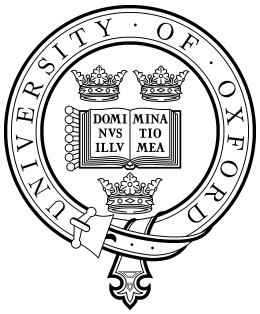
University of Oxford
Institute for Mathematical Finance

Longstaff Schwartz Pricing of Bermudan Options and their Greeks



Howard Thom
Wolfson College

Supervisor: Dr Lajos Gyurko

A dissertation submitted for the degree of
Masters of Science
Trinity 2009

Contents

List of Figures

List of Tables

1. Introduction

In this thesis, I investigated the theory and practice of pricing 1 and 2 dimensional Bermudan options and their Greeks by the regression method developed by Longstaff and Schwartz [2], and by an adaptation of the Likelihood Ratio and Pathwise Sensitivity method. I also implemented a Hedging Error system for measuring the accuracy of my estimates for the value and delta of these Bermudan options.

My first task was to implement this (low-biased) method, and to investigate its convergence behaviour over different numbers of basis functions and paths. In doing so, I was implicitly verifying the convergence results of Clement et al [4], where it is proved that the Longstaff-Schwartz method converges to the correct value as the numbers of basis functions and paths are increased to infinity. I also investigated the result due to Glasserman and Yu [1], who proved that, in certain cases, the number of paths required for convergence should grow exponentially with the number of basis functions.

For comparison with these results, I investigated and implemented the alternative High-Biased estimate developed by Rogers [8], and Haugh and Kogan [9], and outlined in Glasserman [5]. The convergence of this algorithm was also briefly verified and compared with the low-biased estimate. In order to have control values with which to compare my estimates, I also implemented the Binomial Tree method developed by Cox et al [7], and the finite-difference theta-scheme outlined by Reisinger [10].

With my pricing algorithm implemented, its convergence properties checked, and its estimates compared with several control methods, I moved on to pricing Greeks. I decided to use two existing methods: Pathwise Sensitivities, and the Likelihood Ratio method. To do this, I had to adapt them slightly, as there was a concern that if the stopping rule depended on the parameter being varied, the method would not work. For the case of the Delta and the Gamma however, I was able to prove by a backward induction method that both methods were valid, thanks to ideas outlined by Gyurko [3]. I implemented these approaches and checked their convergence properties. For comparison methods, I used finite difference sensitivities of the theta-scheme and the binomial tree.

With a method for pricing the Delta in place, I was able to implement a program to simulate the expected Hedging Error of the Longstaff-Schwartz method. This program allowed me to present a histogram of the simulated hedging errors, which appromimates their distribution. I developed the ideas for this method thanks to suggestions by Gyurko [3].

In the case of pricing the rho and vega by the Pathwise method, I was unable to prove that the method was valid. I decided to assume that the technique was correct, and to investigate the results it produced. Surprisingly, my investigations gave very strong evidence that the technique is valid in these cases. This will be seen from the

convergence properties and by comparison with the control methods.

With these techniques in place and reasonably well tested, I tried implementing them for a non-trivial basket-put option with a payoff depending on two correlated Geometric Brownian Motions $\mathbf{X}$ and $\mathbf{Y}$

$$f(x, y) = \max(K - \max(x, y))$$

For this 2 dimensional Bermudan option, I found estimates for its value for different numbers of paths and basis functions. In order to demonstrate the main advantage of the Longstaff-Schwartz method (that the number of simulations required to obtain the same accuracy does not grow exponentially with the dimensionality), I compared the convergence properties with the 1 dimensional case. I also implemented the Pathwise sensitivity method for the delta of this option, and for a control method I used a finite-difference bumping of the value estimate. Despite the fact that I was unable to prove that the formulae for the rho and vega were valid, I implemented the method for these greeks and found that they compared well with the control method, and had good convergence properties.

The material in this thesis is divided as follows. In Chapter 2 I shall explain the general background and theory of the Longstaff-Schwartz algorithm, including the convergence properties and the different methods of pricing greeks that I investigated. This chapter also includes a discussion of the control methods. Chapter 3 explains some of the particular theoretical changes necessary to implement these techniques to a higher-dimensional option. I will discuss implementation issues and sources of error in my code in Chapter 4, and I will present my results in Chapter 5.

2. Theory and Algorithms

2.1. Longstaff Schwartz

2.1.1. The Basic Algorithm

Here I will outline the basics of the least squares regression method in approximating the price of Bermudan options. This is mostly derived from the treatment by Clement et al [4], Longstaff and Schwartz [2] and from chapter 8 of Glasserman's book [5].

The option $\mathbf{V}$ on the underlying $\mathbf{X}$ we are valuing consists of a payoff function $\mathbf{h}(\cdot)$ and a set of exercise dates $t_1, \ldots, t_M$. At each exercise date, the option holder has the choice to receive the payoff or to wait until the next exercise date, at which time they have the same choice. At expiry time $\mathbf{T}$ the holder either exercises, or the option expires worthless. Our problem is to find a stopping rule, which would tell the holder when to exercise the option. Such a stopping rule could be found if we were able to find the expectation of the option's value at time-step t_{i+1} conditional on the value of the underlying at time t_i:

$$\mathbf{E}[V(X_{i+1})|X_i]$$

The main idea of the regression method is to approximate this conditional expectation at each time-step using a set of $\mathbf{R+1}$ basis functions $\phi_0, \ldots, \phi_R$. i.e.

$$\mathbb{E}[V(X_{i+1})|X_i] \approx \sum_{r=0}^{R} \beta_{i,r}\phi_r(X_i)$$

Before continuing, it is important to note that there are restrictions on the basis functions, as noted in Clement et al[4] and in Glasserman and Yu [1]. For our purposes it is enough for them to be complete and linearly independent, and a set of functions which satisfies these properties is that of the Weighted Laguerre polynomials, as chosen in Longstaff and Schwartz [2]. They are defined as

$$\begin{aligned}
\phi_0(x) &= e^{-x/2} \\
\phi_1(x) &= e^{-x/2}(1-x) \\
\phi_2(x) &= e^{-x/2}(1-2x+x^2/2) \\
\phi_r(x) &= e^{-x/2}\frac{e^x}{r!}\frac{d^r}{dx^r}(x^r e^{-x})
\end{aligned}$$

They are simply $e^{-x/2}$ times the Laguerre polynomials.

