
gmean = function(x, na.rm=TRUE){
  exp(sum(log(x[x > 0]), na.rm=na.rm) / length(x))
}

#option pricing
for (q in 1:L){
  tic();
  for (f in 1:g){
    simulatedpath=list();
    vals=list(var="Normal", dist="norm",params=c(0,1));
    randomnumber=matrix(rnorm(b*(num_execution+1)/2*assetnum),ncol=assetnum);
    #alternative random numbers from quasi sequences
    #for (i in 1:assetnum){
    #randomnumber[,i]=sample(QuasiMCSample.halton(b*(num_execution+1)/2,
    #                 i+f+2,i+f+5+7),b*(num_execution+1)/2);
    #randomnumber[,i]=QuasiMCSample.sobol(b*(num_execution+1)/2);
    #randomnumber[,i]=QuasiMCSample.MM2(b);
  }
    randomnumber=randomnumber%*%chol(rho);
    for (i in 1:assetnum)
    {
      z=randomnumber[,i];
      simulatedpath[[i]]=
          PathGeneration(r, delta,sigma,S[q],tstep,b,num_execution,z,i);
    }
    for (i in 1:assetnum){
      string=paste("asset",i, "=matrix(unlist(simulatedpath[",i,"]),
                    ncol=num_execution+1)",sep = "");
      eval(parse(text=string));
    }

    EUpath=Stockpath=matrix(0, nrow=b, ncol=(num_execution+1));
    for (step in 1:(num_execution+1))
    {
      for (i in 1:assetnum){
      if (i == 1) {matrix.mean=asset1[,step];}
      else{
        string=paste("matrix.mean=cbind(matrix.mean,","asset",i, "[,",step,"])",sep = "");
        eval(parse(text=string));
      }
      }
      Stockpath[,step]=apply(matrix.mean,1,gmean);
    }
    EUpath[,num_execution+1]=pmax(Stockpath[,num_execution+1]-K,0);
    for (step in (num_execution):1)
    {
      EUpath[,step]=GBSOption('c',Stockpath[,step],K,T-tstep*(step-1),r,(r-delta.adj),sqrt(sigma.2))@price;
    }
    CF=pmax(Stockpath-K,0); #generate the cawshflow table
    Continuation=St=Et=EUpath[,num_execution];
    St=pmax(St,CF[,num_execution]);

    for (step in (num_execution-1):2)
    {
      St=St*exp(-r*tstep);
      Et=Et*exp(-r*tstep);
      index=which(CF[,step]>0);
      if (length(index)>=1){
      #continuation value estimation
      #substitute function 'LSMEsitmation' with functions listed in Algorithm 5
      #for different LSMC method
      #continuation value estimation
       Ft=LSMEsitmation(St[index],index,r,step,Stockpath,EUpath);
      #Control variable estimation
      Etest=LSMEsitmation(Et[index],index,r,step,Stockpath,EUpath);
      Et2est=LSMEsitmation(Et[index]^2*exp(r*tstep),index,r,step,Stockpath,EUpath);
      EtCest=LSMEsitmation(St[index]*Et[index]*exp(r*tstep),index,r,step,Stockpath,EUpath);
      coeff=-(EtCest-Ft*Etest)/(Et2est-Etest^2);#Control adjustment coefficient
      Ftc=Ft+coeff*(Etest-EUpath[index,step]);

      Continuation[1:b]=0;
      Continuation[index]=Ftc;
      exerciseID=which(CF[,step]>Continuation);
      St[exerciseID]=CF[exerciseID,step];
      Et[exerciseID]=EUpath[exerciseID,step];
      }
    }
    St=St*exp(-r*tstep);
    Et=Et*exp(-r*tstep);
    St0=mean(St);
    St0
    Etest=mean(Et);
    Et2est=mean(Et^2)*exp(r*tstep);
    EtCest=mean(St*Et)*exp(r*tstep);
    coeff=-(EtCest-St0*Etest)/(Et2est-Etest^2);
    Stc=St0+coeff*(Etest-EUpath[1,1]);
    Value[f,q]=max(Stc,CF[1,1]);
  }
Estimation[q]=mean(Value[,q]);
SE[q]=sd(Value[,q])/sqrt(g);
RE[q]=abs((Estimation[q]-truevalue[q])/truevalue[q]);
RMSE[q]=sqrt(mean((Estimation[q]-truevalue[q])^2));
Estimation
toc()
}
print(Estimation) #Bemudan option price
```

BIBLIOGRAPHY

Abbas-Turki, L. A. and Lapeyre, B (2009). “American Options Pricing on Multicore Graphic Cards.” *Proceedings of International Conference on Business Intelligence and Financial Engineering (BIFE)*. IEEE, pp. 307–311.

