UPPSALA
UNIVERSITET

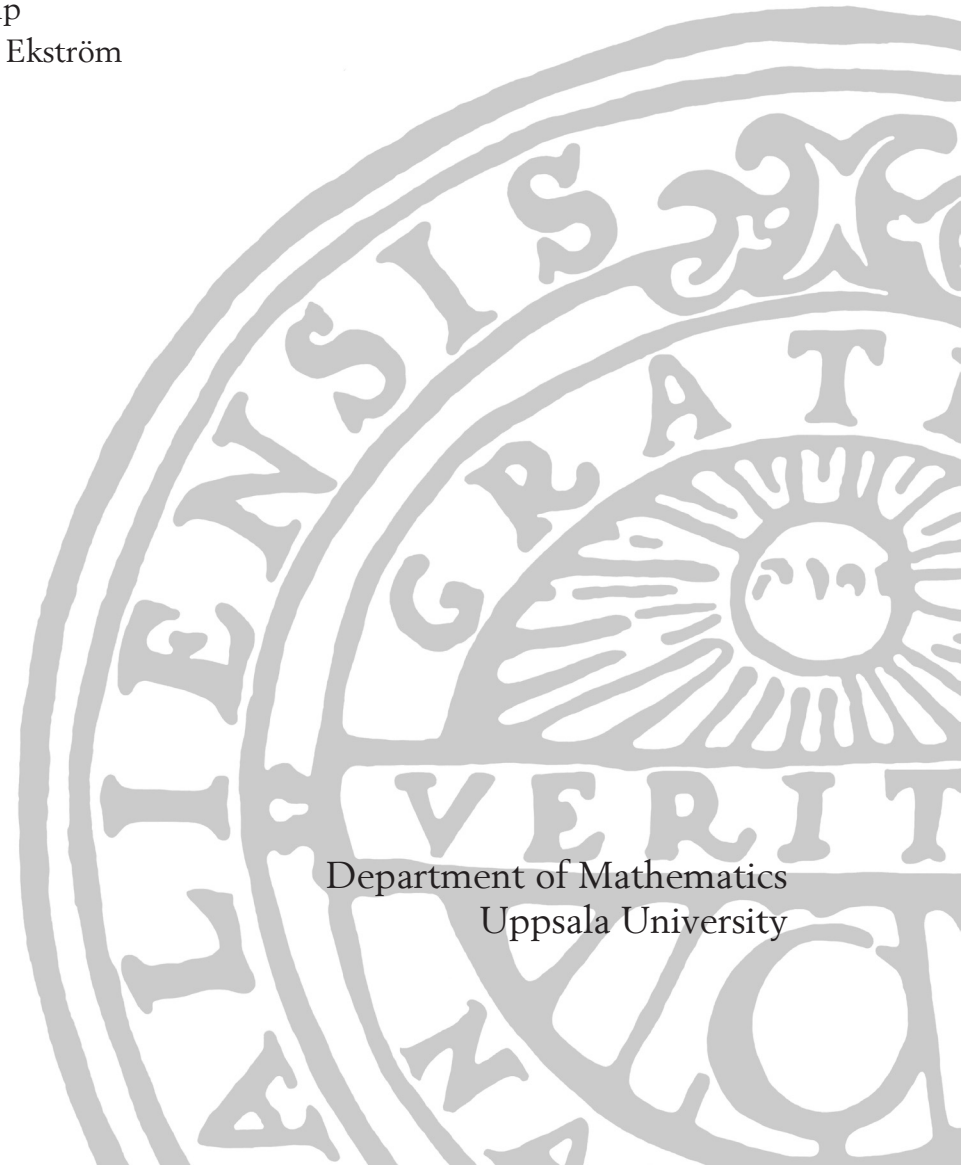
U.U.D.M. Project Report 2011:4

Monte Carlo Methods in American Put Option Pricing

Hady Ahmady Phoulady

Examensarbete i matematik, 30 hp
Handledare och examinator: Erik Ekström

Mars 2011



Department of Mathematics
Uppsala University

I would like to dedicate this thesis to my loving parents who encouraged and supported me through my studies by any means.

Acknowledgements

And I would like to thank my supervisor Erik Ekström who suggested me a subject of my interest, guided and helped me kindly through my thesis. Also my teachers and amongst them, especially professor Maciej Klimek, who gave a good view to me of what Financial Mathematics is about and made me interested in it by his great way of teaching!

Abstract

An *American put option* gives its owner the right to sell an underlying asset at a predetermined price anytime in a time interval, from the beginning of the contract to the maturity time. There is no explicit formula for the value of an American put option but there are several methods to approximate it (rather than *Finite Difference* methods, there exist *Monte Carlo* methods which are more easily implemented and also usually faster). In this thesis three of well-known Monte Carlo methods are explained and also a couple of more intuitive methods are suggested. They are implemented and the results and computational time of them are checked and compared for 20 different American put options.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

This thesis is discusses three recent Monte Carlo methods [2;4;6] for pricing American options with most basic definitions and formulations from a book [3].

Paul Glasserman's book [3], *Monte Carlo Methods in Financial Engineering*, is used for basic definitions, formulations and some tips for approximations of values and stopping rules.

Then three different Monte Carlo methods are discussed: *Broadie* and *Glasserman*'s paper [2], *Francis A. Longstaff* and *Eduardo S. Schwartz*'s paper [4] and *L. C. G. Rogers*' paper [6]. We discuss them briefly, implement and compare them together and also suggest a couple of new ideas (Chapters 5 and 6).

1.2 A Simple Example

Amongst all the methods to value American options (and of course many other arising problems), Monte Carlo methods are some of the most important, easiest to implement and to some extent fastest methods. Monte Carlo methods are based on trying to estimate a random variable by generating several random trajectories which lead to different value for the variable. Then by the different values you can have an estimation of that random variable.

Consider that you want to have an estimation of the temperature of a certain

day in year. One intuitive way is to get an average on the temperature of the same day in last years. Then without having any information about the temperature of other days, you can give an estimation of the temperature in that day. Although the temperature in that day can be different from what you estimated, you can still be hopeful that you have some kind of reliable estimation.

Another easy and famous example of Monte Carlo methods in mathematics is the integral value of a function f on an interval, say the unit interval[3]. Suppose that f is integrable on the unit interval and we want to compute

$$\alpha = \int_0^1 f(x)dx.$$

It can be considered as an expectation $\mathrm{E}[f(U)]$ where U is uniformly distributed over the unit interval. A Monte Carlo method suggests that we draw n random points from the unit interval uniformly, called U_i's, and calculate the average:

$$\hat{\alpha}_n = \frac{1}{n}\sum_{i=1}^{n} f(U_i).$$

$\hat{\alpha}_n$ is an estimation of $\mathrm{E}[f(U)]$ and therefore of α. The strong law of large numbers tells us that also

$$\hat{\alpha}_n \to \alpha$$

with probability 1 as $n \to \infty$.

Coming back to option valuation, Monte Carlo methods helps us in a similar way.

A simple example of using Monte Carlo methods is valuing regular European options. Phelim Boyle did it for the first time in 1977. The approach is easy and intuitive: We simulate n random trajectories using time discretization. Then we compute the value of the option in each trajectory and get an average on all of them, leading to an estimated value of the European option.

Monte Carlo methods are also very efficient for pricing many path dependent options in comparison to other different methods like Finite Difference methods. But using Monte Carlo methods for pricing some options like American options is not as one word as the approach explained above. An early attempt for using Monte Carlo methods for pricing American options was done by *Longstaff*

and *Schwartz*[4]. Their method, called *Least-Squares Method* (LSM), is comparable with other Finite Difference Methods in pricing American options. Apart from *Longstaff* and *Schwartz* Least-Squares approach, *Rogers* suggested another method in his paper in 2002.[6] We will discuss these methods later.

In the non-dividend case, we only consider American put options. The reason for not studying American Call options is that they are actually equivalent to European call options when no dividend is being paid. Several reasons exist for this: If sometime (before the maturity) the option is in the money and we may decide to exercise the option, instead of exercising it we can keep the option and by doing so we still get the benefit of stock price fluctuations upwards and also a protection against downwards stock price. Even by keeping the option and investing the strike price, K, with a risk free interest rate, we also get the profit of its interest. So we have at least two advantages by keeping the option rather than exercising: Protection and Interest profit.

1.3 Problem Formulation

In this thesis, we focus on non-dividend paying option. We start by formulating the problem[3]. Our pricing problem can be formulated by specifying a process $U(t)$, $0 \leq t \leq T$. We can consider it as discounted payoff from exercising the option at time t. Now our problem is to find

$$\sup_{\tau \in T^*} \mathrm{E}[U(\tau)].$$

Here T^* is the set of stopping times before the maturity time, T, that we can exercise the option.

We denote the stock price process by $S(t)$. We also denote the payoff of exercising at time t by $\tilde{h}(S(t))$. Of course it is non-negative and for example in the put options case, it is equal to $(K - S(t))^+$. So if we consider the general case for risk free interest rate: $r(t) = \{r(t), 0 \leq t \leq T\}$, we actually want to calculate

$$\sup_{\tau \in T^*} \left[e^{-\int_0^\tau r(u)du} \tilde{h}(S(\tau)) \right].$$

If the risk free interest rate is constant, then the problem changes to calculate

$$\sup_{\tau \in T^*} \left[e^{-r\tau} \tilde{h}(S(\tau)) \right],$$

and in the case of a constant risk free interest rate and an American put option, we are to find

$$\sup_{\tau \in T^*} \left[e^{-r\tau} (K - S(\tau))^+ \right]. \tag{1.1}$$

If we are able to predict an *optimal exercise boundary* like $b^*(t)$ (see figure 1.1), $0 \leq t \leq T$, then

$$\tau^* = \inf \{ t \geq 0 : S(t) \leq b^*(t) \}, \tag{1.2}$$

can give the supremum we are looking for, in (1.1).

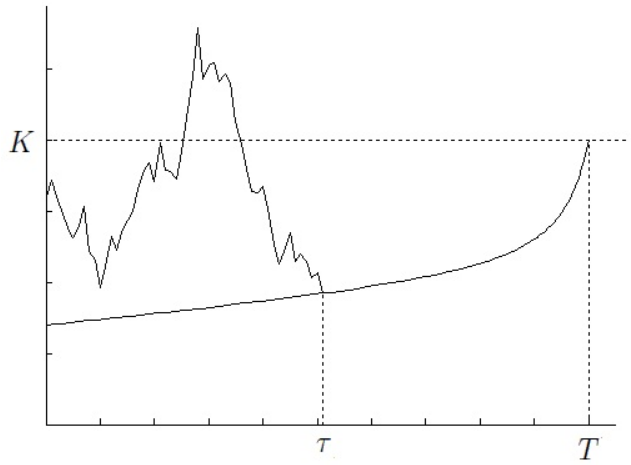


Figure 1.1: The optimal exercise boundary and a stock price trajectory and its corresponding τ^*, the first time it goes below b^*

The way of writing (1.1) and (1.2) and the fact that we used $(K - S(t))^+$ as payoff instead of simply using $(K - S(t))$ gives us the idea that if τ^* be equal to T, the option expires worthless.

1.3.1 Recursive Based Methods

Monte Carlo methods use *time discretization* for pricing American put options. So in fact they consider that exercising the option is possible only on a finite number of times. These kind of options are called *Bermudan*. So we actually are discussing Monte Carlo methods for *Bermudan* put options.

We divide the whole time interval (from 0 to T) into n equal intervals, resulting in different time values $t_0 = 0, t_1, \cdots, t_n = T$. We denote the discount factor from time t_{i-1} to time t_i, by $D_{i-1,i}$[3]. As we already considered two different cases, this value is equal to $\exp(-r\Delta t)$ in constant interest rate case where $\Delta t = \frac{T}{n}$ and is equal to $\exp\left(-\int_{t_{i-1}}^{t_i} r(u)du\right)$ when the interest rate is variable.

Instead of writing $S(t_i)$ we simply write S_i. If we did not exercise the option by time t_i, denote the value of option at time t_i by $\tilde{V}_i(s)$, given $S_i = s$. So we are actually interested in $\tilde{V}_0(S_0)$.

$\tilde{V}_i$'s are determined recursively by[3]

$$\tilde{V}_n(s) = \tilde{h}_n(s) \tag{1.3}$$

$$\tilde{V}_{i-1}(s) = \max\left\{\tilde{h}_{i-1}(s), \mathrm{E}\left[D_{i-1,i}\tilde{V}_i(S_i) \mid S_{i-1} = s\right]\right\} \tag{1.4}$$

We can simplify (1.3) and (1.4) by removing the discount factor and get the following equations

$$V_n(s) = h_n(s) \tag{1.5}$$

$$V_{i-1}(s) = \max\left\{h_{i-1}(s), \mathrm{E}\left[V_i(S_i) \mid S_{i-1} = s\right]\right\} \tag{1.6}$$

It also contains the problem defined by (1.3) and (1.4) as a special case. The problem defined by (1.5) and (1.6) is actually the general form of the problem defined by (1.3) and (1.4).

Indeed if we put

$$h_i(s) = D_{0,i}\tilde{h}_i(s) \text{ for } i = 1, 2, ..., n$$

and

$$V_i(s) = D_{0,i}\tilde{V}_i(s) \text{ for } i = 0, 1, ..., n,$$

equation (1.5) will change to

$$D_{0,n}\tilde{V}_n(s) = D_{0,n}\tilde{h}_n(s), \tag{1.7}$$

and equation (1.6) will change to

$$D_{0,i-1}\tilde{V}_{i-1}(s) = \max\left\{D_{0,i-1}\tilde{h}_{i-1}(x), \mathrm{E}\left[D_{0,i}\tilde{V}_i(S_i) \mid S_{i-1} = s\right]\right\} \tag{1.8}$$

where with dividing (1.7) sides by $D_{0,n}$ and (1.8) sides by $D_{0,i-1}$ we will get (1.3) and (1.4).