We want to minimise the expected squared error in this approximation w.r.t. the coefficients $\beta_{i,r}$, so we differentiate

$$\mathbb{E}\left[\left(\mathbb{E}[V(X_{i+1})|X_i] - \sum_{r=0}^{R} \beta_{i,r}\phi_r(X_i)\right)^2\right]$$

w.r.t. $\beta_{i,r}$ and set the result equal to zero. This gives us:

$$\mathbb{E}\left[\mathbb{E}\left[V(X_{i+1})|X_i\right]\phi_r(X_i)\right] = \sum_{r=0}^{R} \beta_r \mathbb{E}\left[\phi_r(X_i)\phi_s(X_i)\right]$$

It is now convenient to work in a matrix formulation by defining

$$(\mathbf{M}_{\phi\phi})_{r,s} = \mathbb{E}\left[\phi_r(X_i)\phi_s(X_i)\right]$$

and

$$(\mathbf{M}_{V\phi})_r = \mathbb{E}\left[\mathbb{E}[V(X_{i+1})|X_i]\phi_r(X_i)\right]$$

However, $\phi_r(X_i)$ is measurable with respect to X_i so we can write

$$(\mathbf{M}_{V\phi})_r = \mathbb{E}\left[\mathbb{E}\left[V(X_{i+1})\phi_r(X_i)|X_i\right]\right]$$

which, by the Tower rule, gives

$$(\mathbf{M}_{V\phi})_r = \mathbb{E}\left[V(X_{i+1})\phi_r(X_i)\right]$$

and then inverting

$$\beta = \mathbf{M}_{\phi\phi}^{-1}\mathbf{M}_{V\phi}$$

In order to find these coefficients, we perform a monte-carlo simulation of the underlying $X_0, \ldots, X_M$. We generate $\mathbf{N}$ paths and set

$$(\widehat{\mathbf{M}}_{V\phi})_r = \frac{1}{N}\sum_{n=0}^{N} V(X_{i+1}^{(n)})\phi_r(X_i^{(n)})$$

and

$$(\widehat{\mathbf{M}}_{\phi\phi})_{r,s} = \frac{1}{N}\sum_{n=0}^{N} \phi_r(X_i^{(n)})\phi_s(X_i^{(n)})$$

With these approximations, we are now in a position to define a stopping rule and implement an algorithm for pricing a Bermudan option. The algorithm begins with a regression step:

- Generate **N1** paths of **M** timesteps.
- Set $V_M = h(X_M)$ as the terminal condition of each path.

- Work backwards from **M** to **0**, performing the following steps at the ith time-step
  - Calculate the approximations $(\widehat{\mathbf{M}}_{\phi\phi})_{r,s}$ and $(\widehat{\mathbf{M}}_{V\phi})_r$
  - Invert to find $\hat{\beta}_i = \widehat{\mathbf{M}}_{\phi\phi}^{-1}\widehat{\mathbf{M}}_{V\phi}$
  - Now calculate $C_{i+1}^R(X_i) = \sum_{r=0}^R \hat{\beta}_{i,r}\phi_i(X_i)$.
  - Compare this value with the payoff $h(X_i)$
  - Set $V_i = h(X_i)$ if $h(X_i) > C_{i+1}^R(X_i)$ and $V_i = e^{-r\Delta t}V_{i+1}$ otherwise.

This gives both the regression and a possible value for V_i. However, since we are using the same paths for regression as for evaluation, the estimate for V_i could be high biased. In order to ensure that it is low biased, we use a new set of **N2** paths for the evaluation. We now have a choice between working forwards or working backwards. Working backwards would give the following procedure.

- Generate **N2** paths of **M** timesteps.
- Set $V_M = h(X_M)$ for each path.
- Working backwards from **M** to **0**, at each timestep i we:
  - Calculate $C_{i+1}^R(X_i) = \sum_{r=0}^R \beta_{i,r}\phi_i(X_i)$.
  - Compare this value with the payoff $h(X_i)$
  - Set $V_i = h(X_i)$ if $h(X_i) > C_{i+1}^R(X_i)$ and $V_i = e^{-r\Delta t}V_{i+1}$ otherwise.
- Then average over the **N2** values for V_0 to find our estimate.

This backward method is not completely efficient however. For any path that has more than one date on which we find we should exercise, this method will do the calculations at all the dates, rather than just the earliest. If we worked forwards, we could stop after the first exercise date, and save ourselves some computation time. Here is the forward method.

- Generate **N2** paths of **M** timesteps.
- For each path, perform the following steps.
  - Count from exercise date 1 to **M**
  - At each time-step i, $t_i = i\Delta t$
  - Find $C_{i+1}^R(X_i) = \sum_{r=0}^R \beta_{i,r}\phi_i(X_i)$
  - If $C_{i+1}^R < h(X_i)$ or i=**M**, set $V_i(0) = e^{-i\Delta t}h(X_i)$
  - Otherise continue to next time-step.
- Average the V_0 over all **N2** paths.

It is important to note that the Longstaff-Schwartz algorithm does not compare well with our control methods (Binomial Tree and Theta-scheme) in lower dimensions, in terms of convergence or efficiency. In higher dimensions however, the control methods will require exponential increases in the amounts of computation to get comparable results, and it is in this case that the Longstaff-Schwartz method wins out. I will later show how to apply the Longstaff-Schwartz algorithm to the case of 2-dimensional paths, a case where the Binomial tree, for example, would become prohibitavely slow to work with.

2.1.2. Convergence theory

The analysis of the convergence of the Longstaff-Schwartz algorithm is far from trivial. In the original paper [2], Longstaff and Schwartz give few details of the convergence of their algorithm, except in a simple one period case. Clement et al [4] give a more detailed analysis of the convergence properties, but showing that it converges and finding results on the rate of convergence. Glasserman and Yu [1] give an analysis of the relationship between the number of paths required for a particular number of basis functions in order to ensure convergence.

One difficulty in analysing the convergence of Longstaff-Schwartz is that there are various sources of error. The first source is that we are approximating the conditional expectation of our payoff by a linear combination of a finite set of **R+1** basis functions

$$\mathbb{E}\left[V(X_{i+1}|X_i)\right] \approx \sum_{r=0}^{R} \beta_{i,r}\phi_r(X_i)$$

As $R \Rightarrow \infty$, this will become exact. In a practical setting, however, it is not possible to use an infinite set of basis functions! Using this approximation to the continuation value, we find an approximate (and sub-optimal) stopping rule for the option.

A further source of error is that the coefficients $\beta_{i,r}$ are being approximated by a Monte-Carlo regression, so that we are using a set of **R+1** $\hat{\beta}$s whose MSE will be related to the number of paths **N** that are used in the approximation. Once we have this set of regressed $\hat{\beta}$s, we use it to approximate a stopping rule and perform another Monte-Carlo simulation, introducing more error, to find $\hat{V_0}$, the value of our option.

If you are using Longstaff-Schwartz to approximate the value of American options, there will be another source of error based on the number of exercise dates **M** you use in the algorithm. However, in this project I am restricting myself to the case of Bermudan options, so I do not consider this error.

In the paper due to Clement et al [4], he proves two theorems which are relevant to the sources of error described above. In Theorem 3.1, they take a sequence of measurable, real-valued basis functions $\phi_r(X_i)$ satistfying the assumption that for i=1 to M-1, it is total in $\mathbb{L}^2(\sigma(X_i))$, and show that

$$\lim_{R\to\infty} \mathbb{E}\left[f(X_{\tau_i^{(R)}})|\mathcal{F}_i\right] = \mathbb{E}\left[f(X_{\tau_i})|\mathcal{F}_i\right] \text{ in } L^2$$

where τ_i is the optimal stopping time at time-step i, and τ_i^R is the approximation to this stopping time using R basis functions. This covers the first source of error. In Theorem

3.2, they further prove that so long as $\mathbb{P}(\beta_i \cdot \phi(X_i) = f(X_i)) = 0$, then for a Monte-Carlo simulation of $\mathbf{N}$ paths $X^{(1)}, \ldots, X^{(N)}$

$$\frac{1}{N}\sum_{n=1}^{N} f(X^{(n)}_{\tau_i^{n,R,N}}) \text{ converges almost surely to } \mathbb{E}\left[f(X_{\tau_i^{(R)}})\right]$$

as $\mathbf{N}$ goes to infinity, for $i = 1, \ldots, M$, and where $\tau_i^{n,R,N}$ is the n^{th} Monte-Carlo estimate of τ_i^R, the approximation to the optimal stopping time. This covers the convergence of the Monte-Carlo estimate of the approximate sub-optimal stopping rule.

