U.U.D.M. Project Report 2015:13

Evaluating the Longstaff-Schwartz method for pricing of American options

William Gustafsson

Examensarbete i matematik, 15 hp
Handledare: Josef Höök, Institutionen för informationsteknologi
Ämnesgranskare och examinator: Maciej Klimek
Juni 2015

Department of Mathematics
Uppsala University

Evaluating the Longstaff-Schwartz method for pricing of American options

William Gustafsson
Bachelor's Programme in Mathematics, Uppsala University

May 31, 2015

It is ultimately always the subjective value judgments of individuals that determine the formation of prices. [...] The concept of a "just" or "fair" price is devoid of any scientific meaning; it is a disguise for wishes, a striving for a state of affairs different from reality.

– Ludwig von Mises, *Human Action*

Abstract

Option pricing is an important area in the daily activities of banks and other actors in the financial markets. The most common type of options are of American type, which are contracts giving the buyer of the option the *right*, but not the *obligation*, to buy or sell an underlying asset, with the addition that this right can be exercised at any time up until expiry. The last condition means that the pricing of American options is much harder than the European version, that only allow exercise at the expiration of the contract. A common algorithm for pricing American options is the Longstaff-Schwartz method. This method is relatively easy to understand and implement, but its accuracy is limited due to a number numerical factors. This report will study the accuracy and try to improve my implementation of this algorithm.

Acknowledgements

I would like to thank my supervisor Josef Höök for all his help and support during this project, and for introducing me to the area of financial mathematics.

I would also like to thank my former teacher Behrang Mahjani, for getting me interested in numerical methods and for putting me in contact with Josef in the first place.

Contents

1 Introduction

1.1 Option pricing

Options are one of the most common financial derivatives used on financial markets. They are contracts giving the buyer of the option the *right*, but not the *obligation*, to buy or sell an underlying asset to a specified *strike price*. Options can be of two types, *put* or *call*. A put option gives the owner the right to sell the underlying asset at the agreed upon strike price, and a call option the right to buy the asset for the strike price.

There are several different kinds of options, where the most common are called European and American options. The difference is that a European option only gives the owner the right to exercise at a specific time in the future, whereas an American option can be exercised at any time up until the expiration time.

Arguably one of the most important achievements in the field of financial mathematics was the development of the Black-Scholes model. It is a mathematical model describing the value of an option over time, and it makes it possible to explicitly find the value of a European option, under certain conditions.

The value of the option over time can be described as a partial differential equation, called the Black-Scholes equation, which is

$$\frac{\partial u}{\partial t} + rS\frac{\partial u}{S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - ru = 0$$

where

- $S = S(t)$ is the price of the underlying asset
- $u = u\left(S, t\right)$ is the value of the option
- r is the risk-free interest rate
- t is the time, where
  - $t = 0$ denotes the present time
  - $t = T$ denotes the expiration time

- σ is the volatility of the underlying asset
- K is the strike price

Given the boundary condition

$$u(S,T) = \max(K-S,0)$$

which corresponds to the special case of a European put option, where $\max(K-S,0)$ is the payoff of a put option, the Black-Scholes equation has the solution

$$u(S,t) = K\mathrm{e}^{-r(T-t)}\Phi(-d_2) - S\Phi(-d_1)$$

where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r+\frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = \frac{\ln\left(\frac{S}{K}\right) + \left(r-\frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}} = d_1 - \sigma\sqrt{T-t}$$

and Φ is the cumulative distribution function of the standard normal distribution.

When it comes to American options, the Black-Scholes equation becomes

$$\frac{\partial u}{\partial t} + rS\frac{\partial u}{S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 u}{\partial S^2} - ru \leq 0$$

with the boundary conditions

$$u(S,T) = \max(K-S,0) \text{ and } u(S,t) \geq \max(K-S,0)$$

for an American put. This equation does not have an analytic solution, since the option can be exercised at any time up until expiration. The problem then becomes to find the optimal time to exercise, that is the time when the payoff of immediate exercise is greater than the expected reward of keeping the option.

Since this problem doesn't have an analytical solution a number of numerical methods have been developed, such as finite-difference methods, binomial tree models, Monte Carlo methods, and many more.

This report will focus on the least squares Monte Carlo (LSM) method developed by Longstaff and Schwartz [1].

1.2 Geometric Brownian motion

One of the essential assumptions of the Black-Scholes model is that the underlying asset, most commonly a stock, is modelled as a geometric Brownian motion, and the LSM method is based on the same framework.

First we need to define a standard Brownian motion, or Wiener process.

Definition 1. *A random process $\{W(t)\}$ is said to be a standard Brownian motion on $[0, T]$ if it satisfies:*

(I) $W(0) = 0$

(II) $\{W(t)\}$ *has independent and stationary increments*

(III) $W(t) - W(s) \sim \mathcal{N}(0, t - s)$ *for any* $0 \leq s \leq t \leq T$

(IV) $\{W(t)\}$ *has almost surely continuous trajectories*

From this definition it follows that

$$W(t) \sim \mathcal{N}(0, t) \text{ for } 0 < t \leq T$$

which will be important when simulating the Brownian motion.

Next we need to define a Brownian motion with drift and diffusion coefficient.

Definition 2. *A process $\{X(t)\}$ is said to be a Brownian motion with drift $\mu > 0$ and diffusion coefficient $\sigma^2 > 0$ if*

$$\frac{X(t) - \mu t}{\sigma}$$

is a standard Brownian motion.

This means that

$$X(t) \sim \mathcal{N}\left(\mu t, \sigma^2 t\right) \text{ for } 0 < t \leq T$$

and that we can construct the process $\{X(t)\}$ using a standard Brownian motion $\{W(t)\}$ by putting

$$X(t) = \mu t + \sigma W(t)$$

The process $\{X(t)\}$ also solves the stochastic differential equation (SDE)

$$dX(t) = \mu t + \sigma dW(t) \tag{1.1}$$

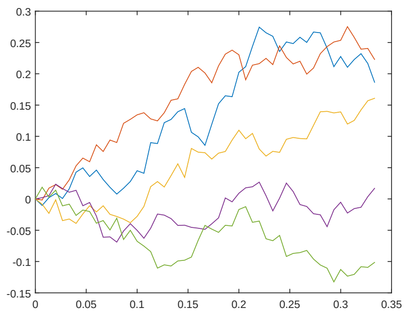
which will be used when we shall construct the geometric Brownian motion.