In theory, using (1.5) and (1.6) is slightly simpler than using (1.3) and (1.4) because it does not have a discount factor to worry about. But in practice it is usually easier to use (1.3) and (1.4) because we only have to multiply the expectation by discount factor and there is no need to define new h_i's as below. In implementations in Appendices, (1.3) and (1.4) are used.

Most of Monte Carlo methods, including Least Squares Methods (which will be discussed later), try to price the American options based on dynamics defined in (1.5) and (1.6).

An exact method (no exact method exists by now!) will solve the dynamic programming problem defined by (1.3) and (1.4) or by (1.5) and (1.6) and get the true value of option, $V_0(S)$. But as it is not easy to determine the true value of expectation in (1.4) or (1.6), Monte Carlo methods just try to approximate it and thus estimate the true value which we will denote by $\hat{V}_i(s)$.

1.4 Continuation and Stopping Regions

Still there exists no algorithm to compute the *optimal stopping time*. If we were able to have the exact optimal stopping time, one good Monte Carlo Method for pricing a Bermudan option would be as below.

We simulate stock price trajectories n times. For each of them we simulate it until it hits the optimal stopping curve, and then we discount the payoff to the starting time, 0. If a trajectory had not been exercised until the maturity time, T, then the option expires worthless (cause the optimal stopping time at

maturity is equal to K).

By getting an average of all of the values that we got, we will have an estimation of the price of the option.

But unfortunately this is not possible as we do not have access to the exact optimal stopping time (except in a few special cases of some option, but not for the American put option which is our main interest)! At this point, one can use different algorithms for pricing the option in two ways. One way is just to focus the on option value, according to (1.5) and (1.6). Another way is to solve the problem using an estimated optimal stopping time, which gives the stopping rules and exercising regions.

1.4.1 Stopping Rules

Any *stopping time* τ results in a value

$$V_0^{(\tau)}(S_0) = \mathrm{E}[h_\tau(S_\tau)]^{[3]}.$$

On the other hands, if we assign any arbitrary values, $\hat{V}_i(s)$, to the option on each of the time states (with the condition that $\hat{V}_n = h_n$), for a new price trajectory, we can make a stopping rule $\hat{\tau}$ as

$$\hat{\tau} = \min\left\{i \in \{1, \cdots, n\} : h_i(S_i) \geq \hat{V}_i(S_i)\right\}. \tag{1.9}$$

So according to above, stopping regions, for every i, $1 \leq i \leq n$, is

$$\left\{s : h_i(s) \geq \hat{V}_i(s)\right\}.$$

This means that for every exercise date, t_i, we exercise the option if the stock has a value s, which satisfies the above equation. The *continuation region* is the completion of above region, namely

$$\left\{s : h_i(s) < \hat{V}_i(s)\right\}.$$

Literally saying, we exercise the option when the payoff from exercising, $h_i(s)$ is higher than what we expect to gain if we continue (stopping region) and vice

versa (continuation region).

1.4.2 Continuation Values

The *continuation values* are the values of holding the option rather than exercising it[3]. We denote these values by $C_i(s)$s. Because of the Markov property of price trajectory, it can be understood that

$$C_i(s) = \mathrm{E}\left[V_{i+1}(S_{i+1}) \mid S_i = s\right],$$

for $i = 0, \cdots, n-1$.

We know that if we do not exercise the option, it expires worthless, so indeed we have $C_n(S_n) = 0$.

On the other hand if we put $C_n = 0$, we then will be able to define C_i's for i's less than n recursively as below

$$C_i(s) = \mathrm{E}\left[\max\left\{h_{i+1}(S_{i+1}), C_{i+1}(S_{i+1}) \mid S_i = s\right\}\right].$$

Like stopping rules, that they could be determined by any values for $\hat{V}_i$'s and also conversely it was possible to compute $\hat{V}_i$'s by any arbitrary stopping rules; here we can also compute V_i's at any time t_i, $i = 1, \cdots, n$, by having the continuation values, C_i's,

$$V_i(s) = \max\left\{h_i(s), C_i(s)\right\}. \tag{1.10}$$

There is a close relation with continuation values and stopping rules too. If we have approximations for C_i's, namely $\hat{C}_i$'s, then we can get approximations for stopping rules, $\hat{\tau}$,

$$\hat{\tau} = \min\left\{i \in \{1, \cdots, n\} : h_i(S_i) \geq \hat{C}_i(S_i)\right\}. \tag{1.11}$$

(1.9) and (1.11) are the same if we approximate for V_i's by approximations of C_i's according to the formula (1.10). More precisely we approximate V_i's by $\hat{V}_i(s) = \max\left\{h_i(s), \hat{C}_i(s)\right\}$.

1.5 Approximations

A perfect pricing of American options needs solving the problem determined by (1.5) and (1.6). But we can also try to solve it in a parametric class which reduces it to a finite dimensional optimization problem[3]. This parametric class can be defined on either the exercise region or stopping rules, as we have already seen that they are actually equivalent.

Even a rough (and not so much close to exact) approximation of the exercise boundary results in a good approximation of the value of the option[3]. The reason for this is that the option value is continuously differentiable across the boundary and therefore is not much sensitive to exact estimation of exercising boundary.

The idea suggests that first we specify a parametric class for example stopping rules. The expectation of value approximated with any stopping rule from the parametric class, V_0^θ, is biased low, meaning that V_0^θ is less than or equal to the true value, V_0, where usually strict inequality happens (because intuitively, the perfect stopping rule is not in our parametric class of stopping rules).

Now, we replicate the portfolio price trajectory n_1 times. We choose the stopping rule in the parametric class that produces the highest average value of all of these n_1 portfolios. This new average value, $\hat{V}_0^{\hat{\theta}}$, is biased high relative to V_0^θ. The high bias in $\hat{V}_0^{\hat{\theta}}$ relative to V_0^θ, may offset the low bias in V_0^θ relative to V_0; but we can not be sure of it. For solving this issue, one way is doing as follows.

We replicate the portfolio price trajectory n_2 times. For these new n_2 price trajectories, we get the average value obtained by the stopping rule $\hat{\theta}$. Since these new trajectories are independent of those that determined the stopping rule $\hat{\theta}$, then it is actually biased low relation to the true value, V_0.

A more general strategy of this instance has been used by *Broadie* and *Glasserman* for separating sources of high and low bias[2].

1.6 Different Kind of Methods

Before describing different kinds of methods, we discuss more about how can we properly use high and low biased estimations to get an interval containing the true value. If these two estimations converge to the true value then we can actually have a interval containing the true value with some certain probability and also as small as wanted.

Suppose that we have two respectively high and low biased estimators, $\hat{V}_n$ and $\hat{v}_n$[1], relative to true value, V_0, based on n true independent replications[3], meaning that

$$\mathrm{E}[\hat{V}_n] \geq V_0 \geq \mathrm{E}[\hat{v}_n].$$

Then suppose that for a specific n, we have H_n where

$$\left(\hat{V}_n - H_n, \hat{V}_n + H_n\right)$$

is a 95% confidence interval for $\mathrm{E}[\hat{V}_n]$[2]. Also suppose that for this n and some L_n,

$$\left(\hat{v}_n - L_n, \hat{v}_n + L_n\right),$$

is the confidence interval for $\mathrm{E}[\hat{v}_n]$.

Now

$$\left(\hat{v}_n - L_n, \hat{V}_n + H_n\right),$$

is a very interesting interval. In the sense that the true value that we are seeking for, V_0, is in this interval with probability at least 90% (and even if the estimators, $\hat{V}_n$ and $\hat{v}_n$, be symmetric around their mean, then this probability is at least 95%).

It shows that how we can make a confidence interval for the true value by having two biased (one high and one low) estimators. Also if the estimators converge to the true value as $n \to \infty$, then the length of the last interval shrinks to zero as $n \to \infty$ (because we can have as small H_n and L_n as we want by increasing n).

Using this idea, we can make desirable confidence intervals by using different

[1] Parameter n can consist of several different independent parameters, but for simplicity we wrote only a single parameter here.

[2] The value is in the interval with 95% probability.

(high and low) biased estimators. It means that by using them we can almost have the advantages of having an unbiased estimator!

The following sections we discuss briefly different kinds of methods and then we give an example of each of random tree, regression-based and stochastic mesh methods respectively in Chapters 2, 3 and 5.

1.6.1 Random Tree Methods

These methods try to estimate the value based on some generated random trees (in which the tree has the height equal to for example n and each internal nodes have $b \geq 2$ children) as shown in 1.2.

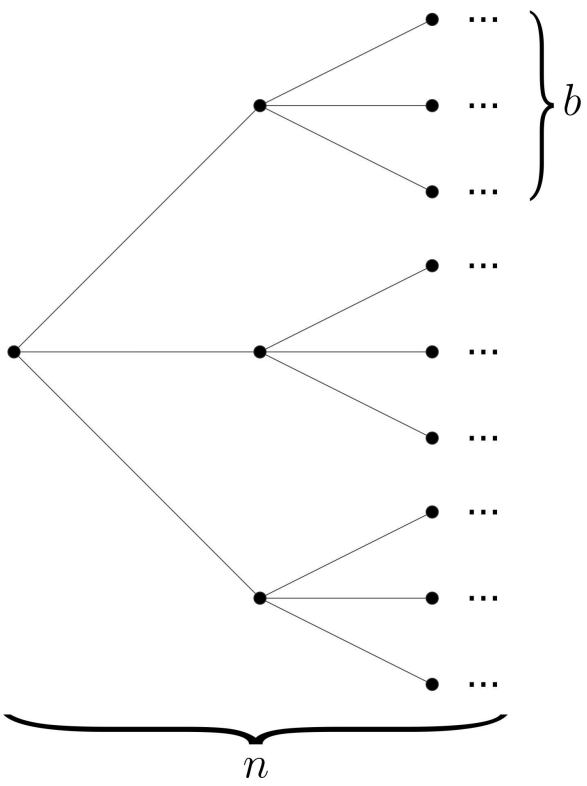


Figure 1.2: A sample of a random tree with height n where each node has either 0 (leaf) or b children

Note that the word *random* refers to random numbers (labels) that each node has, and not the structure on the tree.

Regularly for simulating a random trajectory of strike price from time 0 to maturity time, first we divide the time interval and by starting from the stock price at time 0, we randomly (according to the dynamics of the underlying asset's price) determine the stock price at next time step and we continue in this manner until maturity time (we describe it in more details later[1]) but here, we start from the root of the tree, labeling it with the initial stock price, S_0. Then we simulate the next stock price according to initial price independently b times, calling them $S_1^1, S_1^2, ..., S_1^b$, and we use them to label children of the root. From each of S_1^i's we simulate new stock price according to their parent's stock price. So for example $S_1^{i1}, S_1^{i2}, ..., S_1^{ib}$ will be simulated based on S_1^i independently and they will be used to label children of the node with label S_1^i. We continue until we reach leaves in the tree (after n steps). So labels of leaves will be like $S_n^{i_1 i_2 \cdots i_n}$ ($1 \leq i_j \leq b$ for all $1 \leq j \leq n$).

The result of this process will be a random tree in which each of the paths from the root to any arbitrary leaves will be a Markov Chain of stock price from time 0 to time of maturity (the equivalent simulated random price trajectories in other methods). Each of these paths are consisting of nodes with labels $S_0, S_1^{i_1}, S_2^{i_1 i_2}, ..., S_n^{i_1 i_2 \cdots i_n}$.

This tree will be the data we use to estimate the true value of the option. The sample method[2] that we will explain in Chapter 2, works backward from leaves to root trying to estimate the desired value. In that method they (authors) start from maturity time. For each of the leaves the value of the option will be the (discounted) value of payoff by exercising (according to 1.5). Moving backwards, with the values computed in those leaves that have the same parent the value of the option at the parent nod according to (1.6) will be approximated. So the value of each leaf with label $S_n^{i_1 i_2 \cdots i_n}$ will be equal to $(K - S_n^{i_1 i_2 \cdots i_n})^+$. Then for computing the value of option in nod with label $S_n^{i_1 i_2 \cdots i_{n-1}}$ we use the values of option at nodes $S_n^{i_1 i_2 \cdots i_{n-1} 1}, S_n^{i_1 i_2 \cdots i_{n-1} 2}, \cdots$ and $S_n^{i_1 i_2 \cdots i_{n-1} b}$. These values will be used to compute the (discounted) continuation value of option at nod $S_n^{i_1 i_2 \cdots i_{n-1}}$. Then we simply make the comparison in (1.6) and decide whether to exercise or continue. The process on using those children nod values to approximate the continuation value for the parent nod is essential. By different processes we can

[1] In fact in this way we will have a Markov Chain of the stock price at different time steps.

get different estimators.

Broadie and *Glasserman* after describing the algorithm of generating the random tree, have suggested two slightly different processes for the part of approximating the continuation values which use the same generated random tree and result in a high and a low biased estimator of the option value (they also prove that those estimators are converging to the true option value as $n \to \infty$.

The disadvantage of this method is that the computational time of generating the random tree is increasing exponentially by increasing n so it is good and practically possible to use when n is relatively small. About the memory requirement, although at first it may seem that the required size of memory for this method is exponentially growing relative to n, but it can be easily proved that we only need to save at most $nb + 1$ stock price at each time of running the algorithm to estimate the value (though we would not have the whole tree in the end for sure).