Analytics, R. and Weston, S. (2015). *foreach: Provides Foreach Looping Construct for R*. R package version 1.4.3.

Andricopoulos, A. D. et al. (2003). “Universal option valuation using quadrature methods.” *Journal of Financial Economics* 67.3, pp. 447–471.

Anson, H. T., Thomas, D., and Luk, W. (2012). “Design exploration of quadrature methods in option pricing.” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20.5, pp. 818–826.

Azzalini, A. and Genz, A. (2016). *The R package `mnormt`: The multivariate normal and t distributions (version 1.5-5)*.

Barone Adesi, G. and Whaley, R. E. (1987). “Efficient Analytic Approximation of American Option Values.” *The Journal of Finance* 42.2, pp. 301–320.

Barraquand, J. and Martineau, D. (1995). “Numerical valuation of high dimensional multivariate American securities.” *Journal of Financial and Quantitative Analysis* 30.3, pp. 383–405.

Bäuerle, N and Rieder, U (2011). *Markov decision processes with applications to finance*. Springer Science & Business Media.

Bellman, R (1952). “On the theory of dynamic programming.” *Proceedings of the National Academy of Sciences* 38.8, pp. 716–719.

Bellman, R. (1954). “Some Problems in the Theory of Dynamic Programming.” *Econometrica: Journal of the Econometric Society* 22.1, pp. 37–48.

Biancardi, M. and Villani, G. (2014). “A Robustness Analysis of Least-Squares Monte Carlo for R&D Real Options Valuation.” *Mathematical and Statisti-*

cal Methods for Actuarial Sciences and Finance. Cham: Springer International Publishing, pp. 27–30.

Biancardi, M. and Villani, G. (2016). “Robust Monte Carlo Method for R&D Real Options Valuation.” *Computational Economics* 49.3, pp. 481–498.

Black, F and Scholes, M (1973). “The pricing of options and corporate liabilities.” *Journal of political economy* 81.3, pp. 637–654.

Bossaerts, P. (1989). “Simulation estimators of optimal early exercise.” *Working Paper, Carnegie Mellon University*.

Box, G. E. P. and Cox, D. R. (1964). “An Analysis of Transformations.” *Journal of the Royal Statistical Society. Series B (Methodological)* 26.2, pp. 211–252.

Boyle, P. P. (1977). “Options: A Monte Carlo approach.” *Journal of Financial Economics* 4.3, pp. 323–338.

Boyle, P. P., Evnine, J., and Gibbs, S. (1989). “Numerical Evaluation of Multivariate Contingent Claims.” *Review of Financial Studies* 2.2, pp. 241–250.

Brandão, L. E., Dyer, J. S., and Hahn, W. J. (2005). “Using Binomial Decision Trees to Solve Real-Option Valuation Problems.” *Decision Analysis* 2.2, pp. 69–88.

Brennan, M. J. and Schwartz, S. E. (1978). *Finite difference methods and jump processes arising in the pricing of contingent claims: A synthesis*. Vol. 13. Journal of Financial and Quantitative Analysis.

Broadie, M. and Cao, M. (2008). “Improved lower and upper bound algorithms for pricing American options by simulation.” *Quantitative Finance* 8.8, pp. 845–861.

Broadie, M. and Glasserman, P. (1997). “Pricing American-style securities using simulation.” *Journal of economic dynamics and control* 21.8, pp. 1323–1352.

Broadie, M. and Glasserman, P. (2004). “A stochastic mesh method for pricing high-dimensional American options.” *Journal of Computational Finance* 7, pp. 35–72.

Caflisch, R. E. and Chaudhary, S (2004). “Monte Carlo methods for American options.” *Simulation Conference*, pp. 1656–1660.

Carmona, R., Hinz, J., and Yap, N. (2007). "Solving convex stochastic switching problems." *Journal of LaTeX Class Files* 6.1, pp. 1–6.

Carr, P. (1998). "Randomization and the American Put." *Review of Financial Studies* 11.3, pp. 597–626.

Carriere, J. F. (1995). "Valuation of the early-exercise price for options using simulations and nonparametric regression." *Insurance: Mathematics and Economics* 19.1, pp. 19–30.