A Brownian motion is a continuous stochastic process, and as such it cannot be modelled exactly but has to be discretized for a finite number of time steps $t_0 < t_1 < \ldots < t_N$. Let $t_0 = 0$, $X(0) = 0$, and $Z_1, \ldots, Z_N$ be a set of i.i.d. $\mathcal{N}(0,1)$ random variables. Then the process $\{X(t)\}$ can be simulated as

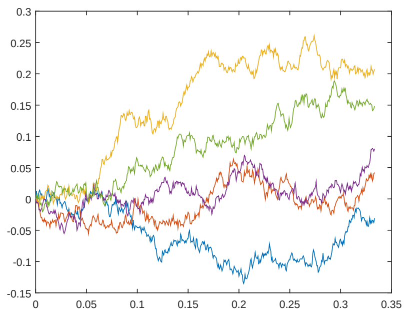
$$X\left(t_{i+1}\right) = X\left(t_i\right) + \mu\left(t_{i+1} - t_i\right) + \sigma\sqrt{t_{i+1} - t_i} Z_{i+1} \tag{1.2}$$

for $i = 0, \ldots, N-1$.

Figure 1.1 shows five simulated paths for a Brownian motion with drift $\mu = 0.0488$ and diffusion coefficient $\sigma^2 = 0.04$, in 1.1a with $N = 50$ and in 1.1b with $N = 500$.



(a) 50 time steps



(b) 500 time steps

Figure 1.1: Simulated Brownian motions

A regular Brownian motion cannot be used as a model for an asset price, since it can assume negative values. For this we need the geometric Brownian motion (GBM), which is essentially an exponentiated Brownian motion.

Definition 3. *A process $\{S(t)\}$ is said to be a geometric Brownian motion if it solves the stochastic differential equation*

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t)$$

To solve this SDE we need to use some Itô calculus, which is a generalised calculus for stochastic processes. More specifically we will use Itô's lemma, which is a generalisation of the chain rule.

Theorem 1 (Itô's lemma)**.** *Let $X(t)$ be a Brownian motion with drift μ and diffusion coefficient σ^2 satisfying the SDE*

$$dX(t) = \mu t + \sigma dW(t)$$

and let $f(X,t)$ be a twice differentiable function. Then

$$df(X,t) = \left(\frac{\partial f}{\partial t} + \mu \frac{\partial f}{\partial X} + \frac{1}{2}\sigma^2 \frac{\partial^2 f}{\partial X^2}\right) dt + \sigma \frac{\partial f}{\partial X} dW(t)$$

Applying Itô's lemma to the GBM $\{S(t)\}$ we get

$$df(S,t) = \left(\frac{\partial f}{\partial t} + \mu S \frac{\partial f}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2}\right) dt + \sigma S \frac{\partial f}{\partial S} dW(t) \tag{1.3}$$

and if we let $f(S,t) = f(S) = \ln(S)$ we get

$$\frac{\partial f}{\partial t} = 0\,, \quad \frac{\partial f}{\partial S} = \frac{1}{S}\,, \quad \frac{\partial^2 f}{\partial S^2} = -\frac{1}{S^2}$$

which means that equation (1.3) becomes

$$\begin{aligned} d\ln(S) &= \left(\mu S \cdot \frac{1}{S} + \frac{1}{2}\sigma^2 S^2 \cdot \left(-\frac{1}{S^2}\right)\right) dt + \sigma S \cdot \frac{1}{S} dW(t) \\ &= \left(\mu - \frac{1}{2}\sigma^2\right) dt + \sigma dW(t) \end{aligned}$$

Integrating both sides of this equation gives

$$\int_{t_0}^{t_1} d\ln(S) = \int_{t_0}^{t_1} \left(\mu - \frac{1}{2}\sigma^2\right) dt + \int_{t_0}^{t_1} \sigma\, dW(t)$$
$$\iff$$
$$\ln(S(t_1)) - \ln(S(t_0)) = \left(\mu - \frac{1}{2}\sigma^2\right)(t_1 - t_0) + \sigma\left(W(t_1) - W(t_0)\right)$$
$$\iff$$

$$\ln\left(S(t_1)\right) = \left(\mu - \frac{1}{2}\sigma^2\right)(t_1 - t_0) + \sigma\left(W(t_1) - W(t_0)\right) + \ln\left(S(t_0)\right)$$

Exponentiating both sides of this yields the solution

$$S(t_1) = S(t_0)\mathrm{e}^{\left(\mu - \frac{1}{2}\sigma^2\right)(t_1 - t_0) + \sigma(W(t_1) - W(t_0))} \tag{1.4}$$

Putting $t_0 = 0$ and $t_1 = t$ gives the more common expression

$$S(t) = S(0)\mathrm{e}^{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t)}$$

Comparing Definition 3 to equation (1.1) and Definition 2 we see that this process has drift $\mu S(t)$ and diffusion coefficient $\sigma^2 S^2(t)$. But the standard notation for geometric Brownian motions is to refer to μ as the *drift* and σ as the *volatility* of the process.

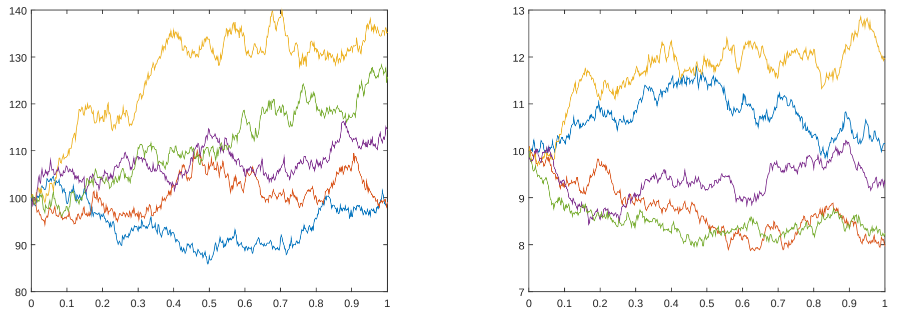
Sine $W(t)$ is a standard Brownian motion, equation (1.4) gives a straight forward way to simulate the GBM $S(t)$ analogue to that of a regular Brownian motion in equation (1.2):

$$S(t_{i+1}) = S(t_i)\mathrm{e}^{\left(\mu - \frac{1}{2}\sigma^2\right)(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i}Z_{i+1}} \tag{1.5}$$

for $i = 0, \dots, N-1$, $Z_1, \dots, Z_N$ i.i.d. $\mathcal{N}(0,1)$ random variables and $S(t_0) = S(0)$ some initial value.

When using a GBM to simulate the price of an asset, the drift μ is often denoted r and represents the risk-free interest rate, as in the Black-Scholes model from section 1.1, and that will be the notation used from now on.

Figure 1.2 shows five simulated paths for a GBM with $r = 0.03$, $\sigma = 0.15$, in 1.2a with $S(0) = 100$ and in 1.2b with $S(0) = 10$, both with 500 time steps.