1.6.2 Regression-Based Methods

In this kind of methods, at first m price trajectories are being simulated and then approximating continuation values is done by making a linear combination of current state known functions (for example the stock price and square stock price of current state) and uses regression to find the best coefficients of them.

These methods are trying to solve the problem of finding the true value of option by solving (or actually just approximating the answers of) (1.5) and (1.6). So in these methods we again put the value of option at the final states (states in maturity time) equal to the payoff of exercising (according to (1.5)). It means that we put $\hat{V}_n(S_{nj}) = h_n(S_{nj})$ for each $j = 1, 2, ..., m$ (we divided the time interval into n time steps and simulated the price trajectory m times).

After the initial step, we move backward, trying to approximate the value at states S_{ij} according to (1.6). Suppose that currently we are in state ij, having the stock price of S_{ij}. It means that by now we approximated the value of the option in all states $i'j'$ where $i' = i+1, i+2, ..., n$ and $j' = 1, 2, ..., m$. Now suppose that we have known functions $\psi_1^{ij}, \psi_2^{ij}, ..., \psi_k^{ij}$ of current states. There may exist some true values of $\beta_1, \beta_2, ..., \beta_k$ so that the exact continuation value, $C_i(S_{ij})$, is equal

to

$$\sum_{r=1}^{k} \beta_r \psi_r^{ij},$$

but also we may be able to get some approximated values of them by regression, like $\hat{\beta}_1, \hat{\beta}_2, ..., \hat{\beta}_k$ resulting in an approximation of continuation value at current state as

$$\hat{C}_i(S_{ij}) = \sum_{r=1}^{k} \hat{\beta}_r \psi_r^{ij}.$$

Then according to (1.6) we put

$$\hat{V}_i(S_{ij}) = \max\left\{h_i(S_{ij}), \hat{C}_i(S_{ij})\right\}. \tag{1.12}$$

In the end we can approximate the true value as

$$\hat{V}_0 = \frac{\hat{V}_1(S_{11}) + \hat{V}_1(S_{12}) + ... + \hat{V}_1(S_{1m})}{m}.$$

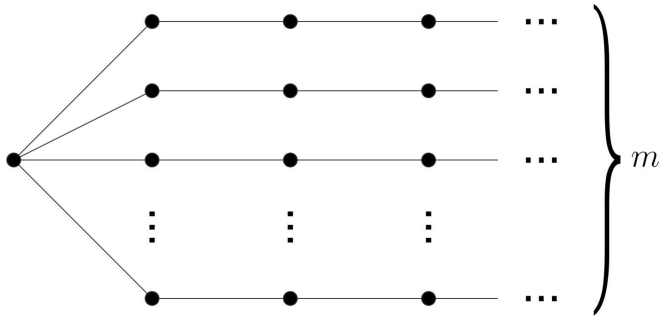
The process described above is the general approach in regression-based methods. Differences between different regression-based methods usually come from the way the regression is being done. In the regression-based method that we discuss later in Chapter 3, *Longstaff* and *Schwartz* use a slightly different approach to estimate the values. Instead of (1.12) they do it as below[3]

$$\hat{V}_i(S_{ij}) = \begin{cases} h_i(S_{ij}) & h_i(S_{ij}) \geq \hat{C}_i((S_{ij}) \\ \hat{V}_{i+1}(S_{i+1,j}) & h_i(S_{ij}) < \hat{C}_i(S_{ij}) \end{cases} \tag{1.13}$$

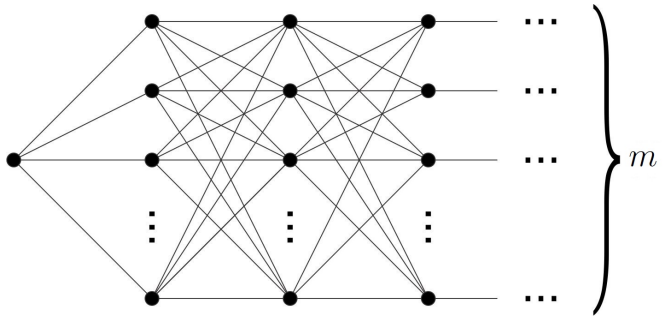
The only difference between (1.12) and (1.13) is when $h_i(S_{ij})$ is less than $\hat{C}_i(S_{ij})$. Also in each step, they take the regression only on the options that are in the money. Their method will be discussed more later.

1.6.3 Stochastic Mesh Methods

In these methods the trajectories simulation are like what is being done in regression-based methods and approximating continuation values are somehow like random tree methods.



(a) m price trajectories



(b) Interactions between different states

Figure 1.3: Simulating and approximating continuation values process diagram in stochastic mesh methods

In Figure 1.3 the simulations and approximation process is shown with two diagrams. In Figure 1.3a the way of trajectories simulation is shown. It shows that each trajectory is actually a Markov chain in which states are only dependent on their previous states in the same trajectory. So states from different trajectories, or from states more than one states after or before a state, are independent. Figure 1.3b shows which states help us to approximate the continuation values of any arbitrary state (in each states, continuation value approximation is dependent on those states that has a line to it and are in the next step).

In random tree methods, the process of approximating continuation values of a state is done according to states in next step which are only successors of it. But in stochastic mesh methods all the states in the next step are interfering.

The general approach in stochastic mesh methods is as follows.

The option value in final step states are computed as (1.5): $\hat{V}_n(S_{nj}) = h_n(S_{nj})$ for $j = 1, 2, ..., m$ where in an American put option they are equal to $(K - S_{nj})^+$. Recursively, for previous steps consider that we want to approximate

the continuation value in a state (i, j) with price S_{ij}. The diagram in Figure 1.3b shows that it can be dependent on option values in states $(i+1, 1), (i+1, 2), ..., (i+1, m)$ (it actually can be dependent on all of them or just a number of them). Now if we define some kind of *weight function* on those option values and we define the approximation as (1.14), then we actually have a stochastic mesh method. In general, this weight function can be like a function as W_{ij}^k, where $i = 1, 2, ..., n-1$ and $j, k = 1, 2, ..., m$. For a specific i, j and k, W_{ij}^k is the weight on the line between states (i, j) and $(i+1, k)$.

$$\hat{C}_i(S_{ij}) = \frac{\sum_{r=1}^{m} W_{ij}^r \hat{V}_{i+1}(S_{i+1,r})}{m}. \tag{1.14}$$

According to (1.6), it means that in these methods, the option value approximation will be

$$\hat{V}_i(S_{ij}) = \max\left\{ h_i(S_{ij}), \frac{\sum_{r=1}^{m} W_{ij}^r \hat{V}_{i+1}(S_{i+1,r})}{m} \right\}$$

for $i = 1, 2, ..., n-1$ and $j = 1, 2, ..., m$.

Let us define the mesh estimator as

$$\hat{V}_0 = \frac{\sum_{j=1}^{m} \hat{V}_1(S_{1j})}{m}.$$

With this definition it is easy to see that the method which will be discussed in Chapter 5 is indeed a stochastic mesh method. In that method, the weight function is actually simply equal to 1 for all states in next step who has a successor in current step that has a price *close* to current state's stock price and 0 for all other steps.

Glasserman describes a detailed construction of a weight function[3] which is not going to be discussed here, but he also states three initial conditions on the mesh construction and weight functions which will be explained below. He then proves that if a mesh estimator satisfies those three conditions is indeed biased high.

Before stating the conditions, let

$$\mathbf{S}_i = (S_{i1}, S_{i2}, ..., S_{im}),$$

which denotes the *mesh state* at step i. Put $\mathbf{S}_0 = S$.

We assumed that each of those simulated trajectories in Figure 1.3a are actually a Markov chain. The first condition states that it has to be the case as follows.

M1: $\{\mathbf{S}_0, \mathbf{S}_1, ..., \mathbf{S}_{i-1}\}$ and $\{\mathbf{S}_{i+1}, \mathbf{S}_{i+2}, ..., \mathbf{S}_n\}$ are independent given $\mathbf{S}_i$ for $i = 1, 2, ..., n$.

M2: W_{ij}^k is a deterministic function of $\mathbf{S}_i$ and $\mathbf{S}_{i+1}$ for $i = 1, 2, ..., n-1$ and $j, k = 1, 2, ..., m$.

(**M**2) says that each value of weight function should be dependent to at most states in the current step or the next step.

M3: We have

$$C_i(S_{ij}) = \frac{\sum_{r=1}^{m} \mathrm{E}\left[W_{ij}^r \hat{V}_{i+1}(S_{i+1,r}) \mid \mathbf{S}_i\right]}{m},$$

for $i = 1, 2, ..., n-1$ and $j = 1, 2, ..., m$.

Having (**M**3) means that in any arbitrary state in steps $1, 2, ..., n-1$, *if* we have the true option values in states at next step, then our expectation from the approximated continuation value in current state is the true continuation value.

Broadie and *Glasserman*[5] prove that the mesh estimator $\hat{V}_0$ is converging when $\mathbf{S}_1, \mathbf{S}_2, ..., \mathbf{S}_n$ are independent of each other, $S_{i1}, S_{i2}, ...$ are i.i.d. for each i, and each weight W_{ij}^k is a function only of S_{ij} and $S_{i+1,k}$.

Also for the error in the mesh estimator, *Avramidis* and *Matzinger*[1] derived a probabilistic upper bound for a type of dependence structure that fits within conditions (**M**1) and (**M**2). Then using this bound they prove convergence as $m \to \infty$.

Chapter 2

A Random Tree Method

Broadie and *Glasserman* in their paper [2] present a random tree method with an example in pricing a American *call* option. It will be described here, and it will be implemented and used to price an American *put* option.

2.1 Approach

The approach to simulate and generate the random tree is as explained in section 1.6.1. Note that when it says that the tree has height n, it means that we are actually dividing the time interval into n steps.

Suppose that we generated the random tree. The way of processing this random tree can result into high or low biased estimators for the option value.

2.1.1 High Estimator

We start from the last step, at maturity time. By (1.5), we put the value of the option at that time equal to the payoff by exercising. Moving backwards, to approximate the continuation value for each state, we compute the average of value of next step states who are successors of current state. In the sense that if we are in a state $(i; j_1, j_2, \cdots, j_i)$ with price $S_i^{j_1 j_2 \cdots j_i}$ trying to approximate

$C_i(S_i^{j_1 j_2 \cdots j_i}), i < n$, we put

$$\hat{C}_i(S_i^{j_1 j_2 \cdots j_i}) = \frac{1}{b} \sum_{j=1}^{b} \hat{V}_{i+1}(S_{i+1}^{j_1 j_2 \cdots j_i j})$$

So according to (1.6) we put

$$\hat{V}_i(S_i^{j_1 j_2 \cdots j_i}) = \max \left\{ h_i(S_i^{j_1 j_2 \cdots j_i}), \hat{C}_i(S_i^{j_1 j_2 \cdots j_i}) \right\}$$

This is a simple approach to obtain a high estimator. An example is shown in Figure 2.1.

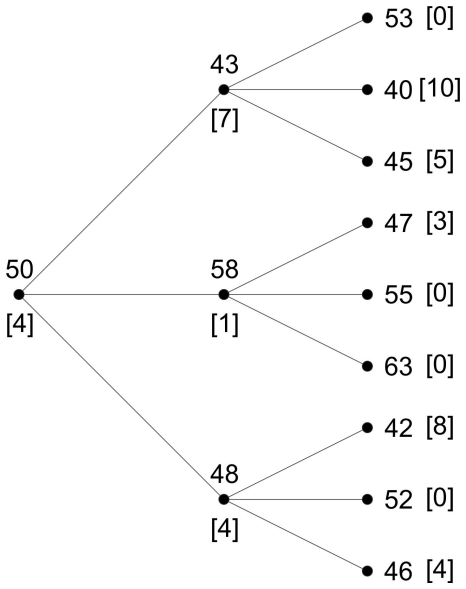


Figure 2.1: The process on the random tree to get a high estimator for a put option with stock and strike price equal to 50 (discount factor considered to be 1). In each state, labels are stock price and estimated option value (in bracket)

It is easy to prove by induction that this high estimator is indeed biased high in relative to the true option value.

We want to prove that

$$\mathrm{E}\left[\hat{V}_i(S_i^{j_1 j_2 \cdots j_i})\right] \geq V_i(S_i^{j_1 j_2 \cdots j_i}).$$

The induction base would be when $i = n$, where actually for all s, $\hat{V}_n(s) =$

$h_n(s) = V_n(s)$. Now we show that if above inequality holds for $i+1$ it also holds for i.

$$\begin{aligned}
\mathrm{E}\left[\hat{V}_i(S_i^{j_1 j_2 \cdots j_i})\right] &= \mathrm{E}\left[\max\left\{h_i(S_i^{j_1 j_2 \cdots j_i}), \hat{C}_i(S_i^{j_1 j_2 \cdots j_i})\right\}\right] \\
&\geq \max\left\{h_i(S_i^{j_1 j_2 \cdots j_i}), \mathrm{E}\left[\hat{C}_i(S_i^{j_1 j_2 \cdots j_i})\right]\right\} \\
&= \max\left\{h_i(S_i^{j_1 j_2 \cdots j_i}), \mathrm{E}\left[\frac{1}{b}\sum_{j=1}^{b} \hat{V}_{i+1}(S_{i+1}^{j_1 j_2 \cdots j_i j}) \mid S_i^{j_1 j_2 \cdots j_i}\right]\right\} \\
&= \max\left\{h_i(S_i^{j_1 j_2 \cdots j_i}), \mathrm{E}\left[\hat{V}_{i+1}(S_{i+1}^{j_1 j_2 \cdots j_i j}) \mid S_i^{j_1 j_2 \cdots j_i}\right]\right\} \\
&\geq \max\left\{h_i(S_i^{j_1 j_2 \cdots j_i}), \mathrm{E}\left[V_{i+1}(S_{i+1}^{j_1 j_2 \cdots j_i j}) \mid S_i^{j_1 j_2 \cdots j_i}\right]\right\} \\
&= V_i(S_i^{j_1 j_2 \cdots j_i})
\end{aligned}$$

The first inequality comes from Jensen's inequality.