Chaudhary, S. (2005). "American options and the LSM algorithm: quasi-random sequences and Brownian bridges." *The Journal of Computational Finance* 8.4.

Chow, Y. S. and Robbins, H. (1963). "On optimal stopping rules." *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete* 2.1, pp. 33–49.

Christophe, D. and Petr, S. (2015). *randtoolbox: Generating and Testing Random Numbers*. R package version 1.17.

Copeland, T and Antikarov, V (2001). *Real options*. Texere.

Cox, J. C., Ross, S. A., and Rubinstein, M. (1979). "Option pricing: A simplified approach." *Journal of Financial Economics* 7.3, pp. 229–263.

Crundwell, F. (2008). *Finance for Engineers*. Evaluation and Funding of Capital Projects. Springer Science & Business Media.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). "Maximum likelihood from incomplete data via the EM algorithm." *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1, pp. 1–38.

Dupire, B (1998). *Monte Carlo: methodologies and applications for pricing and risk management*. Risk Books.

Egloff, D. (2005). "Monte Carlo algorithms for optimal stopping and statistical learning." *The Annals of Applied Probability* 15.2, pp. 1396–1432.

Fu, M. C. et al. (2001). "Pricing American options: A comparison of Monte Carlo simulation approaches." *Journal of Computational Finance* 4.3, pp. 39–88.

Gapeev, P. V. and Rodosthenous, N. (2014). "Optimal Stopping Problems in Diffusion-Type Models with Running Maxima and Drawdowns." *Journal of Applied Probability* 51.3, pp. 799–817.

Gentle, J. E. (2013). *Random Number Generation and Monte Carlo Methods*. Statistics and Computing. Springer Science & Business Media.

Geske, R and Johnson, H. E. (1984). "The American put option valued analytically." *The Journal of Finance* 39.5, pp. 1511–1524.

Geske, R. (1979). "The valuation of compound options." *Journal of Financial Economics* 7.1, pp. 63–81.

Glasserman, P. (2013). *Monte Carlo Methods in Financial Engineering*. Springer Science & Business Media.

Glasserman, P. and Yu, B. (2004). "Number of paths versus number of basis functions in American option pricing." *The Annals of Applied Probability* 14.4, pp. 2090–2119.

Green, P. J. and Silverman, B. W. (1993). *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*. CRC Press.

Haugh, M. B. and Kogan, L. (2004). "Pricing American Options: A Duality Approach." *Operations Research* 52.2, pp. 258–270.

Hinz, J. (2014). "Optimal Stochastic Switching under Convexity Assumptions." *SIAM Journal on Control and Optimization* 52.1, pp. 164–188.

Hull, J. C. (2016). *Fundamentals of Futures and Options Markets, Global Edition*. Pearson Higher Education.

Izrailev, S. (2014). *tictoc: Functions for timing R scripts, as well as implementations of Stack and List structures.* R package version 1.0.

Johnson, H. E. (1983). "An Analytic Approximation for the American Put Price." *The Journal of Financial and Quantitative Analysis* 18.1, p. 141.

Joshi, M. S. (2008). *The Concepts and Practice of Mathematical Finance*. Cambridge University Press.

Kemna, A. G. Z. and Vorst, A. C. F. (1990). "A pricing method for options based on average asset values." *Journal of Banking & Finance* 14.1, pp. 113–129.

Kim, J. et al. (2016). "A practical finite difference method for the three-dimensional Black–Scholes equation." *European Journal of Operational Research* 252.1, pp. 183–190.

Kou, S, Peng, X, and Xu, X (2016). "EM Algorithm and Stochastic Control in Economics." *arXiv.org*. arXiv: `1611.01767,2016.`.

Lai, Y. and Spanier, J. (1998). "Applications of Monte Carlo/quasi-Monte Carlo methods in finance: Option pricing." *Proceedings of a conference held at the Claremont Graduate Univ., CA, USA*. Springer Berlin Heidelberg.

Longstaff, F. A. and Schwartz, E. S. (2001). "Valuing American Options by Simulation: A Simple Least-Squares Approach." *Review of Financial Studies* 14.1, pp. 113–147.

Luo, X. and Shevchenko, P. V. (2014). "Fast and simple method for pricing exotic options using Gauss–Hermite quadrature on a cubic spline interpolation." *Journal of Financial Engineering* 1.04, p. 1450033.