(a) S(0) = 100

(b) S(0) = 10

Figure 1.2: Simulated geometric Brownian motions

1.3 Monte Carlo

As the name least squares Monte Carlo suggests, the LSM algorithm uses Monte Carlo (MC) to estimate the value of the option.

The general idea of MC methods is to simulate a sample of something you are interested in, and taking the mean to find the "true" value. Mathematically MC is often described as a method to estimate the value of an integral:

$$\mu = \int f(x)\, dx$$

Factorizing the function $f(x)$ as

$$f(x) = h(x)p(x)$$

where $p(x)$ is some density, the integral can be interpreted as the expected value

$$\mu = \mathrm{E}\left[h(x)\right] = \int h(x)p(x)\, dx$$

By generating an i.i.d. sample $x_1, x_2, \ldots, x_n$ from $p(x)$, the expected value can be estimated as

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} h(x_i)$$

Given that $h(x)$ is integrable, the law of large numbers says that

$$\hat{\mu} \to \mu \text{ almost surely as } n \to \infty$$

and if $h(x)$ is also square integrable and we define

$$\sigma^2 = \mathrm{Var}\left[h(x)\right] = \int \left(h(x) - \mu\right)^2 p(x)\, dx$$

it also says that

$$\frac{\sqrt{n}\left(\hat{\mu} - \mu\right)}{\sigma} \to \mathcal{N}\left(0, 1\right)$$

This means that the error $\hat{\mu} - \mu$ is normally distributed with mean 0 and standard deviation $\frac{\sigma}{\sqrt{n}}$, which gives the convergence rate $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ for MC methods. This is both their strength and weakness, as it is a comparatively slow convergence but it does not increase with the dimension of the problem as it does for most other numerical methods. This means that MC methods are not very competitive in one dimension, but as dimensions increase so does their competitiveness.

2 Method

2.1 The LSM algorithm

As mentioned in section 1.1, the problem with pricing American options is that they can be exercised at all times up until the expiration of the option, unlike a European option that can only be exercised at the expiration time. Here we will use the same notation as in section 1.1, letting

$$\begin{aligned} g\left(S(t)\right) &= \max\left(K - S(t), 0\right) \quad \text{for a put option, and} \\ g\left(S(t)\right) &= \max\left(S(t) - K, 0\right) \quad \text{for a call option} \end{aligned}$$

be the payoff at time t, and assume that the underlying asset is modelled using the GBM

$$S(t) = S(0)\mathrm{e}^{\left(r - \frac{1}{2}\sigma^2\right)t + \sigma W(t)}$$

For ease of notation we will only consider put options from now on, but all results are off course applicable to call options as well.

Using the assumptions above, the value at time 0 of a European option can be described as

$$u(S,0) = \mathrm{E}\left[\mathrm{e}^{-rT} g\left(S(T)\right)\right]$$

That is, the expected value of the discounted payoff at time T. In a similar way the value of an American option at time 0 is given by

$$u(S,0) = \sup_{t \in [0,T]} \mathrm{E}\left[\mathrm{e}^{-rt} g\left(S(t)\right)\right], \tag{2.1}$$

the expected value of the discounted payoff at the time of exercise that yields the greatest payoff. The reason for this is the assumption of no arbitrage, that is the chance for risk-free reward, which means the option must cost as much as the maximum expected reward from it. This corresponds to the optimization problem of finding the optimal stopping time

$$t^* = \inf\{t \geq 0 \mid S(t) \leq b^*(t)\}$$

for some unknown exercise boundary b^*, as illustrated in Figure 2.1.

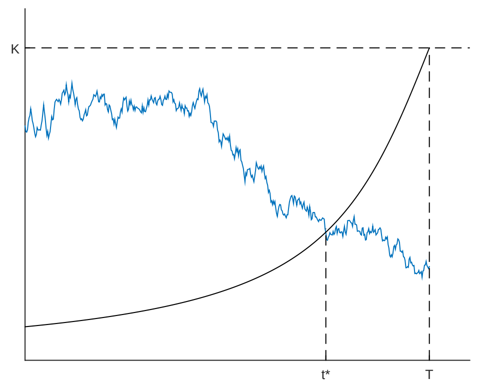


Figure 2.1: Exercise boundary for an American put

So in order to price an American option we need to find the optimal stopping time t^*, and then estimate the expected value

$$u(S,0) = \mathrm{E}\left[\mathrm{e}^{-rt^*} g\left(S(t^*)\right)\right] \tag{2.2}$$

The LSM method, developed by Longstaff and Schwartz in [1], uses a dynamic programming approach to find the optimal stopping time, and Monte Carlo to approximate the expected value. Dynamic programming is a general method for solving optimization problems by dividing it into smaller subproblems and combining their solution to solve the problem. In this case this means that we divide the interval $[0, T]$ into a finite set of time points $\{0, t_1, t_2, \ldots, t_N\}$, $t_N = T$, and for each of these decide if it is better to exercise than to hold on to the option. Starting from time T and working backwards to time 0, we update the stopping time each time we find a time where it is better to exercise until we have found the smallest time where exercise is better.

Let $C\left(S(t_i)\right)$ denote the value of holding on to the option at time t_i, from now on called the *continuation value*, and let the value of exercise at time t_i be the payoff $g\left(S(t_i)\right)$. Then the dynamic programming algorithm to find the optimal stopping time is given by Algorithm 1.

Algorithm 1 Dynamic programming to find optimal stopping time

```
$t^* \leftarrow t_N$
for $t$ from $t_{N-1}$ to $t_1$ do
  if $C(S(t)) < g(S(t))$ then
    $t^* \leftarrow t$
  else
    $t^* \leftarrow t^*$
  end if
end for
```

Using the same arguments as in Equation 2.1, the continuation value at time t_i can be described as the conditional expectation

$$C(S(t_i)) = \mathrm{E}\left[\mathrm{e}^{-r(t^*-t_i)} g(S(t^*)) \mid S(t_i)\right] \tag{2.3}$$

where t^* is the optimal stopping time in $\{t_{i+1}, \ldots, t_N\}$. Since we are using the method described above in Algorithm 1, such a t^* will always exist.