With a similar induction it is possible to prove that this high estimator does converge in probability (and in norm) as $b \to \infty$ to the true value[3].

2.1.2 Low Biased Estimator

Suppose that we want to calculate

$$\max\{h, \mathrm{E}[c]\}$$

Now suppose that we replicate c, b times. The fact that

$$\mathrm{E}[\max\{h, \bar{c}\}] \geq \max\{h, \mathrm{E}[\bar{c}]\} = \max\{h, \mathrm{E}[c]\}$$

produced the high bias in previous high estimator. There, we actually should have calculated the maximum between the payoff of exercising and the expected continuation payoff, and instead we calculated the maximum between the payoff of exercising and the average continuation values of b replications. So if we are able to remove this bias, we can probably get an unbiased estimator, or maybe

an estimator that is biased low. There are several ways to do that and one of them is as follows.

To keep it simple let us explain it on the previous example. Partition b replicates of c into two disjoint groups and call their sample means $\bar{c}_1$ and $\bar{c}_2$. Instead of directly calculating

$$v = \max\{h, \mathrm{E}[c]\},$$

let us calculate

$$\hat{v} = \begin{cases} h, & \text{if } h \geq \bar{c}_1; \\ \bar{c}_2, & \text{otherwise.} \end{cases} \tag{2.1}$$

The way of calculating $\bar{v}$ in our problem is that for calculating the value of the option in an arbitrary state, like $(i; j_1, j_2, ..., j_i)$, we use some of its successors, for example states $(i+1; j_1, j_2, ..., j_i, 1), (i+1; j_1, j_2, ..., j_i, 2), ...(i+1; j_1, j_2, ..., j_i, j)$ and calculate the mean of their option values for $\bar{c}_1$ and use the others, $(i+1; j_1, j_2, ..., j_i, j+1), (i+1; j_1, j_2, ..., j_i, j+2), ...(i+1; j_1, j_2, ..., j_i, b)$ for $\bar{c}_2$. Then we make the comparison between $h_i(S_i^{j_1 j_2 \cdots j_i})$ and $\bar{c}_1$. If $h_i(S_i^{j_1 j_2 \cdots j_i})$ was greater we exercise and put the estimated value equal to it and if not, we put the estimated value equal to $\bar{c}_2$.

Now we prove that $\hat{v}$, defined in (2.1) is indeed biased low:

$$\mathrm{E}[\hat{v}] = P(\bar{c}_1 \leq h) \cdot h + (1 - P(\bar{c}_1 \leq h)) \cdot \mathrm{E}[c] \leq \max\{h, \mathrm{E}[c]\}.$$

The way of partitioning replications into two disjoint subsets can result in different estimators, which of course all of the are biased low. A simple example is that we use the first replication for calculating $\bar{c}_1$ and the rest for calculating $\bar{c}_2$. A sample of this process is shown in Figure 2.2 where actually for calculating $\bar{c}_1$ we used the first replication and used the others for calculating $\bar{c}_2$.

Broadie and *Glasserman* in their paper[2] suggested a way as below.

They calculated b different $\bar{c}_1$ and $\bar{c}_2$'s as follows. For calculating $\bar{c}_1^i$ they got the sample mean of the option values at all of the replications except the i-th replication and for $\bar{c}_2^i$ they put it equal to the value of the option at the i-th replication. It means that they calculated b different $\hat{v}_{ik}^{j_1 j_2 \cdots j_i}$'s as below

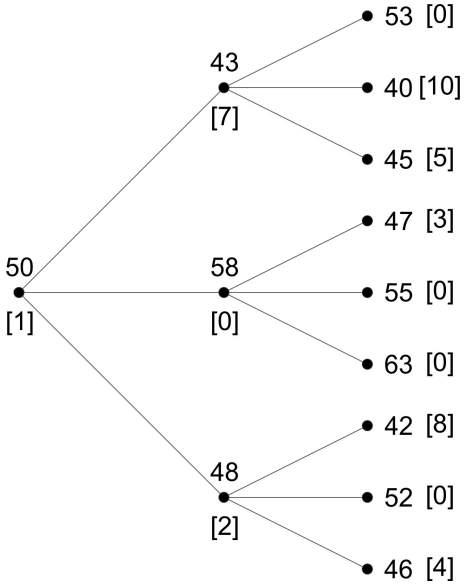


Figure 2.2: The process on the random tree to get a low estimator for a put option with stock and strike price equal to 50 (discount factor considered to be 1). In each state, labels are stock price and estimated option value (in bracket)

$(k = 1, 2, ..., b)$:

$$\hat{v}_{ik}^{j_1 j_2 \cdots j_i} = \begin{cases} h_i(S_i^{j_1 j_2 \cdots j_i}), & \text{if } h_i(S_i^{j_1 j_2 \cdots j_i}) \geq \frac{1}{b-1} \sum_{j=1, j \neq k}^{b} \hat{v}_{i+1}^{j_1 j_2 \cdots j_i j}; \\ \hat{v}_{i+1}^{j_1 j_2 \cdots j_i k}, & \text{otherwise.} \end{cases}$$

Then they set

$$\hat{v}_i^{j_1 j_2 \cdots j_i} = \frac{1}{b} \sum_{k=1}^{b} \hat{v}_{ik}^{j_1 j_2 \cdots j_i}.$$

They also provided a proof for the convergence in their paper[2].

Now, with these two estimators, if we form a 95% confidence interval for each of them, then we can use them to form a 90% confidence interval for the true value of the option.

2.2 Memory Requirement

As it was already stated, this method is not very efficient in practice. With increasing n, the computational time increases exponentially. We can do the im-

plementation in a way that we would not need an exponentially-growing memory requirement though.

Broadie and *Glasserman* described a detailed process how to not be obliged to save the whole tree. But also a more brief description is possible as follows. Suppose that we implement the method as a function to solve the problem recursively. When we execute the program, we actually run the function in which we only do the replication part and then call the same function again for each of the replications. When we know that no more replications are needed, we do not call the function again and we just return the payoff of exercising (it means that we are in a last step state). In this way, even if we save all the b states in each function, the function is called at most n times in a row (until it reaches the maturity time). So with considering the single starting state, we need to save at most an order of $nb + 1$ states.

2.3 Results

The results of this method are not considerable in comparison to other methods that we are going to discuss in following chapters. Even with a computational time around 30 seconds, the variance of the estimated values (for the same options that other methods estimated the value of them) is very high and usually the estimated values by this method are 10% blow or above the value computed by finite difference method.

This method has been implemented in MATLAB as like other methods and its implementation can be seen in appendix A.1.

Chapter 3

The Least-Squares Method

In this chapter Least-Squares Method or LSM suggested by *Longstaff* and *Schwartz*[4] will be discussed. This method is based on using regression to approximate the continuation values.

3.1 Approach

The approach is simple as follows. First divide the time interval to different time steps, $t_0 = 0, t_1, \cdots, t_N = T$. Simulate M (with M more antithetic trajectories) and compute the payoff at maturity time for each trajectory, $(K - s_N^i)^+$ (s_N^i is the stock value in maturity time for the i-th trajectory). Going backward from maturity time, at each time step, regress the discounted continuation value according to the stock price at current time. The regression is done only by the values of in the money trajectories. This regression give a conditional expectation which can be used to see whether to decide to exercise or continue. This process should be done until the first time step after $t_0 = 0$. Also note that at each step, the discounted continuation value is the discounted value of the payoff from exercising at the earliest time after the current time step that we decided to exercise the option.

This approach can be explained by a simple example (regression basis function can be chosen arbitrarily, though some of them are better to get more accurate results but for simplicity in this example regression is based on a constant, X

and X^2).

Suppose we have $S = 2$, $K = 2.5$, $M = 10$ trajectories, $T = 3$, $N = 3$ and $r = 0.06$. Also the simulated trajectories are as shown in Table 3.1.

Table 3.1: Stock price of 10 simulated trajectories

#	$t_0 = 0$	$t_1 = 1$	$t_2 = 2$	$t_3 = 3$
1	2.0000	1.0594	1.0633	1.5612
2	2.0000	1.7688	2.4863	2.6489
3	2.0000	1.9848	3.2129	3.4922
4	2.0000	2.5316	2.2920	1.7711
5	2.0000	2.5659	2.0577	3.3024
6	2.0000	3.5971	6.7394	5.6394
7	2.0000	1.9184	1.6326	1.6076
8	2.0000	2.9906	2.5891	3.9831
9	2.0000	1.6111	2.2270	2.6936
10	2.0000	1.8321	1.9369	2.7104

For the first step, the continuation value for trajectories number 1, 4 and 7 are 0.9388, 0.7289 and 0.8924 respectively and is equal to 0 for the other trajectories (the other trajectories expire worthless).

In time $t_2 = 2$, there are 7 in the money trajectories. For these trajectories, we compute the discounted value of continuation. These discounted continuation values, Y, and the stock price, X, are listed in Table 3.2[1].

With regressing Y according to a constant, X and X^2 we get the conditional expectation: $\mathrm{E[Y|X]} = 0.2693 \times \mathrm{X}^2 - 1.5512 \times \mathrm{X} + 2.2956$. With having these conditional expectation, now we decide to exercise any of in the money trajectories at time 2 or not. These values are shown in Table 3.3.

So by now the expected continuation value for time $t_1 = 1$ is as written in Table 3.4.

Note that continuation value for the second trajectory is 0 because although the trajectory is in the money at time 2, we decided not to exercise and unfortunately we did not get anything at time 3, the maturity time, because the

[1]The discount coefficient from a time step to its left time step is $e^{-0.06*1} = 0.9418$.

Table 3.2: Discounted continuation value and stock price for in the money trajectories at time $t_2 = 2$

#	X	Y
1	1.0633	0.9388×0.9418
2	2.4863	0.0000×0.9418
4	2.2920	0.7289×0.9418
5	2.0577	0.0000×0.9418
7	1.6326	0.8924×0.9418
9	2.2270	0.0000×0.9418
10	1.9369	0.0000×0.9418

Table 3.3: The payoff of exercising and expected payoff of continuing for in the money trajectories at time $t_2 = 2$

#	Exercising	Continuing	Decision
1	1.4367	0.9506	exercise
2	0.0137	0.1036	continue
4	0.2080	0.1550	exercise
5	0.4423	0.2440	exercise
7	0.8674	0.4809	exercise
9	0.2730	0.1767	exercise
10	0.5631	0.3014	exercise

trajectory was still not in the money. Actually we were expecting to get more than the payoff by exercising but we were unlucky!

If we continue the above process for time 1, we get X and Y as shown in Table 3.5 for 6 in the money trajectories.

Again with regressing Y according to a constant, X and X^2 we get the conditional expectation: $\mathrm{E[Y|X]} = 2.2038 \times \mathrm{X}^2 - 7.7261 \times \mathrm{X} + 7.0480$. For each of in the money trajectories at time $t_1 = 1$, the payoff from exercising and expected payoff of continuing is listed in Table 3.6.

So finally the continuation values for time $t_0 = 0$ is as written in Table 3.7. The final value is the discounted average of these values.

The value obtained by this process is 0.5121 and the stopping rule for these

Table 3.4: The expected payoff of continuing at time 1 for all trajectories

#	Continuation Value
1	1.4367
2	0.0000
3	0.0000
4	0.2080
5	0.4423
6	0.0000
7	0.8674
8	0.0000
9	0.2730
10	0.5631

Table 3.5: Discounted continuation value and stock price for in the money trajectories at time $t_1 = 1$

#	X	Y
1	1.0594	1.4367×0.9418
2	1.7688	0.0000×0.9418
3	1.9848	0.0000×0.9418
7	1.9184	0.8674×0.9418
9	1.6111	0.2730×0.9418
10	1.8321	0.5631×0.9418

Table 3.6: The payoff of exercising and expected payoff of continuing for in the money trajectories at time $t_1 = 1$

#	Exercising	Continuing	Decision
1	1.4406	1.3364	exercise
2	0.7312	0.2770	exercise
3	0.5152	0.3949	exercise
7	0.5816	0.3368	exercise
9	0.8889	0.3208	exercise
10	0.6679	0.2903	exercise

Table 3.7: The expected payoff of continuing at time 0 for all trajectories

#	Continuation Value
1	1.4406
2	0.7312
3	0.5152
4	0.1959
5	0.4166
6	0.0000
7	0.5816
8	0.0000
9	0.8889
10	0.6679

trajectories are shown in Table 3.8.

Table 3.8: Stopping rule

#	$t_1 = 1$	$t_2 = 2$	$t_3 = 3$
1	1	0	0
2	1	0	0
3	1	0	0
4	0	1	0
5	0	1	0
6	0	0	0
7	1	0	0
8	0	0	0
9	1	0	0
10	1	0	0

This stopping rule is obtained by checking when was the last time (the most close time to starting time $t_0 = 0$) that each trajectory has been exercised. At last step, we exercised trajectories 1, 2, 3, 7, 9 and 10 (see Table 3.6) and in the step before the last step, we exercised trajectories 1, 4, 5, 7, 9 and 10 (see Table 3.3). But we never exercised trajectories 6 and 8 (we even did not do the comparison to see whether to exercise them or not any time, because they were always out

of the money)[1]. The reason for that is easily understandable. With a perfect stopping rule, even a trajectory that is in the money most of the times may not be exercised ever until it does not go below the optimal stopping boundary. So this case can happen with an approximated stopping rule too!