In Glasserman and Yu [1], the relationship between the number of paths and the number of basis functions was investigated. They analysed the MSE of the βs, and placed bounds on the number of paths required for this quantity to converge in the cases where $\mathbf{X}$ (the defining asset process) is Brownian Motion and Geometric Brownian Motion. In both the normal and log-normal setting, they found that the number of paths required for convergence of $MSE(\beta)$ grew approximately exponentially with the number of basis functions. Their results suggest that I should be careful to use enough paths in my regression step, as otherwise I may not get convergence of he βs. It also suggests that an investigation into the convergence of the $MSE(\beta)$ would be a worthwhile check on the stability of my methods.

2.1.3. Alternate High Biased Estimate

As was previously noted, the version of Longstaff-Schwartz that I outlined, where we use one set of paths for the regression step and a different set for the evaluation step, is low biased. This is because our stopping rule will be sub-optimal. In order to find an upper bound, an alternative method can be used. I shall outline here the duality approach, largely following and Glasserman's [5] discretised development of Haugh and Kogan [9] and Rogers [8].

We begin by letting M_k for $k = 0...m$ be a martingale with $M_0 = 0$. For any stopping time τ and the immediate exericse value $h_\tau(X_\tau)$ we have, from the optimal stopping rule, that

$$\mathbb{E}\left[h_\tau(X_\tau)\right] = \mathbb{E}\left[h_\tau(X_\tau) - M_\tau\right] \leq \mathbb{E}\left[\max_{k=1...m}(h_k(X_k) - M_k)\right]$$

Since this is true for all martingales M_k, we have that

$$\mathbb{E}\left[h_\tau(X_\tau) - M_\tau\right] \leq \inf_M \mathbb{E}\left[\max_{k=1,\ldots,m}(h_k(X_k) - M_k)\right]$$

This holds for all stopping times τ, so it will hold for the supremum over the stopping times, which is equal to $V_0(X_0)$ by definition. So

$$V_0(X_0) = \sup_\tau \mathbb{E}\left[h_\tau(X_\tau) - M_\tau\right] \leq \inf_M \mathbb{E}\left[\max_{k=1,\ldots,m}(h_k(X_k) - M_k)\right]$$

This is the key duality approach, and Rogers [8] has infact proven that equality holds in the continuous case. For Bermudan options, we only need equality in the discrete case,

which is what Glasserman provides. He lets

$$\Delta_i = V_i(X_i) - \mathbb{E}\left[V_i(X_i)|X_{i-1}\right]$$

where V_i is the exact value of Bermudan Option at time i, for $i = 1, \ldots, m$. He then sets

$$M_k = \Delta_1 + \ldots + \Delta_k$$

Clearly, we have the martingale property

$$\mathbb{E}\left[M_k|X_{k-1}\right] = M_{k-1}$$

since

$$\mathbb{E}\left[\Delta_i|X_{i-1}\right] = 0$$

Glasserman proceeds by induction to find that

$$V_0(X_0) = \max_{k=1,\ldots,m} \left(h_k(X_k) - M_k\right)$$

which verifies the duality result

$$\sup_{\tau} \mathbb{E}\left[h_\tau(X_\tau)\right] = \inf_{M} \mathbb{E}\left[\max_{k} \left(h_k(X_k) - M_k\right)\right]$$

in descrete time.

In practice, we are not using the exact values for V_i, but a Monte Carlo estimate $\hat{V}_i$ obtained using the Longstaff-Schwartz algorithm. This has been empirically verified to give a good approximation to the upper bound. In order to do this, we need to find

$$M_k = \sum_{1}^{k} \left(\hat{V}_i(X_i) - \mathbb{E}\left[\hat{V}_i(X_i)|X_{i-1}\right]\right)$$

for any particular path consisting of the m simulated points $X_0, X_1, \ldots, X_m$. To estimate the expectation, we simulate P minipaths from each point X_{i-1}, and average

$$\mathbb{E}\left[\hat{V}_i(X_i)|X_{i-1}\right] \approx \frac{1}{P}\sum_{p=1}^{P} \hat{V}_i(X_{i-1}^{(p)})$$

where

$$\hat{V}_i(X_i^{(p)}) = \max\left(h_i(X_i), C_i^R(X_i)\right)$$

and $C_i^R(X_i)$ is the estimate of X_{i+1} based on Longstaff-Schwartz approximation with R basis functions. This gives

$$\hat{M}_k = \sum_{1}^{k} \left(\hat{V}_i(X_i) - \frac{1}{P}\sum_{p=1}^{P} \hat{V}_i(X_{i-1}^{(p)})\right)$$

And we can write

$$V_{upper} \approx \frac{1}{N}\sum_{n=1}^{N} \max_{k=1,\ldots,m} \left(h_k(X_k^{(n)}) - \hat{M}_k^{(n)}\right)$$

where N is the number of simulated paths. In the form of an algorithm

- Generate N paths.
- For each path n.
  - For each time-step k generate P mini-paths.
  - For each mini-path p find $\hat{V}_i(X_i^{(p)}) = \max(h_i(X_i), C_i^R(X_i^{(p)})$
  - Calculate $\hat{M_k}^{(n)}$
  - Find $\max_k \left(h_k(X_k^{(n)}) - \hat{M_k}^{(n)}\right)$
- Now average these estimates over the N paths.

2.1.4. Pathwise greeks

In this section, I will outline the theory of deriving greeks by pathwise sensitivities, and show how it applies in the case of Longstaff-Schwartz. The ideas in this section are largely built upon those due to Gyurko [3].

First note that at maturity **T**, which is time-step **M**, the value of the option only depends on the observed share price S_M:

$$V_M(S_M) = f(S_M)$$

where V_i is the value function of the Burmudan option at time-step i. At some time-step $i-1$, the value of the option depends on the discounted conditional expectation of V_i and on S_{i-1}

$$V_{i-1}(S_{i-1}) = \max\{f(S_{i-1}), e^{-r\Delta t}\mathbb{E}\left[V_i|S_{i-1}\right]\} \tag{2.1}$$

By the Markovian property of the process $(S_i)_{i\geq 0}$, the conditional expectation does not depend on $S_0, \ldots, S_{i-2}$, but only on S_{i-1}.