For ease of notation, we define the current payoff $P = P(S(t))$ as:

- For $t = t_N$
  - Let $P = g(S(t))$
- From $t = t_{N-1}$ to $t = t_1$
  - If $C(S(t)) < g(S(t))$, let $P = g(S(t))$
  - Otherwise let $P = \mathrm{e}^{-r\Delta t} P$

Here $t_{i+1} - t_i$ is denoted Δt, as we assume that we have equidistant time points. Given this notation, Equation 2.3 becomes

$$C(S(t_i)) = \mathrm{E}\left[\mathrm{e}^{-r\Delta t} P \mid S(t_i)\right] \tag{2.4}$$

To estimate this conditional expectation, the LSM method uses regular least squares regression. This can be done since the conditional expectation is an element in L^2 space, which has an infinite countable orthonormal basis and thus all elements can be represented as a linear combination of a suitable set of basis functions. So to estimate this we need to choose a (finite) set of orthogonal basis functions, and project the discounted payoffs onto the space spanned by these.

In my implementation the basis functions is chosen to be the Laguerre polynomials[1], where the first four are defined as:

$$\begin{aligned} L_0(X) &= 1 & L_2(X) &= \frac{1}{2}(2-4X+X^2) \\ L_1(X) &= 1-X & L_3(X) &= \frac{1}{6}(6-18X+9X^2-X^3) \end{aligned}$$

Given a set of realised paths $S_i(t)$, $i = 1, \ldots, n$ that are *in-the-money* at time t, i.e. $g(S_i(t)) > 0$, and the payoffs $P_i = P\left(S_i(t)\right)$, the conditional expectation in Equation 2.4 can be estimated as

$$\hat{C}\left(S_i(t)\right) = \sum_{j=0}^{k} \hat{\beta}_j L_j\left(S_i(t)\right) \tag{2.5}$$

where $L_0, \ldots, L_k$ are the k first Laguerre polynomials and $\hat{\beta}_0, \ldots, \hat{\beta}_k$ are the estimated regression coefficients. The regression coefficients are obtained by regressing the discounted payoffs $y_i = \mathrm{e}^{-r\Delta t} P_i$ against the current values $x_i = S_i(t)$ by regular least squares:

$$\left(\hat{\beta}_0, \ldots, \hat{\beta}_k\right)^{\mathrm{T}} = \left(\mathbf{L}^{\mathrm{T}}\mathbf{L}\right)^{-1}\mathbf{L}^{\mathrm{T}}\left(y_1, \ldots, y_n\right)^{\mathrm{T}}$$

where $\mathbf{L}_{i,j} = L_j(x_i)$, $i = 1, \ldots, n$ and $j = 0, \ldots, k$.

By approximating (2.4) with (2.5), we introduce an error in our estimate. In [3] it is shown that

$$\lim_{k\to\infty} \hat{C}\left(S(t)\right) = C\left(S(t)\right)$$

but not at which rate. In section 3.2, the convergence with respect to the number of basis functions is investigated further.

Now that we have a way to estimate the continuation value, we can simulate a set of M realised paths $S_i(t)\,, t = 0, t_1, t_2, \ldots, t_N\,, i = 1, 2, \ldots, M$ and use the method from Algorithm 1 to find optimal stopping times t_i^* for all paths, and then estimate the expected value in Equation 2.2 using Monte Carlo:

$$\hat{u} = \frac{1}{M}\sum_{i=1}^{M} \mathrm{e}^{-rt_i^*} g(S(t_i^*)) \tag{2.6}$$

[1]In their original article, Longstaff and Schwartz used the weighted Laguerre polynomials, but in my implementation the regular polynomials gave better numerical results.

One way to simplify the algorithm is to use the discounted payoffs P_i in the Monte Carlo step instead of the optimal stopping times. Since they are constructed and updated recursively in the same way as the stopping times, by the time we have gone from time $t = t_N$ to $t = t_1$ they will be

$$P_i = \mathrm{e}^{-r(t_i^* - t_1)} g(S(t_i^*))$$

which means that

$$\mathrm{e}^{-r\Delta t} P_i = \mathrm{e}^{-rt_i^*} g(S(t_i^*))$$

and thus Equation 2.6 becomes

$$\hat{u} = \frac{1}{M} \sum_{i=1}^{M} \mathrm{e}^{-r\Delta t} P_i \tag{2.7}$$

The entire LSM algorithm is described in Algorithm 2 below. In each step only the paths that are in-the-money are used, since these are the only ones where the decision to exercise or continue is relevant.

Algorithm 2 LSM Algorithm

```
Initiate paths $S_i(t)\,, t = 0, t_1, t_2, \ldots, t_N\,, i = 1, 2, \ldots, M$
Put $P_i \leftarrow g(S_i(t_N)$ for all $i$
for $t$ from $t_{N-1}$ to $t_1$ do
    Find paths $\{i_1, i_2, \ldots, i_n\}$ s.t. $g(S_i(t) > 0$, i.e. that are in-the-money
    Let $itm\_paths \leftarrow \{i_1, i_2, \ldots, i_n\}$
    Let $x_i \leftarrow S_i(t)$ and $y_i \leftarrow \mathrm{e}^{-r\Delta t} P_i$ for $i \in itm\_paths$
    Perform regression on $x, y$ to obtain coefficients $\hat{\beta}_0, \ldots, \hat{\beta}_k$
    Estimate the value of continuation $\hat{C}\,(S_i(t))$ and calculate the value of
    immediate exercise $g(S_i(t)$ for $i \in itm\_paths$
    for $i$ from 1 to $M$ do
        if $i \in itm\_paths$ and $g(S_i(t) > \hat{C}\,(S_i(t))$ then
            $P_i \leftarrow g(S_i(t)$
        else
            $P_i \leftarrow \mathrm{e}^{-r\Delta t} P_i$
        end if
    end for
end for
$price \leftarrow \frac{1}{M} \sum_{i=1}^{M} \mathrm{e}^{-r\Delta t} P_i$
```

2.2 Improving the algorithm

One problem with the standard LSM-algorithm is that it requires all the realized paths to be initialized and stored for all time steps, since the standard random walk construction of a GBM goes from time 0 to time T, and the algorithm goes in the opposite direction. This takes up a lot of memory on the computer and puts a cap on the accuracy, especially for options with a longer time to expiration.

A more preferable way would be to generate the GBM from time T to time 0, and thus only having to store the values needed for the current step in the algorithm. One way to this is to construct the GBM as a Brownian bridge, which is a way to generate the intermediate points using the conditional distribution given the start and endpoints.

The GBM we want to simulate is

$$S(t) = S(0)\mathrm{e}^{\left(\mu - \frac{1}{2}\sigma^2\right)t + \sigma W(t)}$$

which is the same as simulating the regular Brownian motion

$$X(t) = \mu t + \sigma W(t)\,, \quad \text{where } \mu = r - \frac{1}{2}\sigma^2$$

and letting $S(t) = S(0)\mathrm{e}^{X(t)}$. Since $X(t)$ is normally distributed we can use the following theorem to find the conditional expectation.