3.2 Basis Functions for Regression

The basis functions that we used for regression in above example were a constant, X and X^2, but in experiments these basis functions are not good enough to approximate the conditional expectation. The paper[4], suggests different types of basis functions such as (weighted) Laguerre, Hermite, Legendre, Chebyshev, Gegenbauer and Jacobi polynomials.

For implementation, several different type of those functions were used. Numerical tests showed that with the same number of basis functions from each type of functions, one of the best of them to be used as basis functions are weighted Laguerre polynomials.

3.2.1 Weighted Laguerre Polynomials

For implementation of this method, weighted Laguerre polynomials are used. The n-th Laguerre function, denoted by L_n, is defined as below

$$L_n(x) = \frac{e^x}{n!}\frac{d^n}{dx^n}\left(e^{-x}x^n\right),$$

and for getting *weighted* Laguerre polynomials we can multiply them by for example $e^{-\frac{x}{2}}$.

In paper it states that with a constant and first three weighted Laguerre polynomials some sufficiently good results are obtained, but with using only the three first weighted Laguerre polynomials, the obtained results were not very accurate. This can be because of several reasons, like the probably different way of simulating trajectories or maybe the built-in **regress** function of **MATLAB**

[1] Also note that we could have never decided to exercise the second trajectory if it was out of the money at time $t_1 = 1$, although it was in the money most of the times.

that has been used to compute the regression. This function is for calculating multiple linear regressions where the best results are achieved by having different linearly independent observations. If instead of only three first polynomials we use first eight polynomials, we can obtain a good accuracy.

The basis function that had been used in implementation are as below.

$$1$$

$$e^{-\frac{x}{2}}$$

$$e^{-\frac{x}{2}}(-x+1)$$

$$e^{-\frac{x}{2}}\left(\frac{x^2-4x+2}{2}\right)$$

$$e^{-\frac{x}{2}}\left(\frac{-x^3+9x^2-18x+6}{6}\right)$$

$$e^{-\frac{x}{2}}\left(\frac{xx^4-16x^3+72x^2-96x+24}{24}\right)$$

$$e^{-\frac{x}{2}}\left(\frac{-x^5+25x^4-200x^3+600x^2-600x+120}{120}\right)$$

$$e^{-\frac{x}{2}}\left(\frac{x^6-36x^5+450x^4-2400x^3+5400x^2-4320x+720}{720}\right)$$

$$e^{-\frac{x}{2}}\left(\frac{-x^7+49x^6-882x^5+7350x^4-29400x^3+105840x^2-35280x+5040}{5040}\right)$$

3.3 Convergence

The authors did not provide any proof for the convergence of LSM, but they proved that the approximated value is biased low relative to true value when the number of trajectories goes to infinity (with a finite number of basis functions, number of time steps and coefficients of basis functions obtained by regression).

The fact of being biased low relative to true value has a simple explanation: The perfect stopping rule maximizes the option value obtained by infinite number of truly random trajectories, while any other approximated stopping rule (which is the case in LSM of course) ca not achieve that maximum.

3.4 Results

The implemented LSM which can be seen in appendix A.2, has been tested for 20 different American put options. The results are presented in Table 3.9[1].

Table 3.9: Comparing results of Finite Difference Method and Least-Squares Method. LSM is using 50000 simulated trajectories (plus 50000 antithetic trajectories) and 50 time steps for each year. Option strike price, K, is 40 and risk free interest rate is 0.06 in all cases.

S	σ	T	FDM Values	LSM Values	Difference
36	.20	1	4.478	4.478	0.000
36	.20	2	4.840	4.858	-0.018
36	.40	1	7.101	7.087	-0.013
36	.40	2	8.508	8.456	0.052
38	.20	1	3.250	3.263	-0.013
38	.20	2	3.745	3.765	-0.020
38	.40	1	6.148	6.157	-0.009
38	.40	2	7.670	7.636	0.033
40	.20	1	2.314	2.317	-0.003
40	.20	2	2.885	2.883	0.001
40	.40	1	5.312	5.283	0.029
40	.40	2	6.920	6.899	0.021
42	.20	1	1.617	1.605	0.011
42	.20	2	2.212	2.216	-0.004
42	.40	1	4.582	4.600	-0.018
42	.40	2	6.248	6.232	0.015
44	.20	1	1.110	1.118	-0.008
44	.20	2	1.690	1.703	-0.013
44	.40	1	3.948	3.972	-0.024
44	.40	2	5.647	5.607	0.039

The Finite Difference Method Value (FDM Value) column in the table is taken from the paper[4]. According to it, the FDM used to obtain those results were

[1]Note that the results are computed with more than 3 precision digits, but are written with only their first 3 precision digits. Also the difference is shown with only 3 precision digits, so the positive differences may look like that they are 0.001 unit less than the true difference.

from an implicit finite difference method with 40000 time steps for each year and 1000 price step for stock price.

Computational time varying from less than 1 second to less than 5 seconds. The average computation time was around 2 seconds[1].

[1]All of the methods in this thesis were tested on a computer with Intel Core 2 Duo, 2.53 GHz CPU.

Chapter 4

Rogers' Martingale Method

The method that we explain in this chapter is based on *L. C. G. Rogers* paper[6]. He suggests a method using martingales which gives a biased high estimator relative to the true value. The approach is pretty similar to what has been described in section 1.5 and is as follows.

4.1 Approach

The theoretical basis of the paper is the following theorem.

Theorem 4.1.1

$$V_0 = \inf_{M \in H_0^1} \mathrm{E}\left[\sup_{0 \le t \le T} (Z_t - M_t)\right]$$

where H_0^1 *is the space of martingales* M *for which* $\sup_{0 \le t \le T} \mid M_t \mid \in L^1$*, and such that* $M_0 = 0$*. Also* Z_t *is the discounted payoff of exercising the option at time* t*.*

The proof of Theorem 4.1.1 is beyond the scope of this thesis.

Let us explain the idea in this method in more simple words. Suppose that we have an arbitrary martingale, M_t^*, which vanishes at 0. Now if we simulate n trajectories, and for each of the trajectories, like S_t, we calculate $\sup_{0 \le t \le T}(Z_t - M_t^*)$ where $Z_t = (K - S_t)^+$, and then we get the average of all of them (so that we estimate the expectation), then the value would be an upper bound for the true value (as $n \to \infty$).

So it suggests that we first find a (group of) *suitable* martingale(s), then simulate a lot of trajectories and calculate the corresponding value, and then get the average and that would be an (biased high) approximation of the true value.

Rogers suggests a single martingale for pricing an American put option which also gives some good results. For each trajectory, the martingale is the discounted value of the corresponding European put option, started at the first time that S_t goes below the Strike price, K. It means that our martingale is as bellow.

$$M_t = \begin{cases} 0 & \text{for } t < \gamma \\ e^{-r(t-\gamma)}\text{BS}(S_t, K, r, \sigma, T-t) - \text{BS}(K, K, r, \sigma, T-\gamma) & \text{for } t \geq \gamma \end{cases}$$

where γ is the first time that option goes in the money, $S_t = S(t)$, and BS give the Black-Scholes value of the European put option.

With using only one martingale we can not be sure that the approximation is actually accurate, but the results show that it is reasonably accurate.

4.2 Optimizations

Clearly if M is a martingale which vanishes at 0, then also $\lambda \cdot M$ is a martingale which vanishes at 0.

So we can probably find a better λ than $\lambda = 1$ for martingale defined above.

We do as follows. We simulate a small number of trajectories (usually less than 1000) and will try to find a λ which minimizes $\text{E}\left[\sup_{0\leq t\leq T}(Z_t - \lambda M_t)\right]$ (of course we are not able to find the true expectation, but we are just approximating it by those few trajectories). Suppose that λ^* is minimizing that expectation. Then we put $M^* = \lambda^* \cdot M$, and simulate a large number of independent trajectories to previous trajectories and approximate $\text{E}\left[\sup_{0\leq t\leq T}(Z_t - M_t^*)\right]$ according to them. This value would still be biased high, but it should give a better approximation of the true value.

Of course, in practice, for a simple American put option, λ^* is usually very close to 1 as also stated in the paper but with using it, it *does* optimize the final approximated value.

The approach to do this optimization in the implementation is that first we

decrease λ by a small value a couple of times and then until it seems that decreasing it makes the above expectation smaller, then we do the converse process. In one of these direction we will eventually get enough close to λ^* (intuitively, decreasing/increasing smaller values result in better approximations of λ^*).

4.3 Results

The results with this method for the same 20 American put options as other chapters is shown in Table 4.1.

As it can be seen, most of the differences are negative, showing that the approximated value is actually an upper bound of the true value.

Computational time was varying from 4 to 30 seconds, averagely around 16 seconds.

The implementation of this method can be seen in appendix A.3.

Table 4.1: Comparing results of Finite Difference Method and Rogers' Method. It is using 5000 final simulated trajectories and 300 simulated trajectories for optimization and 50 time steps for each year. Option strike price, K, is 40 and risk free interest rate is 0.06 in all cases.

S	σ	T	FDM Values	Rogers' Method Values	Difference
36	.20	1	4.478	4.525	-0.047
36	.20	2	4.840	4.983	-0.143
36	.40	1	7.101	7.095	-0.006
36	.40	2	8.508	8.592	-0.084
38	.20	1	3.250	3.295	-0.045
38	.20	2	3.745	3.885	-0.140
38	.40	1	6.148	6.140	0.007
38	.40	2	7.670	7.729	-0.059
40	.20	1	2.314	2.364	-0.050
40	.20	2	2.885	2.986	-0.101
40	.40	1	5.312	5.314	-0.002
40	.40	2	6.920	6.989	-0.069
42	.20	1	1.617	1.642	-0.025
42	.20	2	2.212	2.287	-0.075
42	.40	1	4.582	4.575	0.006
42	.40	2	6.248	6.281	-0.033
44	.20	1	1.110	1.139	-0.029
44	.20	2	1.690	1.765	-0.075
44	.40	1	3.948	3.947	0.000
44	.40	2	5.647	5.670	-0.023

Chapter 5

Interval-Based Expectation Method

5.1 Approach

The first new method to be discussed is an *Interval-Based Expectation* method. This method is one word and is trying to solve the problem determined by (1.5) and (1.6) directly. A few optimization steps used to make some better results which will be discussed later.

The whole approach is as follows. It uses time discretization and splits the interval from time zero to maturity time to N different times, $t_0 = 0, t_1, \cdots, t_N = T$. With simulating M trajectories (plus M antithetic trajectories) it tries to approximate the value of option corresponding to each trajectory in each of time steps. First it calculates the option value at maturity time which for each trajectory is simply equal to payoff function (according to (1.5)), $(K-s_N)^+ = \max\{K-s_N, 0\}$. By going backward from the maturity time, at each step it approximates the expected payoff of each trajectory at that step. For doing the comparison in (1.6) two values should be computed: First the payoff at that time step and second the corresponding expectation. The payoffs of each trajectory are determined obviously. To determine the expectation, for each trajectory, algorithm gets an average of values of all trajectories that pass through a small interval containing that trajectory (see Figure 5.1) in previous step (one time step closer to maturity

time). For choosing those trajectories that are being used for approximation, algorithm starts from the main trajectory and search for the next trajectory that passes through the interval and has the most close trajectory to main trajectory in either direction. In the sense that if for example we had the value of 34.5 for the stock at a time step in the main trajectory, algorithm gets the first trajectory that has the value (equal to or) less than 34.5; after choosing that trajectories, it continues to choose in this manner until a limited number of trajectories. It then gets the trajectories with stock value (equal to or) greater than 34.5.

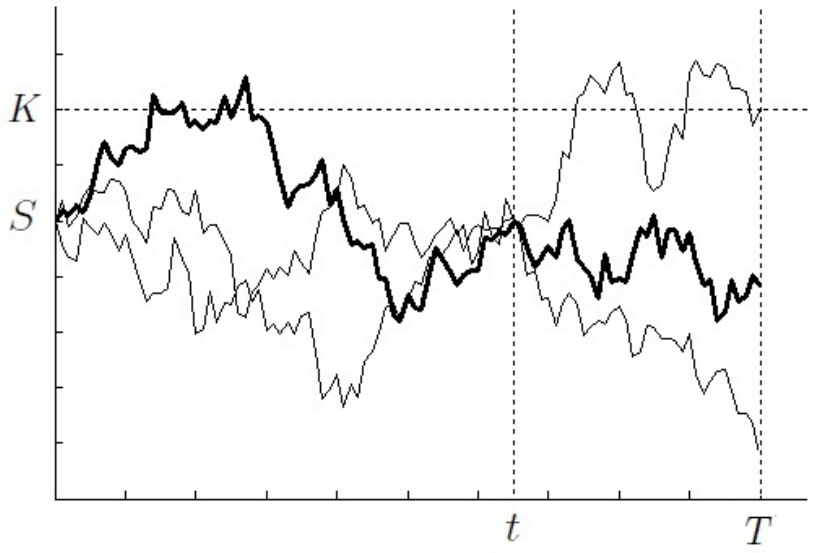


Figure 5.1: The main trajectory (bold) and other trajectories which pass through the interval at time t

If we, by a chance, have sufficiently many trajectories passing through that interval, then we hopefully can have a good approximation close enough to real value.