By backwards induction, one can prove that V_i depends only on S_i and does not depend on $S_0, \ldots, S_{i-1}$. Let $\theta = S_0$ if $S_t \in \Re$, otherwise (if $S_t \in \Re^d$, the higher dimensional case) let θ be one of the coordinates of $S_0 = (S_0^1, \ldots, S_0^d)$. From here on, we will regard S_i as a function $S_i(\theta)$ of θ. The option price can be written as follows

$$\begin{aligned} V_0(S_0(\theta)) = \quad & \mathbb{E}[e^{-rt_1} f(S_1(\theta)) \mathbb{1}_{\{f(S_1(\theta)) \geq V_1(S_1(\theta))\}} \\ & + e^{-rt_2} f(S_2(\theta)) \mathbb{1}_{\{f(S_1(\theta)) < V_1(S_1(\theta))\}} \mathbb{1}_{f(S_2(\theta)) \geq V_2(S_2(\theta))\}} \\ & + \ldots \\ & + e^{-rt_M} \{f(S_M(\theta)) \mathbb{1}_{\{f(S_1(\theta)) \leq V_1(S_1(\theta))\}} \\ & \cdots \mathbb{1}_{\{f(S_{M-1}(\theta)) \leq V_{M-1}(S_{M-1}(\theta))\}} \mathbb{1}_{\{f(S_M(\theta)) \geq 0\}}\}] \end{aligned}$$

Note that by the arguements above, the functions $V_1, \ldots, V_m$ (and hence the stopping rule) don't depend on the choice of θ. If the process $(S_i(\theta))_{i\geq 0})$ is (almost surely) continuous with respect to the parameter θ for each fixed i, then on each path,

$$\mathbb{1}_{\{f(S_i(\theta)) < V_i(S_i(\theta))\}} = \mathbb{1}_{\{f(S_i(\hat{\theta})) < V_i(S_i(\hat{\theta}))\}}$$

whenever f is continuous and $|\theta - \hat{\theta}| < \epsilon$, where ϵ might depend on the path. This implies that the pathwise delta and gamma will work (so long as the differentian and the expection can be swapped, which they can under certain reasonable assumptions). We have

$$\frac{\partial V_0(S_0(\theta))}{\partial \theta} = \mathbb{E}\left[e^{-r\tau}\frac{\partial f(S_\tau(\theta))}{\partial S_\tau}\frac{\partial S_\tau(\theta)}{\partial \theta}\right], \tag{2.2}$$

where τ is the stopping time. In d dimensions, i.e. when $S_i = (S_i^1, \ldots, S_i^d)$ and $f : \Re^d \Rightarrow \Re$:

$$\frac{\partial V_0(S_0(\theta))}{\partial \theta} = \mathbb{E}\left[e^{-r\tau}\sum_{j=1}^{d}\frac{\partial f(S_\tau(\theta))}{\partial S_\tau^j(\theta)}\frac{\partial S_\tau^j(\theta)}{\partial \theta}\right] \tag{2.3}$$

In our case, the gamma will not work by this method, as the payoffs of our options (a put and a basket put on the maximum of some assets) will not be twice differentiable.

In order to calculate the different greeks, we therefore require $\frac{\partial f(S)}{\partial S}$ for a particular payoff $f(S)$. In our base case, we are calculating for a put with strike K, so the payoff is

$$f(S_t) = (K - S_t)_+$$

and so the derivative is

$$\frac{\partial f(S)}{\partial S} = \mathbb{1}_{K>S} \tag{2.4}$$

In calculating the $\frac{\partial S}{\partial \theta}$ term, we first note that

$$S_\tau = S_0 e^{(r-\frac{1}{2}\sigma^2)\tau+\sigma W_\tau}$$

so we can now write down the derivative w.r.t. S_0

$$\frac{\partial S_\tau}{\partial S_0} = e^{(r-\frac{1}{2}\sigma^2)\tau+\sigma W_\tau} \tag{2.5}$$

which gives the delta.

Note that the conditional expectation in 2.1 does depend on r and σ, so the above argument does not work for the vega and rho. However, there is strong empirical evidence that the method continues to be valid, as will be seen in section 5.1. We therefore naively assume that equation 2.2 holds in these cases. The derivitive w.r.t. σ

$$\frac{\partial S_\tau}{\partial \sigma} = (W_\tau - \sigma\tau)e^{(r-\frac{1}{2}\sigma^2)\tau+\sigma W_\tau} \tag{2.6}$$

which gives the vega. There is a slight modification necessary to calculate sensitivity to the risk free rate, or rho, because of the dependence of the discounting factor upon it. Since

$$V_0 = \mathbf{E}[e^{-r\tau}f(S_\tau]$$

we have

$$\rho_0 = \frac{\partial V_0}{\partial r} = \mathbf{E}\left[-\tau e^{-r\tau}f(S_\tau) + e^{-r\tau}\frac{\partial f}{\partial S}\frac{\partial S}{\partial r}\right] \tag{2.7}$$

where

$$\frac{\partial S}{\partial r} = \tau S_0 e^{(r-\frac{1}{2}\sigma^2)\tau+\sigma W_\tau} \tag{2.8}$$

2.1.5. Likelihood ratio greeks

I will now outline the theory behind the likelihood ratio method of deriving greek, as applied to the Longstaff-Schwartz algorithm. The ideas for this section are again built on those due to Gyurko [3].

The arguments of the previous section imply (note that there is no exercise allowed before time-step 1):

$$V_0(S_0) = \mathbb{E}\left[e^{-r\Delta t}V_1(S_1)|S_0\right] = \int e^{-r\Delta t}V_1(S_1)g_1(S_0,S_1)dS_1$$

where $g(S_0,S_1)$ is the transition density of (S_0,S_1). If we let $F(S_1) = e^{-r\Delta t}V_1(S_1)$, we can write

$$V_0(S_0) = \int F(S_1)g_1(S_0,S_1)dS_1$$

If we assume that we can switch the order of the integration and differentiation, which we can under certain assumptions, we can calculate the delta as follows

$$\begin{aligned}
\Delta(S_0) &= \frac{\partial V}{\partial S_0} \\
&= \int F(S_1)\frac{\partial g_1}{\partial S_0}dS_1 \\
&= \int F(S_1)\frac{\partial \log g_1}{\partial S_0}g_1 dS_1 \\
&= \mathbb{E}\left[F(S_1)\frac{\partial \log g_1}{\partial S_0}\right]
\end{aligned} \tag{2.9}$$

For the gamma we have

$$\begin{aligned}
\Gamma(S_0) &= \int F(S_1)\left[\frac{\partial^2 \log g_1}{\partial S_0 2}g_1 + \frac{\partial \log g_1}{\partial S_0}\frac{\partial g_1}{\partial S_0}\right]dS_1 \\
&= \int F(S_1)\left[\frac{\partial^2 \log g_1}{\partial S_0^2} + \left(\frac{\partial \log g_1}{\partial S_0}\right)^2\right]g_1(S_0,S_1)dS_1 \\
&= \mathbb{E}\left[F(S_1)\left[\frac{\partial^2 \log g_1}{\partial S_0^2} + \left(\frac{\partial \log g_1}{\partial S_0}\right)^2\right]\right]
\end{aligned} \tag{2.10}$$

Now, by standard theory of log normal distributions, we have

$$g_1(S_0,S_1) = \frac{1}{\sqrt{2\pi\sigma^2\Delta T}}\frac{1}{S_1}\exp\left(-\left(\log S_1 - \log S_0 - \left(r - \frac{\sigma^2}{2}\right)\Delta T\right)^2/2\sigma^2\Delta T\right)$$

So

$$\log g_1(S_0, S_1) = C - \frac{\left(\log S_1 - \log S_0 - (r - \frac{\sigma^2}{2})\Delta T\right)^2}{2\sigma^2 \Delta T}$$

where **C** is some constant which does not depend on S_0. Differentiating, we get

$$\frac{\partial \log g_1(S_0, S_1)}{\partial S_0} = \frac{1}{S_0} \frac{\left(\log S_1 - \log S_0 - (r - \frac{\sigma^2}{2})\Delta T\right)}{\sigma^2 \Delta T}$$

In order to use our expression for the gamma, we differentiate again

$$\begin{aligned} \frac{\partial^2 \log g_1(S_0, S_1)}{\partial S_0} &= -\frac{1}{S_0^2} \frac{\left(\log S_1 - \log S_0 - \left(r - \frac{\sigma^2}{2}\right)\Delta T\right)}{\sigma^2 \Delta T} - \frac{1}{S_0^2} \frac{1}{\sigma^2 \Delta T} \\ &= -\frac{1}{S_0^2} \frac{1}{\sigma^2 \Delta T} \left(1 + \log S_1 - \log S_0 - \left(r - \frac{\sigma^2}{2}\right)\Delta T\right) \end{aligned}$$

Putting this all together, we have the likelihood ratio method for obtaining the price of the delta and gamma greeks.