Theorem 2. *Let* $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$ *be multivariate normally distributed with mean* $\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$ *and covariance matrix* $\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$*, where* $|\Sigma_{22}| > 0$*. Then the conditional distribution of* X_1 *given that* $X_2 = x_2$ *is given by*

$$(X_1 \mid X_2 = x_2) \sim \mathcal{N}\left(\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}\right)$$

Given that $X(t) \sim \mathcal{N}\left(\mu t, \sigma^2 t\right)$, $\mathrm{Cov}\left[X(s), X(t)\right] = \sigma^2 \min(s,t)$ and assuming that $0 < u < s < t$, the unconditional distribution of $(X(u), X(s), X(t))$ is given by

$$\begin{pmatrix} X(u) \\ X(s) \\ X(t) \end{pmatrix} \sim \mathcal{N}\left(\mu \begin{pmatrix} u \\ s \\ t \end{pmatrix}, \sigma^2 \begin{pmatrix} u & u & u \\ u & s & s \\ u & s & t \end{pmatrix}\right) \tag{2.8}$$

Assuming we want the conditional distribution of $X(s)$ given that $X(u) = x, X(t) = y$, we can rearrange (2.8) to get

$$\begin{pmatrix} X(s) \\ X(u) \\ X(t) \end{pmatrix} \sim \mathcal{N}\left(\mu \begin{pmatrix} s \\ u \\ t \end{pmatrix}, \sigma^2 \begin{pmatrix} s & u & s \\ u & u & u \\ s & u & t \end{pmatrix}\right)$$

Applying Theorem 2 to this we get that $(X(s) \mid X(u) = x, X(t) = y)$ is normally distributed with mean

$$\mu s + \sigma^2(u,s)\left(\sigma^2 \begin{pmatrix} u & u \\ u & s \end{pmatrix}\right)^{-1} \begin{pmatrix} x - \mu u \\ y - \mu t \end{pmatrix} = \frac{(t-s)x + (s-u)y}{t-u} \tag{2.9}$$

and variance

$$\sigma^2 s - \sigma^2(u,s)\left(\sigma^2 \begin{pmatrix} u & u \\ u & s \end{pmatrix}\right)^{-1} \sigma^2 \begin{pmatrix} u \\ s \end{pmatrix} = \sigma^2 \frac{(s-u)(t-s)}{t-u} \tag{2.10}$$

Since a Brownian motion is a type of Markov process, we know that given a set of values $X(t_1), \ldots, X(t_N)$, each $X(t_i)$ is independent of all other except $X(t_{i-1})$ and $X(t_{i+1})$. This means we only have to condition on the values closest to the one we want to simulate, and this gives a way to recursively generate the Brownian bridge backwards from $t_N = T$ to $t_0 = 0$. In fact, since the Brownian motion, and thus the Brownian bridge, is by definition equal to 0 in time 0, we only have to condition on $X(t_{i+1})$ to get the distribution of $X(t_i)$, if we want to go from $X(T)$ to $X(0)$.

So applying Theorem 2 to the conditional distribution of $X(t_i)$ given $X(t_{i+1})$, and letting $X(T) = \left(r - \frac{1}{2}\sigma^2\right)T + \sigma W(T)$, we have that

$$(X(t_i) \mid X(t_{i+1}) = x_{t_{i+1}}) \sim \mathcal{N}\left(\frac{t_i x_{t_{i+1}}}{t_{i+1}}, \sigma^2 \frac{t_i(t_{i+1} - t_i)}{t_{i+1}}\right)$$

for $i = N-1, N-2, \ldots, 1$. So to simulate $X(t)$ backwards we start by generating the endpoint $X(T) = \left(r - \frac{1}{2}\sigma^2\right)T + \sigma Z$, where $Z \sim \mathcal{N}(0,1)$, and then letting

$$X(t_i) = \frac{t_i\, X(t_{i+1})}{t_{i+1}} + \sigma\sqrt{\frac{t_i\, \Delta t}{t_{i+1}}}Z\,, \quad i = N-1, N-2, \ldots, 1$$

Then the GBM $S(t)$ can be generated backwards as well by putting

$$S(t_i) = S(0)\mathrm{e}^{X(t_i)}\,, \quad i = N, N-1, \ldots, 1$$

This means we can start by initiating the M endpoints $S_i(T)$ and then generate the other $S_i(t)$ for each step backwards in the algorithm, only having to store the values needed for the current step. This reduces the memory needed for storing the realised paths by a factor N, the number of time steps.

Algorithm 3 describes the LSM algorithm using this method.

Algorithm 3 LSM Algorithm - Brownian Bridge

```
Generate $X_i(t_N)\,, i = 1, 2, \ldots, M$
Initiate endpoints $S_i(t_N) \leftarrow S(0)\mathrm{e}^{X_i(t_N)}$ for all $i$
Put $P_i \leftarrow g(S_i(t_N)$ for all $i$
for $t$ from $t_{N-1}$ to $t_1$ do
  Generate $X_i(t)$
  Put $S_i(t) \leftarrow S(0)\mathrm{e}^{X_i(t)}$
  Find paths $\{i_1, i_2, \ldots, i_n\}$ s.t. $g(S_i(t) > 0$, i.e. that are in-the-money
  Let itm_paths $\leftarrow \{i_1, i_2, \ldots, i_n\}$
  Let $x_i \leftarrow S_i(t)$ and $y_i \leftarrow \mathrm{e}^{-r\Delta t} P_i$ for $i \in$ itm_paths
  Perform regression on $x, y$ to obtain coefficients $\hat{\beta}_0, \ldots, \hat{\beta}_k$
  Estimate the value of continuation $\hat{C}\left(S_i(t)\right)$ and calculate the value of
  immediate exercise $g(S_i(t)$ for $i \in$ itm_paths
  for $i$ from 1 to $M$ do
    if $i \in$ itm_paths and $g(S_i(t) > \hat{C}\left(S_i(t)\right)$ then
      $P_i \leftarrow g(S_i(t)$
    else
      $P_i \leftarrow \mathrm{e}^{-r\Delta t} P_i$
    end if
  end for
end for
price $\leftarrow \frac{1}{M} \sum_{i=1}^{M} \mathrm{e}^{-r\Delta t} P_i$
```

3 Numerical results

This section will present the numerical results obtained when evaluating my implementation of the algorithm. These kind of results will of course differ between different platforms, but they will give an idea about the general behaviour.

All tests were performed using MATLAB release 2014b, using the code in the appendix. The computer used for the tests have an Intel® Core i5-2500k processor running at 3.3 GHz, and 8 GB of RAM.