After approximating that value, algorithm makes the comparison in (1.6), seeing whether the payoff of immediate exercise is higher or the expected payoff of continuing (which already had been approximated by nearby trajectories at that point), deciding whether to exercise or continue. If it decided to exercise, then put the value of payoff as corresponding option value at that time step and if it decided to continue it puts the discounted approximated value. So the

algorithm assumes that when going backward, it has well approximated the value of option corresponded to each trajectory, in previous step.

5.2 Convergence

We can prove the convergence of the algorithm, when the number of trajectories goes to infinity, using induction and law of large numbers. A sketch of proof is as follows. Induction base is the maturity time where clearly the computed value for each trajectory is exactly equal to value of the corresponding option at that time. While going backward we can assume that we have computed the exact values for each trajectory in previous time step (by induction). For current time step, because of the infinite number of trajectories and the way algorithm chooses them in the interval, we get infinite number of trajectories that for each of them we already have the exact values of continuation and where also their stock value is equal to the main trajectory stock value at that time. Because of the assumption that we have the exact value of continuation of those trajectories, when we get the average, we still have the exact value. And so at the time of making the comparison we do not have any error.

This proves the convergence of the algorithm for a *Bermudan* put option. So if maximum length of time steps goes to zero, the option will be in fact an American put option (which is exercisable in every moment until maturity time).

5.3 Optimizations

5.3.1 Bias

It is not easy to see whether the result from this algorithm is biased high or low relative to the true value. The bias happens because of various reasons. For example it can happen because of probably not being able to simulate trajectories based on a true randomness. It also can happen because of the finite number of exercise date (which changes the option to Bermudan put option). Apart from those two reasons, a bias can happen when we approximate the expectation with finite number of trajectories.

If we assume that we were looking for the price of the Bermudan put option already and also that we can simulate the trajectories with a sufficiently (high) accuracy, the last reason can be the main reason producing the bias. When testing the algorithm, it has been seen that the number of decisions to exercise (in 1.6) is less than the number of decisions to continue (around one tenth in many experiments) which is of course understandable (because the expected average of trajectories is a straight line with the slope equal to the risk free interest rate and on the other hand, the optimal stopping boundary is well below the straight horizontal line at K and so trajectories are usually above the optimal stopping boundary in comparison being below of it which we would exercise the option). So this bias (if be a large bias) can effect the result pretty much. This bias will be high if the approximated expectation is usually bigger than the true expectation and vice versa. In tests it has been seen the first is happening: The final value is usually biased high relative to the true value. We can try to offset this bias in many ways. One way is to increase the number of trajectories or the number of time steps. But both of these ways, increase the computation time a lot.

5.3.2 Computational Time

Consider that we are approximating the expectation for a trajectory at a time step. The number of trajectories close to the main trajectory (and in the interval) can be unlimitedly large (in comparison to the number of all simulated trajectories) which leads to a big computational time. But in implementation a limit has been defined, so that we do not get greater number of trajectories than a defined limit (for example 250). Also to offset the mentioned bias we can set the limit of number of trajectories with stock value below the main trajectory's stock value to be a bit less than the limit of number of trajectories with stock value above it (for example 240 rather than 250). You can see that this way has been used in the implementation of the algorithm in appendix A.4. And also with this limit (250) the computational time in not much, usually less than 20 seconds.

5.3.3 Simulating New Trajectories

What happens if the number of trajectories in a small interval around the main trajectory at some time steps is too small? It can be said that the approximation can lose its accuracy to some extent. For solving this issue one idea can be to add some new independent trajectories, to increase the accuracy. This idea is discussed in below.

5.3.3.1 Approach

Look at Figure 5.1. If we have many simulated trajectories, the chance of having too few trajectories in that interval is low, but still it can happen in the extreme trajectories (trajectories which are fluctuating very much). One may say that the effect of value of these (probably) few trajectories in the final result is not that much; but still it is not easily possible to estimate it. The good part is that in put options, the final value of a trajectory is bounded. The estimated value of each trajectory is at most K, which may happen if the stock price be close to 0 at some points. The effect of these *bad* simulated trajectories could be much worse if the estimated value could get arbitrarily large (like call options). But even in put options, this effect can be big enough to make the final result not-accurate and it would be better to control it in some ways.

The idea that was thought can be used to (probably!) increase the accuracy is as follows. We set a limit, and whenever the number of trajectories in the interval be less than this limit, we simulate some new independent trajectories from the stock price at main trajectory until the maturity time and we compute the value of option with these new parameters (S equal to stock price at that point, K is the same, and T is the length of time interval between current time step to the maturity time). For these new trajectories, we use the same method as discussed in this chapter, but without any limit that we set in this new revised version. See Figure 5.2.

This revision can arise several new questions. One is that, is it not better to put a limit for the new trajectories too? How many exercise points should be considered for the new simulated trajectories to achieve the best possible approximation? How many new trajectories should be simulated? How big or

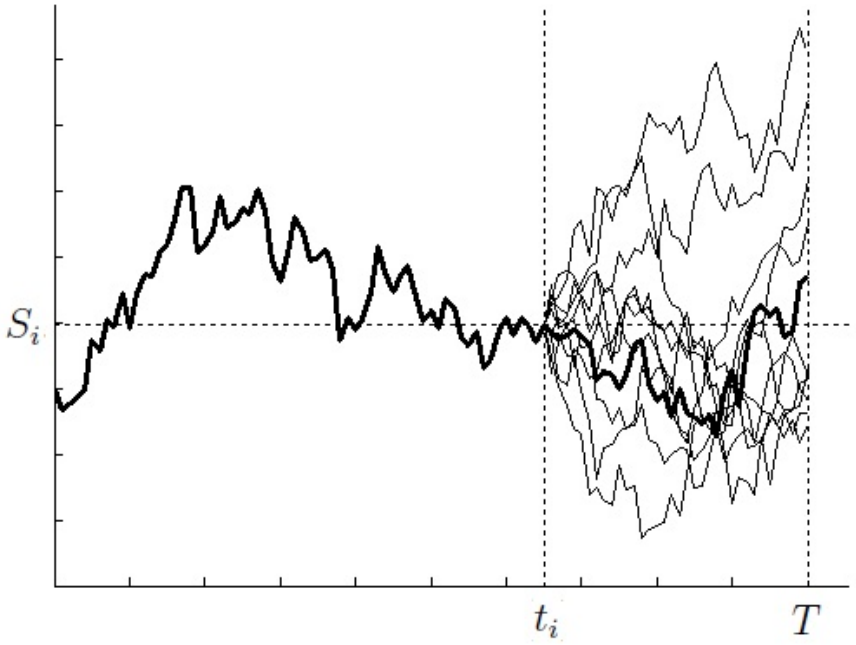


Figure 5.2: The main trajectory (bold) and new simulated trajectories which start from the current time until maturity with starting stock price equal to main trajectory's stock price

small can be the limit? Or is it good to have different limits for different number of trajectories? And many more questions. Rather than theoretically, we can try to guess the answers to these questions practically. For example, it has been seen that in practice, reducing the number of simulated trajectories does not decrease the computation time as expected, because the number of times that we need to simulate new trajectories for approximating the expectation, is increased in some cases. Or that increasing the number of exercise points in new simulated trajectories does not necessarily increase the accuracy. Or increasing the number of new simulated trajectories are not good enough in comparison to increase the number of starting simulated trajectories. And several other observations that has been achieved during testing this revised method.

5.3.3.2 Convergence

In comparison to previous method, the new simulated trajectories are added. If the number of starting trajectories goes to infinity, and if the limit of trajectories in interval be larger than twice the limit introduced in this chapter (in which we simulate new trajectories if the number of trajectories in interval be less than it), then this revised part does not effect the method and convergency would be the same as the original method. Also even if we do simulate new trajectories, but still consider it to be infinitely many and with infinite number of exercise points, it will actually again converge to the true value.

On the other hand, intuitively the convergence is easily understood when the number of trajectories and time steps goes to infinity because of the law of large numbers and the fact that algorithm base the approximations and results according to exact values in each trajectory (not for example make a regression like LSM or any other approximation functions).

5.3.3.3 Outcome

Unfortunately the results for this revised method are not good as expected. In analyzing the result, several reasons had been found for this issue. The main reason seems to be that in several cases the approximated continuation value from new simulated trajectories are not accurate enough in compare to even the approximation from those few (starting) trajectories. And in these many cases, new simulated trajectories just decrease the accuracy. Of course this can give a more accurate approximation if we increase the number of new simulated trajectories, but it needs a lot of more computational time rather than just forget these new simulated trajectories and increase the starting trajectories to increase the accuracy!

5.4 Results

The computational time is dependent on the number of trajectories, number of time steps, the length of interval, the limit of the number of trajectories in the interval (and of course different ways of implementation). It depends on how our

time constraint is. For example For a $T = 2, r = 0.06, \sigma = 0.4$ good results can be obtained with 20000 trajectories (plus 20000 antithetic trajectories) and 100 time steps with around 30 seconds for computation time.

For reducing the computational time, we can do several things: We can reduce the items mentioned above (which reduces the algorithm's accuracy) or maybe just skip doing the expectation approximations for options that are out of the

Table 5.1: Comparing results of Finite Difference Method and two versions of Interval-Based Expectation method; first (#1) is with 20000 simulated trajectories (plus 20000 antithetic trajectories), 100 time steps and skipping approximations for out of the money options, second (#2) is with 15000 simulated trajectories (plus 15000 antithetic trajectories), 100 time steps and approximating the expectation for all trajectories. Option strike price, K, is 40 and risk free interest rate is 0.06 in all cases.

S	σ	T	FDM Value	Interval-Based Expectation Results #1	Interval-Based Expectation Results #2	FDM and #1 Results Differences	FDM and #2 Results Differences
36	.20	1	4.478	4.519	4.511	-0.041	-0.033
36	.20	2	4.840	4.857	4.848	-0.017	-0.008
36	.40	1	7.101	7.074	7.109	0.026	-0.008
36	.40	2	8.508	8.490	8.538	0.018	-0.030
38	.20	1	3.250	3.239	3.225	0.011	0.024
38	.20	2	3.745	3.737	3.719	0.007	0.025
38	.40	1	6.148	6.148	6.152	0.000	-0.004
38	.40	2	7.670	7.674	7.695	-0.004	-0.025
40	.20	1	2.314	2.304	2.275	0.009	0.038
40	.20	2	2.885	2.876	2.858	0.008	0.026
40	.40	1	5.312	5.315	5.334	-0.003	-0.022
40	.40	2	6.920	6.917	6.947	0.003	-0.027
42	.20	1	1.617	1.605	1.597	0.011	0.020
42	.20	2	2.212	2.205	2.188	0.006	0.023
42	.40	1	4.582	4.572	4.587	0.009	-0.005
42	.40	2	6.248	6.241	6.262	0.007	-0.014
44	.20	1	1.110	1.100	1.083	0.009	0.026
44	.20	2	1.690	1.692	1.661	-0.002	0.028
44	.40	1	3.948	3.949	3.959	-0.001	-0.011
44	.40	2	5.647	5.643	5.681	0.003	-0.034

money and simply put the discounted value of that option at next time step (which seems to also reduce the accuracy). It will reduce the computational time from around 30 seconds to around 15 seconds (this computational time is around 10 seconds when the starting stock price is bigger than the strike price). Without skipping computing the expectation approximations and by just reducing the number of trajectories to 15000, we get almost 20 seconds for the computational time. With skipping the approximations or reducing the trajectories, we in fact are decreasing the accuracy. But the accuracy in both cases is still good as shown in Table 5.1.

For these tests, average computational time for results #1 was around 14 seconds. This average for results #2 were around 19 seconds. By knowing the computational time and looking at the error columns (last two columns) in Table 5.1, we conclude that if we have a time constraint, it is better to have more simulated trajectories and just skip the computations for out of the money options rather than keep computing them and reduce the number of simulated trajectories. Also in that case, if the beginning stock price be larger than the strike price, we can have a very faster algorithm and also in general we will expect a higher accuracy in compare to when we reduce the number of simulated trajectories.

Chapter 6

Optimal Stopping Boundary Estimation Method

In this chapter we try to estimate *optimal stopping boundary*. As stated before, no solution had been found to the problem of optimal stopping boundary. Several methods in pricing options try to just estimate it and then with the help of this estimation they price the options. Although as already been said that the value of option is not much sensitive to the exact knowledge of optimal stopping boundary, for a very accurate price, we need an accurate stopping boundary for sure.

6.1 Approach

Suppose we have a new option, which is obtained from the equivalent European put option plus the possibility of exercising at time 0. We call it a semi-European put option. With Black-Scholes formula we can value an European put option, but what about the new semi-European option?

By checking European put options values in Table 6.1 we see that from a point (after increasing the maturity time), the option value goes below the absolute value of difference between the stock and the strike price.

In Table 6.1, the first time that the option values goes below the absolute value of difference between stock and strike price (equal to 4, in this case), is with maturity time of 25. So what can we say about the value of semi-European

Table 6.1: European put option values with different maturity time. The stock price, strike price, risk free interest rate and volatility are 36, 40, 0.06 and 0.4 respectively.

T	Option Value
5	8.3961
10	7.5608
15	6.2475
20	4.9883
25	3.9107

put option with the same parameters? At the time of 0, this value is not less than the regular European option for sure. Because more than exercising at maturity, in semi-European put option, we have the option to exercise at time 0. So for a semi-European put option, we surely choose not to exercise in any one of first four options in Table 6.1. That is because the value of European put option is actually the expected payoff of continuing at time 0! So because the European put option value is decreasing with increasing maturity time, we can obtain that the value of the semi-European put option with the same parameters is as shown in Table 6.2.