2.2. Control Methods

2.2.1. Binomial tree pricing of Bermudan options

Here I will give a brief outline of the Binomial tree method for pricing Bermudan options. The ideas for this section come primarily from the original paper by Cox, Ross and Rubinstein [7] and from chapter 3 of Joshi's book [6].

The Binomial tree is used to model stock price evolution, and its parameters are chosen so that this evolution matches a Geometric Brownian motion with volatility σ and drift r. We first divide the time **T** over which we are modelling the stock's evolution into **N** smaller steps of length $\delta T = \frac{T}{N}$. After the n^{th} time-step $t_n = n\Delta T$, the stock-price S_n can move up to uS_n or down to dS_n with probabilities p and (1-p) respectively. Note that at each time-step n, the tree has $n + 1$ nodes, labelled $S_{n,1}, \ldots, S_{n,n+1}$.

Our first choice is to set $d = \frac{1}{u}$ so as to ensure that the tree recombines. With this, we choose $u = e^{\sigma\sqrt{\Delta T}}$ so as to match the volatility of the process. In order to match the risk-neutral behaviour, and have our expected stock price grow at the risk-free rate, we need to choose the probability of an up move to be

$$p = \frac{e^{r\Delta T} - d}{u - d}$$

These definitions are sufficient to define a binomial tree which simulates a Geometric Brownian motion stock price.

In order to price a European option $V_0(S_0)$ with a payoff function $f(S_T)$, we use the terminal condition

$$V_N(S_{N,i}) = f(S_{N,i}) \text{ for } i \in \{1, \ldots, n+1\}$$

We then proceed by backward induction from N to 0, and at each time-step $n-1$ use the relation

$$V_{n-1}(S_{n-1,i}) = e^{-r\Delta t}\mathbb{E}\left[V_n(S_n)|S_{n-1,i}\right]$$

where S_n is the value of the stock at time-step n. This is modelled in our binomial tree as

$$V_{n-1}(S_{n-1,i}) = e^{-r\Delta t}\left(pV_n(uS_{n-1,i}) + (1-p)V_n(dS_{n-1,i})\right)$$

With this procedure, we can price back to $V_0(S_0)$.

To price a Bermudan option with a Binomial tree, we make the simple modification that if $n-1$ is an exercise date, we first set

$$\hat{V}_{n-1}(S_{n,i}) = e^{-r\Delta t}\left(pV_n(uS_{n-1,i}) + (1-p)V_n(dS_{n-1,i})\right)$$

and then set the value of the option

$$V_{n-1}(S_{n,i}) = \max(\hat{V}_{n-1}(S_{n,i}), f(S_{n,i}))$$

Which allows us to find the price at time 0 of the Bermudan option.

Note that when we implement this method, we should not store the whole tree $S_{n,i}$, where $n \in (0,\ldots,N)$ and $i \in (1, n+1)$, as this would require $\frac{(N+1)(N+2)}{2}$ nodes to be stored. If we only hold onto one step of the tree at a time, and evolve it backwards, then we must store at most $N+1$ nodes. The disadvantage of this method is that if we want to price an option in 2 or more dimensions, the computation time increases exponentially. For this reason it is inferior to the Longstaff-Schwartz method for pricing multi-dimensional options.

2.2.2. Finite difference Theta-Scheme for Bermudan options

The ideas for this section largely derive from the course on Finite Difference techniques in Mathematical Finance given by Dr Christoph Reisinger of Oxford University [10]. I shall summarise the derivation of the theta-scheme and show how it can be applied to the pricing of Bermudan options.

To begin with, consider the ordinary Black-Scholes differential equation

$$\begin{gathered}\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \\ \text{where} \quad S \in (0,\infty)\ t \in (0,T) \\ \text{and} \quad V(S,T) = \max(K-S,0)\end{gathered}$$

In order to discretise this equation, we first set an upper bound on S, S_{max} say. We then divide the range of possible S into N intervals of length $\Delta S = S_{max}/N$ and the time into M equal timesteps of length $\Delta t = T/M$. With this, we use a finite central difference to approximate the derivatives in S:

$$\begin{aligned}\frac{\partial V}{\partial S}(S_n,t_n) &= \frac{V(S_{n+1},t_m) - V(S_{n-1},t_m)}{2\Delta S} + \mathbf{O}(\Delta S^2) \\ \frac{\partial^2 V}{\partial S^2}(S_n,t_m) &= \frac{V(S_{n+1},t_m) - 2V(S_n,t_m) + V(S_{n-1},t_m)}{\Delta S^2} + \mathbf{O}(\Delta S^2)\end{aligned}$$

For the time derivative, we choose between a backward difference:

$$\frac{\partial V}{\partial t}(S_n, t_m) = \frac{V(S_n, t_m) - V(S_n, t_{m-1})}{\Delta T} + \mathbf{O}(\Delta t)$$

and a forward difference:

$$\frac{\partial V}{\partial t}(S_n, t_m) = \frac{V(S_n, t_{m+1}) - V(S_n, t_m)}{\Delta T} + \mathbf{O}(\Delta t)$$

Using the backward time difference and a central difference in the asset direction gives the following discretisation

$$\frac{V_n^m - V_n^{m-1}}{\Delta t} + \frac{1}{2}\sigma^2 \Delta S^2 n^2 \frac{V_{n+1}^m - 2V_n^m + V_{n-1}^m}{\Delta S^2} + r\Delta S n \frac{V_{n+1}^m - V_{n-1}^m}{2\Delta S} - rV_n^m = 0$$

In this case we can solve for V_n^{m-1} explicitly, giving the Explicit Euler Scheme:

$$V_n^{m-1} = A_n^m V_{n-1}^m + B_n^m V_n^m + C_n^m V_{n+1}^m$$

where

$$\begin{aligned} A_n^m &= \frac{1}{2}n^2\sigma^2\Delta t - \frac{1}{2}nr\Delta t \\ B_n^m &= 1 - n^2\sigma^2\Delta T - r\Delta t \\ C_n^m &= \frac{1}{2}n^2\sigma^2\Delta t + \frac{1}{2}nr\Delta t \end{aligned}$$

The forward time difference and central difference in the asset direction gives an alternate discretisation

$$\frac{V_n^{m+1} - V_n^m}{\Delta t} + \frac{1}{2}\sigma^2 \Delta S^2 n^2 \frac{V_{n+1}^m - 2V_n^m + V_{n-1}^m}{\Delta S^2} + r\Delta S n \frac{V_{n+1}^m - V_{n-1}^m}{2\Delta S} - rV_n^m = 0$$