3.1 Memory

As indicated in Section 2.2, the memory consumption of the stored paths should decrease by a factor N, the number of discrete time steps, when using the Brownian bridge construction. Figure 3.1 shows the memory consumed by the variable storing the values of the realised paths, for $M = 10^6$ paths, which is the number I have been using for the other tests as well. The Y-axis is on a logarithmic scale, to better fit both plots in the same window.

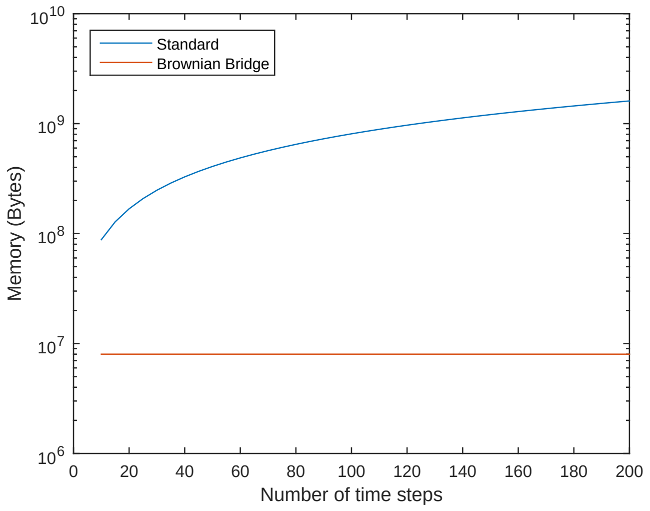


Figure 3.1: Comparison of memory consumption, $M = 10^6$ paths

This only shows the memory used by the realised paths, not the total memory consumption of the algorithm. This is because there is no simple way to monitor memory usage in MATLAB, and no other variables depend on the number of time steps. But the total memory usage is also drastically reduced, since the stored paths is by far the variable taking up the most memory in the standard algorithm.

3.2 Accuracy

There are three main sources of error when using the LSM method. First of all there is the Monte Carlo error, described in Section 1.3. I will not expand more upon this, since the properties of this is well known.

Second there is the discretization error that comes from approximating an American option, which can be continuously exercised, using only a finite set of discrete time points. This approximation will clearly tend to the true value as the number of time points goes to infinity, but this comes at the price of an increasing execution time of the algorithm as the number of time points increase.

Third there is the truncation error when estimating the continuation value in (2.5) using only a finite set of basis functions. As mentioned earlier the rate of convergence for this is unknown, but in practice only a few polynomials are generally used.

In this section the last two errors will be investigated numerically.

To investigate the accuracy, I have compared the results of my algorithm to those used in [4], which were obtained using the method described in Appendix A.1 in the same. These are highly accurate values of an American put option with the parameters

- Volatility $\sigma = 0.15$
- Risk-free interest rate $r = 0.03$
- Strike price $K = 100$
- Time to expiry $T = 1$

and three different initial prices, S_0, of the underlying asset, one deep in-the-money, on at the strike price, and one deep out-of-the-money. The value of the option, u, is given in Table 3.1.

Initial price S_0	Option value u
90	10.726486710094511
100	4.820608184813253
110	1.828207584020458

Table 3.1: Option values for different S_0

The accuracy is measured by calculating the relative error

$$e_r = \left| \frac{\hat{u} - u}{u} \right|$$

for an increasing number of time points and basis functions. Here u is the "true" values above and $\hat{u}$ is a mean of 20 samples calculated for each set of parameters using my implementation of the LSM algorithm. Since $\hat{\mu}$ will be a random variable, the random number generator in MATLAB was reset between each iteration, so that the only difference in error would depend on the number of time steps and basis functions used.

For each initial price S_0, a sample of 20 values was computed using $M = 10^6$ realised paths for 10-200 time steps with increments of 5. This was done for 1-4 basis functions for $S_0 = 90$ and $S_0 = 100$, and 1-5 basis functions for $S_0 = 110$. For each sample the relative error was calculated and the execution time measured, and the results are shown in the following plots.

Figure 3.2 shows the relative error as a function of the number of time steps, for one to four basis functions, for initial price $S_0 = 90$.

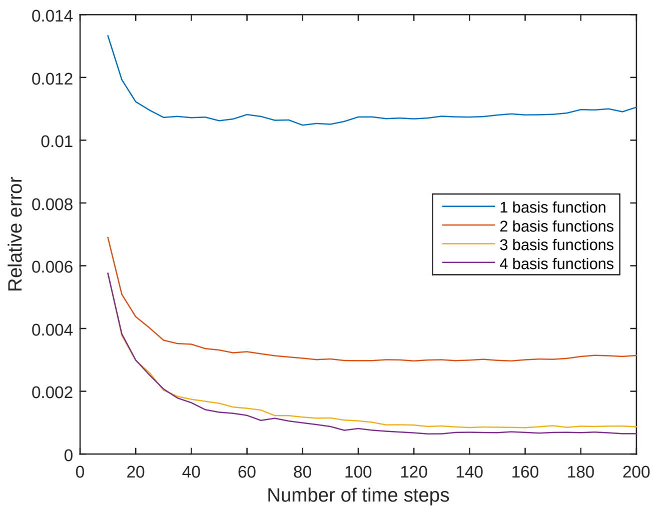


Figure 3.2: Relative error for $S_0 = 90$

Figure 3.3 shows the corresponding execution times.

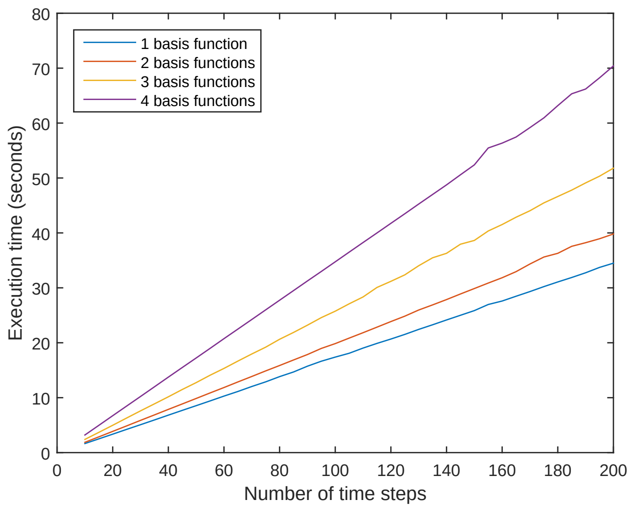


Figure 3.3: Execution time for $S_0 = 90$

Figure 3.4 shows the relative error as a function of the number of time steps, for one to four basis functions, for initial price $S_0 = 100$.