Table 6.2: Semi-European put option values with different maturity times. The stock price, strike price, risk free interest rate and volatility are 36, 40, 0.06 and 0.4 respectively.

T	Option Value
5	8.3961
10	7.5608
15	6.2475
20	4.9883
≥ 25	4.0000

So actually we can say that the value of semi-European put option is equal to $\max\{(K-S)^+$, value of equivalent European put option$\}$.

Now coming back to American put option, it is obvious that their values are

greater than or equal to value of their equivalent semi-European put options. So for having a rough estimation of optimal stopping boundary of (Bermudan) American put option, in each time step, we can search for the highest price for which the value of semi-European put option starting at that position is equal to $(K - S)$.

Look at the Figure 6.1. It is understandable that to find this rough estimation, the stock price, S, is not important. For any arbitrary stock price, but the same strike price, risk free interest rate, volatility and maturity time, this rough estimation is the same[1].

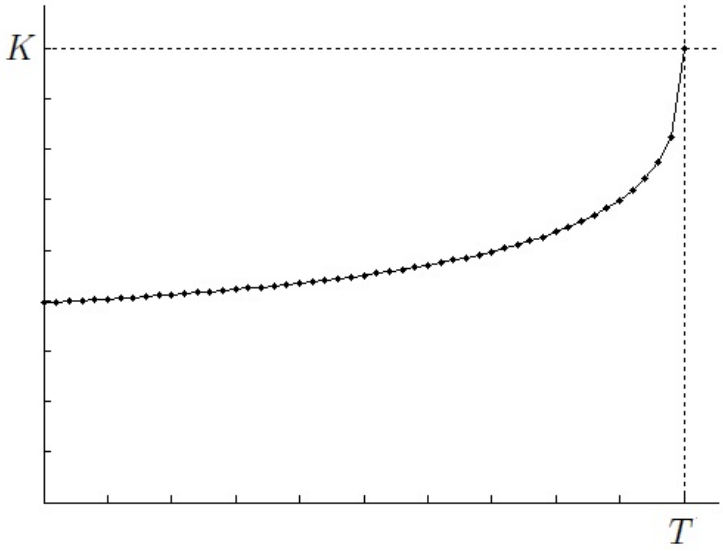


Figure 6.1: A rough approximation of optimal stopping boundary, with 50 time steps.

Now that we have this rough estimation, we may be able to optimize it in some ways. A suggestion to do it, which has been also implemented, is as follows.

[1]According to this fact, we can use this rough estimation (and/or its optimized version) for several groups of new independent trajectories. These boundaries are not dependent on the stock price of American put options and neither on any new simulated trajectories to exercise according to them and to obtain the price the option, and therefore we can for example compute these boundaries with high accuracy once, save them and use them anytime later for any American put options with the same strike price, maturity time, risk free interest rate and volatility.

We try to alter this boundary so that it give us a good estimation of the real optimal stopping boundary depended on the finite number of time steps. The value of this boundary at maturity time is K obviously. This value at time step right before the maturity is, can be get better as what already we have estimated it. Because the Bermudan put option which starts at this time step, is actually a semi-European put option (we can either exercise at that point or decide to continue and wait until the maturity time without any more options to exercise before the maturity time). Now by going backward from the time step before that time step (two time steps before maturity time), for each remaining time steps, we do as this:

- We start from the stock price equal to minimum of the value obtained at previous step and the value of the rough estimation at current time step.
  - We simulate some new independent trajectories started at this time step and at this stock price. According to previous time steps approximations of optimal stopping boundary, we approximate the value of an American put option (according to those new simulated trajectories) that starts at this time step with this stock price. The process of approximating the value of option is that while we simulate the new trajectories, we exercise them according to the optimal stopping boundary that has been estimated until now (we already has the new estimations of optimal stopping boundary in time steps towards maturity time).
  - We consider this new estimated value of option as the discounted continuation value. Now we see whether the payoff of immediate exercise is higher or the expected payoff of continuing. If the expected payoff from continuing is higher, then we decrease the stock price (which leads to increasing the payoff of immediate exercise) and do the whole previous process again (We do it until we arrive at a stock price that if we decrease the stock price further, the payoff from immediate exercise be higher that continuation time. This means that we approximately got the stock price on the real optimal stopping boundary).

If we do as the algorithm above, we might end up getting an optimized version of the rough estimation of optimal stopping time, as shown in Figure 6.2[1].

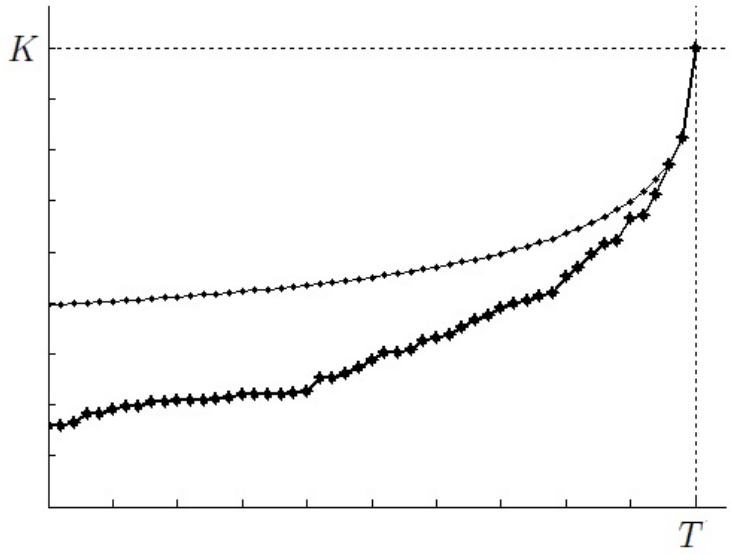


Figure 6.2: The optimized version (bold) of previous rough approximation of optimal stopping boundary.

The rate of decreasing the stock price and then simulating new trajectories from the new stock price, and also the number of new simulated trajectories are some of the main parameters to achieve the desired accuracy of the new stopping boundary.

We will discuss more about this in Optimization section.

6.2 Convergence

The proof of this method's convergence is simple.

Suppose that the rate of decreasing stock price goes to zero and the number of new simulated trajectories goes to infinity. Then in this case the stopping boundary converges to the true optimal stopping boundary for the corresponding Bermudan put option obviously. Now if we increase the time steps unlimitedly,

[1]Note that the figure is not starting exactly from the point 0 of vertical axis.

this optimal stopping boundary will be the optimal stopping boundary an American put option.

On the other hand when we are going to use this optimal stopping boundary to value the option, if we simulate infinitely many trajectories and exercise them according to this optimal stopping boundary and then get the average, we will in fact get the exact value of the option (law of large numbers).

So the method converges to the true value when the number of simulated trajectories in both cases goes to infinity and also we decrease the stock price at each step with $\varepsilon \to 0$.

6.3 Optimization

As also wrote in Convergence section, obviously simulating more trajectories and also decreasing the rate of reducing stock price are helping to optimize the stopping boundary. More than that, by increasing the number of time steps we can optimize this boundary. But all of them will increase the computational time a lot. So can we do something else if we want to keep the computational time constant and increase the accuracy?

By any arbitrary parameters, because of the nature of Monte Carlo methods we always get different results by running the method again. But one way is using antithetic variates method to help reducing the variance. It will help a lot optimizing the stopping boundary after roughly approximating it.

The most important part of where we can try to optimize the algorithm while having the same computational time is when we are going to decrease the stock price. Several ways had been checked to see whether they increase the accuracy without necessarily increasing computational time and one of the best ways is as below.

The way has been used, uses the idea that we do not have to always decrease the stock price with a constant value. As can be seen in implementation in Appendix A.5, we first reduced the price by for example 0.1, then we decrease the price until we reach to the point that payoff is more than the approximated continuation value. Then we increase it by 0.1, and this time start decreasing it by 0.01 instead of 0.1. When we realize that we probably went below the optimal

stopping boundary, we increase the price by 0.01 and do the process again with 0.001 this time. We can do it as many times as we want and with more steps, we get more accuracy for sure (every time around a digit of precision, but not exactly as that good of course, because the simulated trajectories are not perfect neither and their number is limited too).

Also while we change the amount of decreasing the stock price we may try to change the number of simulated trajectories too. But unfortunately this way seems not to be useful practically.

6.4 Results

This method has been used to price the same options as previous tests and the option prices obtained by this method are shown in Table 6.3.

Every parameter discussed in previous sections, obviously effect the computational speed. But in this method it has been tried to always have a computational time almost less than 5 seconds. The average computational time is around 3 seconds.

Again, because this method tries to estimate optimal stopping boundary and then uses this boundary to exercise some simulated trajectories and get the value of the option, it is intuitively biased low relative to the true value. But practically it does not happen, as by looking at the results, most of the differences are negative. This suggests that by increasing the simulated trajectories we can be hopeful to converge to the true value with a good rate and that the difference appeared on those results are coming mostly from the limited number of trajectories and/or the not high accuracy of simulated trajectory (the randomness in computer programs is not perfect anyway).

One of the good things about this method is that it does not need to store all of the trajectories, so the number of trajectories can be increased unlimitedly, as long as we do not mind about the computational time. So memory limitations does not bound this method to achieve some certain accuracy.

Table 6.3: Comparing results of Finite Difference Method and OSB Estimation Method. It is using 200000 final simulated trajectories (plus 200000 antithetic trajectories), 1000 simulated trajectories (plus 1000 antithetic trajectories) for each round of reducing stock price and 50 time steps for each year. Option strike price, K, is 40 and risk free interest rate is 0.06 in all cases.

S	σ	T	FDM Values	OSB Estimation Values	Difference
36	.20	1	4.478	4.476	0.001
36	.20	2	4.840	4.838	0.001
36	.40	1	7.101	7.111	-0.010
36	.40	2	8.508	8.522	-0.014
38	.20	1	3.250	3.253	-0.003
38	.20	2	3.745	3.752	-0.007
38	.40	1	6.148	6.150	-0.002
38	.40	2	7.670	7.683	-0.012
40	.20	1	2.314	2.313	0.001
40	.20	2	2.885	2.889	-0.004
40	.40	1	5.312	5.323	-0.011
40	.40	2	6.920	6.930	-0.010
42	.20	1	1.617	1.616	0.010
42	.20	2	2.212	2.211	0.000
42	.40	1	4.582	4.597	-0.015
42	.40	2	6.248	6.258	-0.010
44	.20	1	1.110	1.114	-0.004
44	.20	2	1.690	1.702	-0.012
44	.40	1	3.948	3.969	-0.021
44	.40	2	5.647	5.659	-0.012

Chapter 7

Conclusions

Due to exponentially growing computational time of random tree method, it is practically impossible to use it even for a simple American put option.

Let us compare the other four methods: LSM, RMM[1], IBEM[2] and OSBEM[3].

Amongst them, IBEM and worst than that, RMM are slow. IBEM has an advantage to RMM and that is we are sure about its convergence. But for RMM, we can not be sure about the accuracy and even the convergence. More than that, IBEM is giving very more accurate results with almost the same computational time.

Both of the other two methods are pretty fast. Although the convergence of LSM has not been proved completely by the authors, but it is reliable and seems to converge. Also the results of it are enough accurate. Though the results of the implemented version of LSM in this thesis are not as accurate as those in their paper[4], they are still enough accurate to rely on. OSBEM, on the other hand, is converging to the true value intuitively and also is almost as fast as LSM. The results of it is more accurate than LSM too.

In general it can be said that if we are looking for a more faster method, we can choose either LSM or OSBEM and if we are looking for a more accurate method, we can choose between IBEM and OSBEM.

The best method amongst them for a simple American put option seems to be OSBEM. It gives the most accurate results (the average absolute error is equal

[1] Rogers' Martingale Method
[2] Interval-Based Expectation Method
[3] Optimal Stopping Boundary Estimation Method

to almost 2% of the stock price) for 20 American put options that we have been tested and also with a very small computational time.

For a quick comparison between LSM, RMM, IBEM and OSBEM take a look at the Table 7.1.

Table 7.1: Comparison of computational time and accuracy between different methods.