McDonald, R. and Siegel, D. (1986). "The Value of Waiting to Invest." *The Quarterly Journal of Economics* 101.4, pp. 707–727.

Merton, R. C. (1973). "Theory of Rational Option Pricing." *The Bell Journal of Economics and Management Science*, pp. 141–183.

Mordecki, E. (2002). "Optimal stopping and perpetual options for Lévy processes." *Finance and Stochastics* 6.4, pp. 473–493.

Moreno, M. and Navas, J. F. (2003). "On the robustness of least-squares Monte Carlo (LSM) for pricing American derivatives." *Review of Derivatives Research* 6.2, pp. 107–128.

Niederreiter, H. and Shiue, P. J. (2012). "Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing." *Proceedings of a conference at the University of Nevada, Las Vegas, Nevada, USA*. Springer Science & Business Media.

Pagès, G., Pironneau, O., and Sall, G. (2016). "The parareal algorithm for American options." *Comptes Rendus Mathematique* 354.11, pp. 1132–1138.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Rasmussen, N. S. (2005). "Control Variates for Monte Carlo Valuation of American Options." *Working Paper*.

Rodrigues, A. and Rocha Armada, M. J. (2006). "The Valuation of Real Options with the Least Squares Monte Carlo Simulation Method." *SSRN Electronic Journal*.

Rogers, L. (2002). "Monte Carlo valuation of American options." *Mathematical Finance* 3.12, pp. 271–286.

Roll, R. (1977). "An analytic valuation formula for unprotected American call options on stocks with known dividends." *Journal of Financial Economics* 5.2, pp. 251–258.

Rust, J. (1997). "Using randomization to break the curse of dimensionality." *Econometrica: Journal of the Econometric Society* 65.3, pp. 487–516.

Stentoft, L (2013). "American option pricing using simulation with an application to the GARCH model." *Handbook of Research Methods and Applications in Empirical Finance*. Edward Elgar, pp. 114–147.

Stentoft, L. (2004). "Assessing the Least Squares Monte-Carlo Approach to American Option Valuation." *Review of Derivatives Research* 7.2, pp. 129–168.

Stentoft, L. (2014). "Value function approximation or stopping time approximation: a comparison of two recent numerical methods for American option pricing using simulation and regression." *The Journal of Computational Finance* 18.1, pp. 65–120.

Sullivan, M. A. (2000). "Valuing American put options using Gaussian quadrature." *The Review of Financial Studies* 13.1, pp. 75–94.

Tan, K. S. and Boyle, P. P. (2000). "Applications of randomized low discrepancy sequences to the valuation of complex securities." *Journal of economic dynamics and control* 24.11, pp. 1747–1782.

Team, R. C. et al. (2015). *fOptions: Rmetrics - Pricing and Evaluating Basic Options*. R package version 3022.85.

Tilley, J. A. (1993). "Valuing American options in a path simulation model." *Transactions of the Society of Actuaries* 45, pp. 83–104.

Trigeorgis, L. (1996). *Real options: Managerial flexibility and strategy in resource allocation*. MIT Press.

Tsitsiklis, J. N. and Van Roy, B. (2001). “Regression methods for pricing complex American-style options.” *IEEE Transactions on Neural Networks* 12.4, pp. 694–703.

Ueberhuber, C. W. (1997). *Numerical Computation 2*. Methods, Software, and Analysis. Springer Science & Business Media.

Wald, A. and Wolfowitz, J (1949). “Bayes solutions of sequential decision problems.” *Proceedings of the National Academy of Sciences of the United States of America* 35.2, pp. 99–102.

Warnes, G. R., Bolker, B., and Lumley, T. (2015). *gtools: Various R Programming Tools*. R package version 3.5.0.

Wickham, H. (2009). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN: 978-0-387-98140-6.

Williams, D. (1991). *Probability with Martingales*. Cambridge University Press.

Wilmott, P., Dewynne, J., and Howison, S. (1993). *Option Pricing: Mathematical Models and Computation*. Oxford Financial Press.

Zhang, Q and Guo, X (2004). “Closed-Form Solutions for Perpetual American Put Options with Regime Switching.” *SIAM Journal on Applied Mathematics* 64.6, pp. 2034–2049.

Zhou, Y. (2004). “On the existence of an optimal regression complexity in the least-squares Monte-Carlo (lsm) framework for option pricing.” *Proceedings to 39th Actuarial Research Conference, Society of Actuaries*.

Final Version as of November 24, 2017.