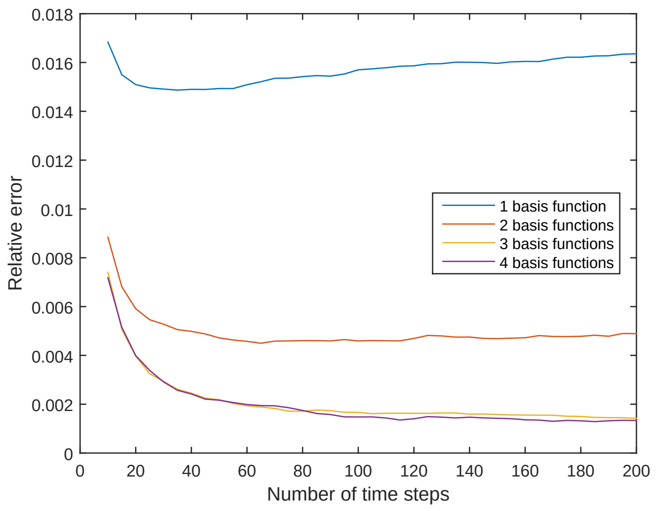


Figure 3.4: Relative error for $S_0 = 100$

Figure 3.5 shows the corresponding execution times.

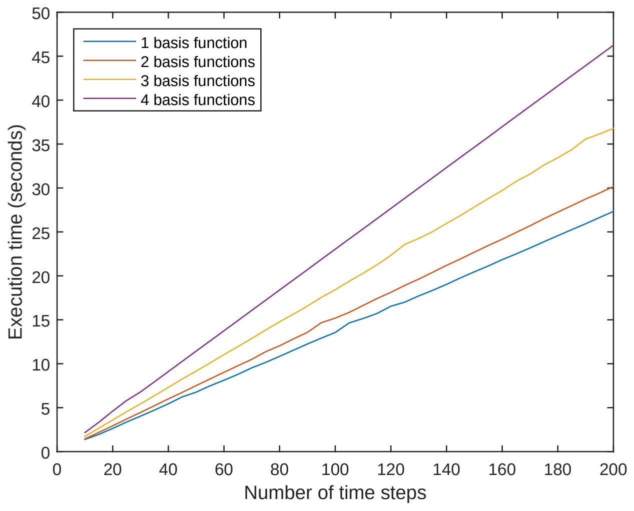


Figure 3.5: Execution time for $S_0 = 100$

Figure 3.6 shows the relative error as a function of the number of time steps, for one to five basis functions, for initial price $S_0 = 110$.

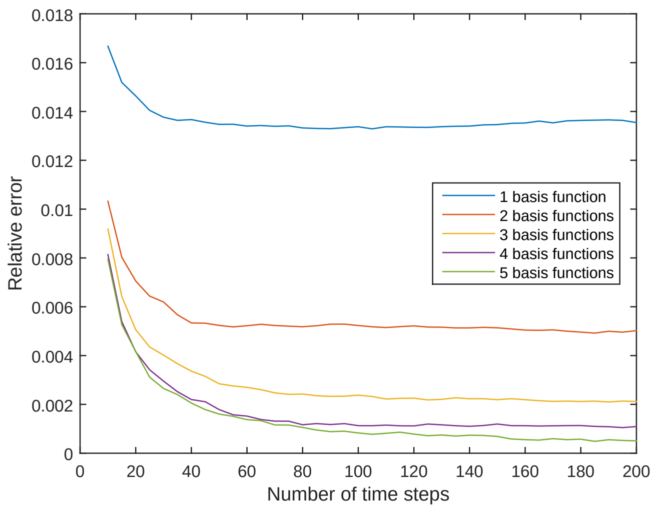


Figure 3.6: Relative error for $S_0 = 110$

Figure 3.7 shows the corresponding execution times.

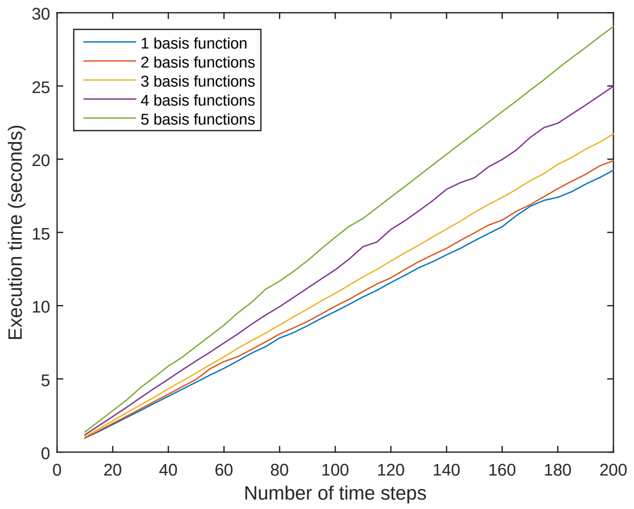


Figure 3.7: Execution time for $S_0 = 110$

In the following plots the actual computed mean option values with confidence intervals are shown together with the true values, for the most accurate number of basis functions.