	LSM	RMM	IBEM	OSBEM
Average Absolute Error	0.017	0.051	0.009	0.008
Average Computational Time	2	16	14	3

Appendices

Appendix A

Matlab Implementations

A.1 A Random Tree Method

```
1 function value = Random_Tree_Method(m, b, S, K, T, sigma,
      r, lh)
2
3 value = max(0, K - S);
4
5 if m == 0
6     return
7 end
8
9 dt = T / m;
10 v = zeros(b, 1);
11 dc = exp(-r * dt);
12
13 for i = 1: b
14     dw = randn * sqrt(dt);
15     s = S + r * S * dt + sigma * S * dw;
16     v(i) = Random_Tree_Method(m - 1, b, s, K, T - dt,
           sigma, r, lh);
17 end
18
19 if lh == 'h'
20     c = dc * mean(v);
21     value = max(value, c);
22 elseif lh == 'l'
23     vk = zeros(b, 1);
24     for i = 1: b
25         c1 = (sum(v) - v(i)) / (b - 1);
26         c2 = v(i);
27         if value > c1
28             vk(i) = value;
29         else
```

```
30                  vk(i) = c2;
31              end
32          end
33          value = mean(vk);
34  end
```

A.2 Least-Squares Method

```
1 M = 50000; % Number of trajectories
2 N = 50; % Discretization points per unit of maturity time
3 T = 1; % Maturity time
4 S = 40; % Stock price
5 K = 40; % Strike price
6 sigma = .4; % Volatility
7 r = .06; % Riskfree interest rate
8
9 dt = T / N;
10 s = zeros(2 * M, N + 1);
11 s(:, 1) = S * ones(2 * M, 1);
12 xy = zeros(2 * M, 2);
13
14 for j = 1: M
15     for i = 1: N
16         dw = randn * sqrt(dt);
17         s(2 * j - 1, i + 1) = s(2 * j - 1, i) + r * s(2 *
               j - 1, i) * dt + sigma * s(2 * j - 1, i) * dw;
18         s(2 * j, i + 1) = s(2 * j, i) + r * s(2 * j, i) *
               dt - sigma * s(2 * j, i) * dw;
19     end
20     s(2 * j - 1, N + 1) = max(0, K - s(2 * j - 1, N + 1));
21     s(2 * j, N + 1) = max(0, K - s(2 * j, N + 1));
22 end
23
24 tic;
25
26 cont = s(:, N + 1);
27 discount = exp(-r * dt);
28
29 for i = N: -1: 2
```

```
30      cont = discount * cont;
31      p = 0;
32      for j = 1: 2 * M
33          if K - s(j, i) > 0
34              p = p + 1;
35              xy(p, 1) = s(j, i);
36              xy(p, 2) = cont(j);
37          end
38      end
39
40      x = xy(1: p, 1);
41      x2 = xy(1: p, 1) .^ 2;
42      x3 = xy(1: p, 1) .^ 3;
43      x4 = xy(1: p, 1) .^ 4;
44      x5 = xy(1: p, 1) .^ 5;
45      x6 = xy(1: p, 1) .^ 6;
46      x7 = xy(1: p, 1) .^ 7;
47      x8 = xy(1: p, 1) .^ 8;
48
49      basreg = [ones(p, 1) ...
50          exp(-x / 2) ...
51          exp(-x / 2) .* (1 - x) ...
52          exp(-x / 2) .* (x2 - 4 * x + 2) / 2 ...
53          exp(-x / 2) .* (-x3 + 9 * x2 - 18 * x + 6) / 6 ...
54          exp(-x / 2) .* (x4 - 16 * x3 + 72 * x2 - 96 * x +
                24) / 24 ...
55          exp(-x / 2) .* (-x5 + 25 * x4 - 200 * x3 + 600 *
                x2 - 600 * x + 120) / 120 ...
56          exp(-x / 2) .* (x6 - 36 * x5 + 450 * x4 - 2400 *
                x3 + 5400 * x2 - 4320 * x + 720) / 720 ...
57          exp(-x / 2) .* (-x7 + 49 * x6 - 882 * x5 + 7350 *
                x4 - 29400 * x3 + 105840 * x2 - 35280 * x +
                5040) / 5040];
```

```
58
59      reg = regress(xy(1: p, 2), basreg);
60
61      expec = basreg * reg;
62      p = 0;
63      for j = 1: 2 * M
64          if K - s(j, i) > 0
65              p = p + 1;
66              if K - s(j, i) > expec(p)
67                  cont(j) = K - s(j, i);
68              end
69          end
70      end
71 end
72
73 cont = discount * cont;
74
75 value = mean(cont);
76 disp(value);
77
78 toc;
```

A.3 Rogers' Martingale Method

```
M = 5000; % Number of trajectories
m = 300; % Number of trajectories for optimization
T = 2; % Maturity time
N = 50 * T; % Number of time steps
S = 44; % Stock price
K = 40; % Strike price
sigma = .4; % Volatility
r = .06; % Riskfree interest rate

dt = T / N;
s = zeros(M, N + 1);
s(:, 1) = S * ones(M, 1);

dci = exp(-r * dt) .^ [0: N];

tic

ind = (N + 1) * ones(m, 1);
for j = 1: m
    for i = 1: N
        dw = randn * sqrt(dt);
        s(j, i + 1) = s(j, i) + r * s(j, i) * dt - sigma *
            s(j, i) * dw;
        if (ind(j) == N + 1) && (s(j, i + 1) < K)
            ind(j) = i + 1;
        end
    end
end

v1 = 4 * K;
v2 = 4 * K - 1;
```

```
31 lambda = 1;
32 while v2 < v1
33     v1 = v2;
34     v2 = 0;
35     lambda = lambda - .05;
36     for j = 1: m
37         if ind(j) == N + 1
38             v2 = v2 + max(s(j, N + 1) - K, 0);
39         else
40             ov = 0;
41             bs = black_scholes_put(s(j, ind(j)), K, r,
                   sigma, T - (ind(j) - 1) * dt);
42             for i = ind(j): N + 1
43                 martingale = dci(i - ind(j) + 1) *
                       black_scholes_put(s(j, i), K, r, sigma,
                        T - (i - 1) * dt) - bs;
44                 ov = max(ov, dci(i) * max(K - s(j, i), 0)
                       - lambda * martingale);
45             end
46             v2 = v2 + ov;
47         end
48     end
49     v2 = v2 / m;
50 end
51 lambda = lambda + .05;
52 v1 = v2 + v1;
53 v2 = v1 - v2;
54 v1 = v1 - v2;
55 while v2 < v1
56     v1 = v2;
57     v2 = 0;
58     lambda = lambda + .05;
59     for j = 1: m
```

```
60          if ind(j) == N + 1
61              v2 = v2 + max(s(j, N + 1) - K, 0);
62          else
63              ov = 0;
64              bs = black_scholes_put(s(j, ind(j)), K, r,
                    sigma, T - (ind(j) - 1) * dt);
65              for i = ind(j): N + 1
66                  martingale = dci(i - ind(j) + 1) *
                        black_scholes_put(s(j, i), K, r, sigma,
                        T - (i - 1) * dt) - bs;
67                  ov = max(ov, dci(i) * max(K - s(j, i), 0)
                        - lambda * martingale);
68              end
69              v2 = v2 + ov;
70          end
71      end
72      v2 = v2 / m;
73
74  end
75  lambda = lambda - .05;
76
77  ind = (N + 1) * ones(M, 1);
78  for j = 1: M
79      for i = 1: N
80          dw = randn * sqrt(dt);
81          s(j, i + 1) = s(j, i) + r * s(j, i) * dt - sigma *
                s(j, i) * dw;
82          if (ind(j) == N + 1) && (s(j, i + 1) < K)
83              ind(j) = i + 1;
84          end
85      end
86  end
87
```

```
88 value = 0;
89
90 for j = 1: M
91     if ind(j) == N + 1
92         value = value + dci(N + 1) * max(K - s(j, N + 1),
               0);
93     else
94         ov = 0;
95         bs = black_scholes_put(s(j, ind(j)), K, r, sigma,
               T - (ind(j) - 1) * dt);
96         for i = ind(j): N + 1
97             martingale = dci(i - ind(j) + 1) *
                   black_scholes_put(s(j, i), K, r, sigma, T -
                    (i - 1) * dt) - bs;
98             ov = max(ov, dci(i) * max(K - s(j, i), 0) -
                   lambda * martingale);
99         end
100        value = value + ov;
101    end
102 end
103
104 value = value / M;
105
106 disp(value);
107
108 toc;
```

A.4 Interval-Based Expectation Method

```
M = 20000; % Number of trajectories
N = 100; % Number of time steps
T = 2; % Maturity time
S = 42; % Stock price
K = 40; % Strike price
sigma = .4; % Volatility
r = .06; % Riskfree interest rate

dt = T / N;
s = zeros(2 * M, N + 1);
s(:, 1) = S * ones(2 * M, 1);
dc = exp(-r * dt);

tic;

for j = 1: M
    for i = 1: N
        dw = randn * sqrt(dt);
        s(2 * j - 1, i + 1) = s(2 * j - 1, i) + r * s(2 *
            j - 1, i) * dt + sigma * s(2 * j - 1, i) * dw;
        s(2 * j, i + 1) = s(2 * j, i) + r * s(2 * j, i) *
            dt - sigma * s(2 * j, i) * dw;
    end
end

value = zeros(2 * M, N + 1);
value(:, N + 1) = max(K - s(:, N + 1), 0);

interval = .5;

for i = N: -1: 2
```

```
30      dci = dc ^ (i - 1);
31      [sr ind] = sort(s(:, i));
32
33      s(:, 2: i) = s(ind, 2: i);
34      value(:, i + 1) = value(ind, i + 1);
35
36      value(:, i) = dc * value(:, i + 1);
37
38      for j = 1: 2 * M
39          if s(j, i) > K
40              continue
41          end
42          interval = coef * s(j, i);
43
44          frs = j;
45          lst = j;
46          while (s(frs, i) > s(j, i) - interval) && (j - frs
                < 240) && (frs > 5)
47              frs = frs - 5;
48          end
49           while (s(lst, i) < s(j, i) + interval) && (lst -
                j < 250) && (lst < 2 * M - 5)
50               lst = lst + 5;
51           end
52          expec = sum(value(frs: lst, i + 1)) / (lst - frs +
                1);
53
54          if K - s(j, i) < dc * expec
55              value(j, i) = dc * expec;
56          else
57              value(j, i) = K - s(j, i);
58          end
59      end
```

```
60 end
61
62 value(:, 1) = max(K - S, dc * value(:, 2));
63 value = mean(value(:, 1));
64
65 toc;
```

A.5 Optimal Stopping Boundary Estimation Method

```
M = 200000; % Number of trajectories
m = 1000; % Number of trajectories to approximate OST
T = 1; % Maturity time
N = 50 * T; % Discretization points per unit of maturity
    time;
S = 36; % Stock price
K = 40; % Strike price
sigma = .4; % Volatility
r = .06; % Riskfree interest rateeu_value =
    black_scholes_put(S, K, r, sigma, T);

tic;

dt = T / N;
dc = exp(-r * dt);
dci = dc .^ [1: N];

eu_value = black_scholes_put(S, K, r, sigma, T);

OST = zeros(N + 1, 1);
OST(N + 1) = K;

for i = N: -1: 1
    OST(i) = OST(i + 1);
    value = K;
    sub = .1;
    for j = 1: 5
        sub = sub / 10;
        while K - OST(i) < value
            value = black_scholes_put(OST(i), K, r, sigma,
                (N - i + 1) * dt);
```

```
29              OST(i) = OST(i) - sub;
30          end
31          OST(i) = OST(i) + sub;
32      end
33  end
34
35  AST = OST;
36
37  for i = N - 1: -1: 1
38      AST(i) = min(AST(i), AST(i + 1));
39
40      sub = 1;
41      for l = 1: 5
42          value = K;
43          sub = sub / 10;
44          while K - AST(i) < value
45              AST(i) = AST(i) - sub;
46
47              value = 0;
48
49              for j = 1: m
50                  s1 = AST(i);
51                  s2 = AST(i);
52                  cs1 = 1;
53                  cs2 = 1;
54                  for k = i: N
55                      dw = randn * sqrt(dt);
56                      s1 = s1 + r * s1 * dt + sigma * s1 *
                            dw;
57                      s2 = s2 + r * s2 * dt - sigma * s2 *
                            dw;
58                      if cs1 && s1 < AST(k + 1)
59                          value = value + (K - s1) * dci(k -
```

```
                                    i + 1);
60                              cs1 = 0;
61                          end
62                          if cs2 && s2 < AST(k + 1)
63                              value = value + (K - s2) * dci(k -
                                    i + 1);
64                              cs2 = 0;
65                          end
66                          if ~cs1 && ~cs2
67                              break
68                          end
69                      end
70                  end
71                  value = value / (2 * m);
72              end
73              AST(i) = AST(i) + sub;
74          end
75      end
76
77      value = 0;
78
79      for j = 1: M
80          s1 = S;
81          s2 = S;
82          cs1 = 1;
83          cs2 = 1;
84          for i = 1: N
85              dw = randn * sqrt(dt);
86              s1 = s1 + r * s1 * dt + sigma * s1 * dw;
87              s2 = s2 + r * s2 * dt - sigma * s2 * dw;
88              if cs1 && s1 < AST(i + 1)
89                  value = value + (K - s1) * dci(i);
90                  cs1 = 0;
```

```
91          end
92          if cs2 && s2 < AST(i + 1)
93              value = value + (K - s2) * dci(i);
94              cs2 = 0;
95          end
96          if ~cs1 && ~cs2
97              break
98          end
99      end
100 end
101
102 value = value / (2 * M);
103 disp(value);
104 toc
```

References

[1] Athanassios N. Avramidis and Heinrich Matzinger. Convergence of the stochastic mesh estimator for pricing american options. *Proceedings of the Winter Simulation Conference*, pages 1560–1567, 2002. 17

[2] Mark Broadie and Paul Glasserman. Pricing american-style securities using simulation. *Journal of Economic Dynamics and Control*, 21(8-9):1323–1352, June 1997. 1, 9, 12, 18, 21, 22

[3] Paul Glasserman. *Monte Carlo Methods in Financial Engineering (Stochastic Modelling and Applied Probability)*. Springer, August 2003. ISBN 0387004513. 1, 2, 3, 5, 7, 8, 9, 10, 14, 16, 20

[4] Francis A. Longstaff and Eduardo S. Schwartz. Valuing american options by simulation: A simple least-squares approach. *Review of Financial Studies*, 14 (1):113–47, 2001. 1, 3, 24, 29, 31, 54

[5] Broadie Mark and Paul Glasserman. A stochastic mesh method for pricing high-dimensional american options. *Journal of Computational Finance*, 1997. 17

[6] L. C. G. Rogers. Monte Carlo valuation of American options. *Mathematical Finance*, 12(3):271–286, Jul 2002. 1, 3, 33