In this case, we cannot solve for V_n^m explicitly, but can write it implicitly. This gives the Implicit Euler Scheme:

$$a_n^m V_{n-1} m + b_n^m V_n m + c_n^m V_{n+1}^m = V_n^{m+1}$$

where

$$\begin{aligned} a_n^m &= -\frac{1}{2}n^2\sigma^2\Delta t + \frac{1}{2}nr\Delta t \\ b_n^m &= 1 + n^2\sigma^2\Delta T + r\Delta t \\ c_n^m &= -\frac{1}{2}n^2\sigma^2\Delta t - \frac{1}{2}nr\Delta t \end{aligned}$$

In order to use these schemes to price a european put option, we would use the discretise boundary condition

$$V_n^M = \max(K - n\Delta S, 0)$$

and solve backwards in time to V_n^0. In order to obtain a scheme with good convergence and stability properties, we can average the implicit and explicit schemes with a weighting parameter $\theta \in [0,1]$. This gives the θ-scheme

$$a_n^m V_{n-1}^m + b_n^m V_n^m + c_n^m V_{n+1}^m = A_n^m V_{n-1}^{m-1} + B_n^m V_n^{m-1} + C_n^m V_{n-1}^{m-1}$$

where

$$\begin{aligned} a_n^m &= -\frac{1}{2}\theta\Delta t\left(\sigma^2 n^2 - rn\right) \\ b_n^m &= 1 + \theta\Delta t\left(\sigma^2 n^2 + r\right) \\ c_n^m &= -\frac{1}{2}\theta\Delta t\left(\sigma^2 n^2 + rn\right) \end{aligned}$$

and

$$\begin{aligned} A_n^m &= \frac{1}{2}\left(1-\theta\right)\Delta t\left(\sigma^2 n^2 - rn\right) \\ B_n^m &= 1 - \left(1-\theta\right)\Delta t\left(\sigma^2 n^2 + r\right) \\ C_n^m &= \frac{1}{2}\left(1-\theta\right)\Delta t\left(\sigma^2 n^2 + rn\right) \end{aligned}$$

This can be rewritten in matrix form

$$\begin{pmatrix} b_0^m & c_0^m & & \cdots & 0 \\ a_1^m & b_1^m & c_1^m & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & a_{N-1}^m & b_{N-1}^m & c_{N-1}^m \\ 0 & \cdots & 0 & a_N^m & b_N^m \end{pmatrix} \begin{pmatrix} V_0^{m-1} \\ V_1^{m-1} \\ \vdots \\ V_{N-1}^{m-1} \\ V_N^{m-1} \end{pmatrix} =$$

$$\begin{pmatrix} B_0^m & C_0^m & & \cdots & 0 \\ A_1^m & B_1^m & C_1^m & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & A_{N-1}^m & B_{N-1}^m & C_{N-1}^m \\ 0 & \cdots & 0 & A_N^m & B_N^m \end{pmatrix} \begin{pmatrix} V_0^m \\ V_1^m \\ \vdots \\ V_{N-1}^m \\ V_N^m \end{pmatrix}$$

or

$$\mathbf{M}_1 \mathbf{V}^{m-1} = \mathbf{M}_2 \mathbf{V}^m$$

For further details on the reasons for implementing the θ-scheme, I refer the interested reader to Dr Reisinger's notes.

It is straightforward to adapt this backward method to the task of pricing Bermudan options with $\mathbf{L}$ exercise dates. For the Bermudan put, we first set

$$V_n^M = \max(K - S_n, 0)$$

We then solve backwards to each date m via the theta-scheme, obtaining an approximation $\hat{V}_n^m$. If m is an early-exericise date, we set

$$V_n^m = \max(\hat{V}_n^m, \max(K - S_n, 0))$$

otherwise

$$V_n^m = \hat{V}_n^m$$

This provides us with a complete method for pricing the Bermudan put via finite-differences.

As in the Binomial Tree method, it should be noted that in order to adapt this scheme to pricing a Bermudan in higher dimensions, the number of computations will grow exponentially with the dimensionality. This again points to the advantage of the Longstaff-Schwartz algorithm.

3. Application to Exotic Option

Since the principle advantage of Longstaff-Schwartz is the non-exponential growth of the number of calculations required as we increase the dimension of the option, it is a good idea to try applying it in this case.

3.1. Longstaff Schwartz in 2-dimensions

The argument presented in section 2.1.1 works for any Markov Process $\mathbf{X}$, and so it holds for a 2-dimensional process ($\mathbf{X}$,$\mathbf{Y}$), which is the case I considered. There are certain practical differences, however, which I shall briefly outline here.

I will assume that the processes $\mathbf{X}$ and $\mathbf{Y}$ have correlation ρ and are defined by the following pair of Stochastic Differential Equations,

$$\begin{aligned} dX_t &= rX_t dt + \sigma_x X_t dB_t^1 \\ dY_t &= rY_t dt + \sigma_y Y_t \left(\rho dB_t^1 + \sqrt{1-\rho^2} dB_t^2\right) \end{aligned}$$

where we assume B_t^1 and B_t^2 are independent Brownian Motions. If we define a new Brownian motion $\hat{B}_t$ such that it has a correlation ρ with B_t^1,

$$d\hat{B}_t = \rho dB_t^1 + \sqrt{1-\rho^2} dB_t^2$$

then we can rewrite our equation for $\mathbf{Y}$ as

$$dY_t = rY_t dt + \sigma_y T_t d\hat{B}_t$$

This pair of SDEs can be solved as

$$\begin{aligned} X_t &= X_0 \exp\left(\left(r - \frac{1}{2}\sigma_x^2\right)t + \sigma_x B_t^1\right) \\ Y_t &= Y_0 \exp\left(\left(r - \frac{1}{2}\sigma_y^2\right)t + \sigma_y \hat{B}_t\right) \end{aligned} \tag{3.1}$$

Another practical difference is that our basis functions are now 2-dimensional, and for this purpose I will choose a product of two of the weighted Laguerre functions used in the 1-dimensional case. I have not established their completeness or linear independence,

and this warrants further investigation. I will choose a set of $(\mathbf{R}+1)^2$ of these functions ϕ_{rs} where $r \in (0, R)$ and $s \in (0, R)$, and the approximation takes the form

$$\begin{aligned} \mathbb{E}\left[V(X_{i+1}, Y_{i+1})|X_i = x, Y_i = y\right] &\approx \sum_{r,s=0}^{R} \phi_{rs}(x,y)\beta_{i,rs} \\ &= C_{i+1}^{R}(x,y) \end{aligned}$$

Which is what the regression step will try to fit.

3.2. Pathwise greeks in 2-dimensions

I will here derive formulas for the greeks of the Longstaff-Schwartz approximation in 2-dimensions by the Pathwise sensitivity method. The calculations are quite similar to those presented in section 2.1.4, but will be required for the implementation.