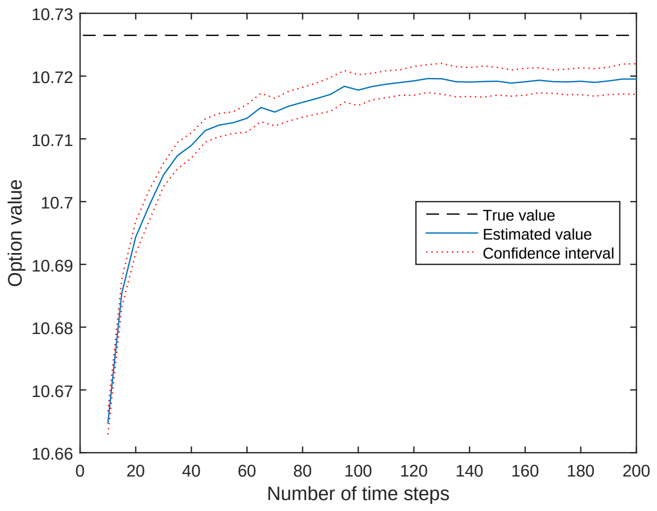


Figure 3.8: Computed option value for $S_0 = 90$

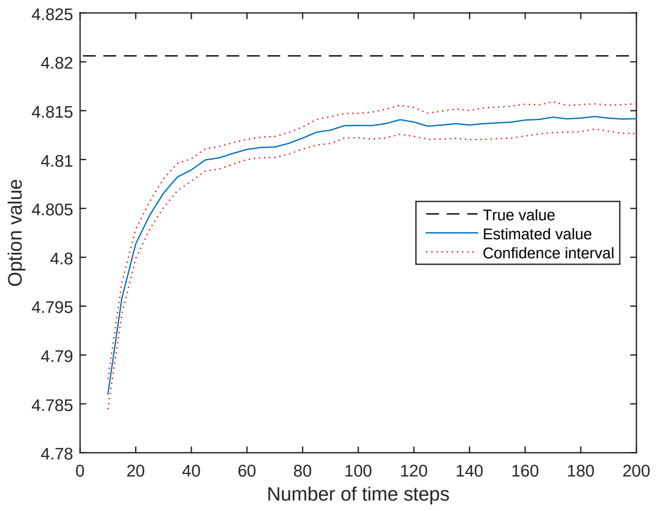


Figure 3.9: Computed option value for $S_0 = 100$

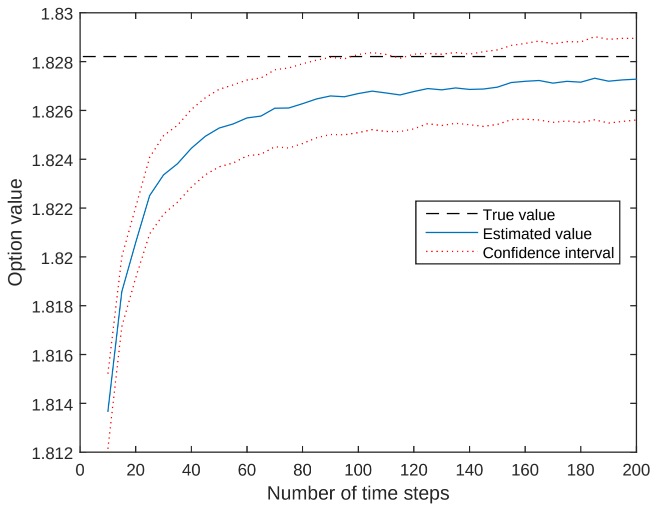


Figure 3.10: Computed option value for $S_0 = 110$

4 Conclusions

As shown in Section 3.1, using the Brownian bridge construction instead of the regular method of storing all paths drastically decreases the memory consumption while not adding any computational difficulty. Indeed it is generated in the same recursive way, the only difference being that it starts from the end time and goes backwards instead the other way around, which suits the LSM algorithm better. Additionally, as is pointed out in [2, p. 86], the Brownian bridge has exactly the same statistical properties as the regular Brownian motion while being easier to control. All this indicates that this method is the preferable method to use when implementing a standard version of the LSM algorithm.

When it comes to the accuracy, the relative errors seemed to converge at around 10^{-3} for all three initial prices. And when plotting the actual computed values, it was only for $S_0 = 110$ that the true value was inside the confidence interval, but all means were consistently lower than the true value. This seems to indicate a systematic downwards bias of the LSM algorithm. It is mentioned in [5, p. 34] that the fact that the LSM method uses the same paths to estimate the exercise boundary as to estimate the value of the option might give an upwards biased result, but because the estimated exercise boundary is generally sub-optimal it will tend to be downwards biased, which is supported by my results.

Different methods for improving the estimate of the exercise boundary have been proposed, see for example [6], which would be needed to further increase the accuracy of the LSM method.

References

[1] Francis A. Longstaff, Eduardo S. Schwartz, Valuing American Options by Simulation: A Simple Least-Squares Approach (The Review of Financial Studies) (2001) Vol 14, No 1, pp. 113-147.

[2] Paul Glasserman, Monte Carlo Methods in Financial Engineering(Springer) (2004)

[3] Clement et al., An analysis of a least squares regression method for American option pricing, Finance and Stochastics (2002)

[4] BENCHOP - The BENCHmarking project in Option Pricing, To appear in the International Journal of Computer Mathematics

[5] Eric Couffignals, Quasi-Monte Carlo Simulations for Longstaff Schwartz Pricing of American Options, University of Oxford, MScMCF: Dissertation (2009)

[6] Nicki S. Rasmussen, Control Variates for Monte Carlo Valuation of American Options, Journal of Computational Finance, Vol. 9, No. 1 (2005)

A MATLAB code

A.1 Main algorithm

```
function u = LSM(T,r,sigma,K,S0,N,M,k)
% T         Expiration time
% r         Riskless interest rate
% sigma     Volatility
% K         Strike price
% S0        Initial asset price
% N         Number of time steps
% M         Number of paths
% k         Number of basis functions

dt = T/N;       % Time steps
t = 0:dt:T;     % Time vector

z = randn(M/2,1);
w = (r-sigma^2/2)*T + sigma*sqrt(T)*[z;-z];
S = S0*exp(w);

P = max(K-S,0);         % Payoff at time T

for i = N:-1:2

    z = randn(M/2,1);
    w = t(i)*w/t(i+1) + sigma*sqrt(dt*t(i)/t(i+1))*[z;-z];
    S = S0.*exp(w);

    itmP = find(K-S>0);         % In-the-money paths

    X = S(itmP);                  % Prices for the in-the-money
                                  % paths
    Y = P(itmP)*exp(-r*dt);       % Discounted payoffs

    A = BasisFunct(X,k);
    beta = A\Y;                 % Regression

    C = A*beta;                 % Estimated value of continuation
    E = K-X;                    % Value of immediate exercise

    exP = itmP(C<E);              % Paths where it's better to
                                  % exercise
    rest = setdiff(1:M,exP);     % Rest of the paths

    P(exP) = E(C<E);             % Better to exercise? Insert value
                                 % in payoff vector

    P(rest) = P(rest)*exp(-r*dt);    % Better to continue? Insert
                                     % previous payoff and discount
                                     % back one step

end

u = mean(P*exp(-r*dt));        % Value of option
```

A.2 Basis functions for regression

```
function A = BasisFunct(X,k)

if k == 1
    A = [ones(size(X)) (1-X)];
elseif k == 2
    A = [ones(size(X)) (1-X) 1/2*(2-4*X+X.^2)];
elseif k == 3
    A = [ones(size(X)) (1-X) 1/2*(2-4*X+X.^2) ...
        1/6*(6-18*X+9*X.^2-X.^3)];
elseif k == 4
    A = [ones(size(X)) (1-X) 1/2*(2-4*X+X.^2) ...
        1/6*(6-18*X+9*X.^2-X.^3) ...
        1/24*(24-96*X+72*X.^2-16*X.^3+X.^4)];
else if k == 5
    A = [ones(size(X)) (1-X) 1/2*(2-4*X+X.^2) ...
        1/6*(6-18*X+9*X.^2-X.^3) ...
        1/24*(24-96*X+72*X.^2-16*X.^3+X.^4) ...
        1/120*(120-600*X+600*X.^2-200*X.^3+25*X.^4-X.^5)];
else
    error('Too many basis functions requested');
end

end
```