In our example case, we use a type of put payoff based on both of the driving processes

$$f(x,y) = \max\left(K - \max\left(x,y\right), 0\right)$$

In order to use 2.3, we need to differentiate:

$$\begin{aligned} \frac{\partial f}{\partial x} &= -\mathbb{1}_{K>\max(x,y)}\mathbb{1}_{x>y} \\ \frac{\partial f}{\partial y} &= -\mathbb{1}_{K>\max(x,y)}\mathbb{1}_{y>x} \end{aligned} \tag{3.2}$$

Note that for our driving correlated GBM processes $\mathbf{X}$ and $\mathbf{Y}$, there are two deltas, two gammas and two vegas, corresponding to which of parameters we are changing. Given that $\mathbf{X}$ is solved by 3.1, we can write down the following formulae for the delta and vega,

$$\Delta_X(0) = \mathbb{E}\left[e^{-r\tau}\left(-\mathbb{1}_{K>\max(x,y)}\mathbb{1}_{x>y}\right)\exp\left((r - \frac{1}{2}\sigma_X^2)\tau + \sigma_X W_1(\tau)\right)\right] \tag{3.3}$$

Empirical results again suggest that we can apply our reasoning to the vega, which produces the following equation:

$$VegaX(0) = \mathbb{E}\left[e^{-r\tau}\left(-\mathbb{1}_{K>\max(x,y)}\mathbb{1}_{x>y}\right)\left(\sigma_X W_1(\tau) - \sigma_X\tau\right)\exp\left((r - \frac{1}{2}\sigma_X^2)\tau + \sigma_X W_1(\tau)\right)\right] \tag{3.4}$$

The equations for the $\Delta_Y(0)$ and $VegaY$ are almost identical. For the ρ, there is again the slight modification that the discount factor has to be differentiated as well as the integrand. Also, both the derivative terms in equation 3.2 contribute to our result. We get

$$\begin{aligned}\rho_0(X_0, Y_0) &= \mathbb{E}\left[\left(-\tau e^{-r\tau}\right)\max\left(K - \max(X_\tau, Y_\tau)\right)\right] \\ &+ \mathbb{E}\left[-e^{-r\tau}\left(\mathbb{1}_{K>\max(x,y)}\mathbb{1}_{x>y}e^{\left(r-\frac{1}{2}\sigma_X^2\right)\tau+\sigma_X W_1(\tau)}\right.\right. \\ &+ \left.\left.\mathbb{1}_{K>\max(x,y)}\mathbb{1}_{y>x}e^{\left(r-\frac{1}{2}\sigma_Y^2\right)\tau+\sigma_Y W_2(\tau)}\right)\tau\right]\end{aligned} \tag{3.5}$$

With these equations, we can calculate the values of the greeks of the in the 2-dimensional case.

Note that we can implement an alternative method for calculating greeks by simply varying our parameters in the regression step and evaluation step by a small quantity ϵ, and then using

$$\Delta_X(X_0) = \frac{V_0(X_0) - V_0(X_0 - \epsilon)}{\epsilon} \tag{3.6}$$

or (for the rho and $vega$)

$$Vega_X(X_0, \sigma_X) = \frac{V_0(X_0, \sigma) - V_0(X_0, \sigma - \epsilon)}{\epsilon} \tag{3.7}$$

$$Rho_X(X_0, r) = \frac{V_0(X_0, r) - V_0(X_0, r - \epsilon)}{\epsilon} \tag{3.8}$$

4. Implementation of Techniques

Here I will describe some of the choices made in the implementation of my techniques in Matlab. I will also note some of the problems I encountered, and some possible improvements.

4.1. Implementation issues in 1-dimensional case

Since we are working with standard Geometric Brownian motion paths, I was able to use the explicit formula when generating the NxM paths required. The code for doing so is included in A.2.

My first implemention was a program to perform the regression step as outlined in section 2.1.1. I initially used a generalised Laguerre polynomial function downloaded from the internet to calculate the weighted laguerre functions, but this was very slow. I decided to simply write the polynomials out explicitly, and this allowed the regression code to run almost 100 times faster. Changing from *for* loops to vectorised code allowed the program to about 5 times faster again. I also tried implementing a look-up table, which pre-evaluated the weighed-polynomials for about 100,000 possible inputs and stored these values in an array. It turned out that this method gave only a relatively small improvement to the runtime (cutting it by about 30%). It also added some uncertainty to the results, as I was unsure how good the approximation was. As will be seen in the results section 5.1, 1000,000 paths was found to produce a reasonable level of convergence, in terms of having a small MCE (explained below). With regards the discounting, I made the choice to include the discount factor in the Beta coefficients themselves, though this is arbitrary. Although there is intuitive reason to use the payoff as one of the basis functions, extensive testing showed that it made no large improvement to the convergence properties of my program. The code is contained in A.1.

There is an large source of error in the inversion step, as the matrices are close to singular. This produced large errors in the estimates of the at-the-money strikes (even as the number of paths was brought to 1000,000). I tried using sparse matrix techniques, but there was no noticeable improvement. There was also no noticeable difference if I used the *inv* function in Matlab or the backslash inversion operator.

For the evaluation step, I implemented the forward method given in section 2.1.1. This gave a slight increase in the efficiency, but also made the code easier to understand. I chose to have the evaluation program generate a set of random paths determined by an inputted set of N(0,1) random variables. Since the same set of paths will be always be generated by these random numbers, this allowed me to find the greeks by likelihood ratio and pathwise sensitivity for the same set of paths. It also removed variability

in the approximated value for any particular set of inputs (Spot, Strike, risk-free rate, volatility, expiry), thus making it possible to estimate the greeks by finite differences. The code is contained in A.2.

For the reported error, I gave the variance of the payoffs and the variance of the Monte Carlo estimates

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}\left(V_n(0) - \frac{1}{N}\sum_{i=0}^{N} V_i(0)\right)^2$$

$$MCE = \sqrt{\frac{\sigma^2}{N}} \tag{4.1}$$

The σ converged to a positive value, while the MCE converged to zero, as I will show.

To implement the pathwise greeks, I utilised the results from section 2.1.4. In all cases, I used a set of paths to simulate the usual asset price evolution, and applied the stopping rule obtained in the regression step. For each greek, I also generated the parallel processes defined in equations 2.5,2.6 and 2.8. I then combined this with the derivative of the put payoff given in 2.4.

For the delta, this procedure was easy, since the parallel process given in 2.5 could be obtained by simple division:

$$\frac{\partial X}{\partial S_0} = \frac{X}{S_0}$$

For the Vega, this was not quite so straightforward. Because the parallel process given in 2.6 depends on the random increments of $W(t)$ (the driving brownian motion) up to time t, I had to generate generate both processes (one to define the stopping rule, and the other to find the value of the vega) within the function for evaluating the vega. I still used the same random numbers to generate the X process and the $\frac{\partial X}{\partial \sigma}$ process, so this did not cause difficulty.

The code for evaluating the Rho is very similar to that used to evaluate the delta, but is included for completeness. The main difference is that there is now an extra term according to 2.7. The code for all these greeks is contained in section A.3.

In the case of the Likelihood-Ratio method, I implemented code to find estimates of the delta and the gamma, and chose to do these estimations in parallel as it made analysing the results slightly simpler. I used a forward method along a set of N paths of X, and implemented the stopping rule defined by an inputted set of βs. As can be seen from equations 2.9 and 2.10, I only need the value of the derivative processes at the first time-step (first exercise date). This meant that the likelihood ratio method required less memory for the parallel procesess than the pathwise methods did, as they needed the full paths to find their estimates. However, I found that the likelihood ratio method required many more paths than the pathwise method did to obtain the same order of convergence, as will be seen in the results section. The code is given in section A.4.

4.2. Implementation in 2-dimensional case

The implementation in 2-dimensions was similar to that in 1-dimensions. The code for the regression step is included in appendix A.5.1, and can be seen to follow the same structure as the 1d case. The main difference is that there are now two correlated paths being simulated. Also note that the Regression function now takes in a set of $N(0,1)$ random variables from which to generate the paths, which was necessary in order to implement a stable parameter bumping method for approximating the greeks, as in equations 3.6,3.7 and 3.8. The code for the Evaluation is contained in appendix A.5.2, while that for the pathwise greeks are in appendix A.5.3.

5. Numerical Results

In this chapter I will present some of the results my programs produced, and show how they compared with the control methods.

5.1. Convergence of Algorithms

I chose an in-the-money example for the Bermudan put option

- S_0=11 (spot)
- K=15 (strike)
- M=10 (exercise times)
- σ=0.4 (volatility)
- r=0.05 (risk free rate)
- T=1 (expiry time)

For these values, I ran my programs for a number of test functions (R+1), where R=3,5,6, and for a number of paths **N**= $10^2, 10^3, 10^4, 10^5, 10^6$. The results are contained in table 5.1, where I also report the MCE (standard deviation of the estimator) defined in equation 4.1. The reported MCEs (standard deviations) appear to be reasonable, as the values reported for fewer paths are within $3MCE$s of the next, more accurate, value. The values reported at a particular **N** but different **R** are within $3MCE$s of each other, so our estimates are reasonably consistent. Surprisingly, the MCE for different **R** but the same **N** are very close to each other, suggesting that the algorithm converges at a rate that is independent of the number of basis functions. This would warrant further theoretical and empirical investigation, as it seems to contradict Glasserman and Yu [1]. I also plotted the results with error bounds for R=3 in figure 5.1, and for R=6 in 5.2, where we can see the convergence of the method.

The results obtained by the control methods (the Binomial Tree and the Theta-Scheme) are presented in table 5.2. The binomial tree had 2000 time-steps, while the finite difference scheme consisted of 750 asset-steps and 1000 time-steps. The greeks were obtained by a simple finite-differencing. As can be seen, our results are not with $3MCE$ of the control values, but are not completely off either. The Longstaff-Schwartz values are generally below those given by the control methods, however, which is what you would expect since it is a sub-optimal method.

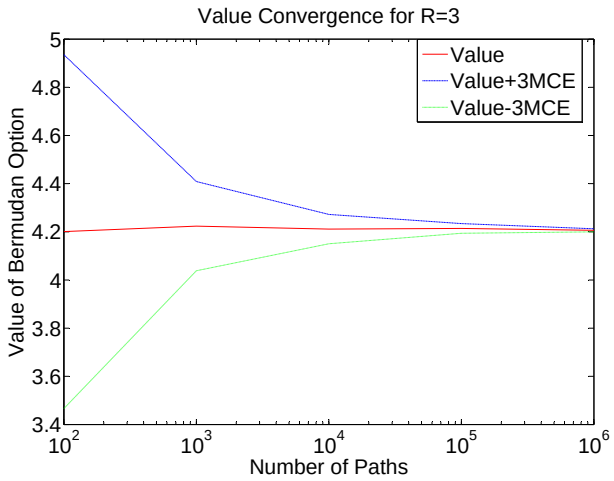


Figure 5.1.: Convergence of Longstaff Schwartz algorithm for R=3

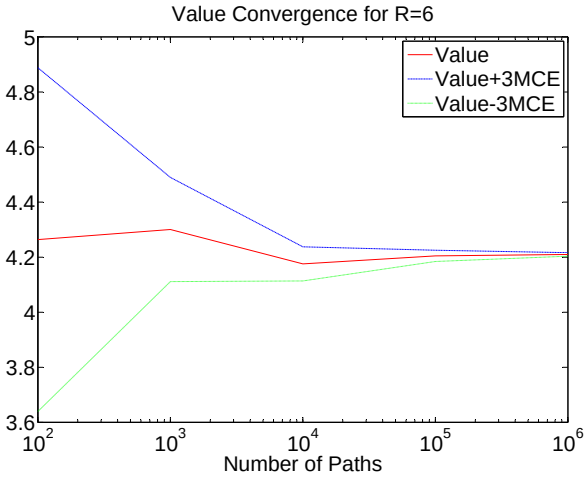


Figure 5.2.: Convergence of Longstaff Schwartz algorithm for R=6

Table 5.1.: Convergence Results for value by Longstaff-Schwartz

N	Value for R=3	MCE	Value for R=5	MCE	Value for R=6	MCE
10^2	4.200888	0.244865	3.900634	0.264411	4.2639	0.208066
10^3	4.223554	0.061730	4.377371	0.071921	4.300519	0.063081
10^4	4.211328	0.020325	4.194128	0.020364	4.175692	0.020660
10^5	4.214111	0.006696	4.212556	0.006803	4.204823	0.006813
10^6	4.206238	0.002104	4.203127	0.002155	4.209995	0.002146

Table 5.2.: The Results for the Value and Greeks by the Control Methods

Value of put by Binomial Tree	4.3068
Value of put by FD theta-scheme	4.2582
Value of delta by Binomial Tree	-0.7448
Value of delta by FD theta-scheme	-0.7648
Value of gamma by Binomial Tree	0.0724
Value of gamma by FD theta-scheme	0.0971
Value of vega by Binomial Tree	3.2534
Value of rho by Binomial Tree	-5.6860

For the upper bound, I made the further choice that P, the number of mini-paths used to approximate the martingale, be equal to 100. I also only calculated for R=3, as unpresented empirical tests showed this to be sufficient. The results obtained are presented in table 5.3. Although the values are consistent with each other (in that they are within $3MCE$ of the more accurate estimates), they are not consistent with the values given by the Longstaff-Schwartz program, nor are they consistent with those given by the Control methods. They are higher the Longstaff-Schwartz values, which means they are consistent with being an upper bound, though they are not higher than the value given by the binomial tree method.

Table 5.3.: The results for the Upper Bound with R=3, P=100

N	Upper Bound Value	MCE
100	4.281906	0.034112
1000	4.303028	0.012913
10000	4.287583	0.003970
100000	4.288507	0.001266

The results for the pathwise delta are presented in table 5.4, and a plot of the convergence for $R = 3$ is presented in figure 5.3. They are once again within $3MCE$ of the next more accurate estimate, and the MCE for each value of $\mathbf{R}$ seems to be of the same order. There is also a trend for the MCE to decrease by a factor of 3, every time we increase the number of paths by a factor of 10. This trend is present also in the MCEs of the estimates of the Put values and of the Upper Bounds. These estimates of

the Delta are also reasonably close to the two control values given in table 5.2, though not within $3MCE$, as in the previous cases.

Table 5.4.: Convergence Results for the Pathwise Delta

N	Delta for R=3	MCE	Delta for R=5	MCE	Delta for R=6	MCE
10^2	-0.766615	0.034767	-0.689335	0.036451	-0.810259	0.034343
10^3	-0.767261	0.010468	-0.78708	0.009818	-0.730166	0.011634
10^4	-0.778554	0.003158	-0.775909	0.003168	-0.758301	0.003329
10^5	-0.766503	0.001024	-0.757258	0.001037	-0.765073	0.001021
10^6	-0.769279	0.000321	-0.757149	0.000328	-0.761438	0.000325

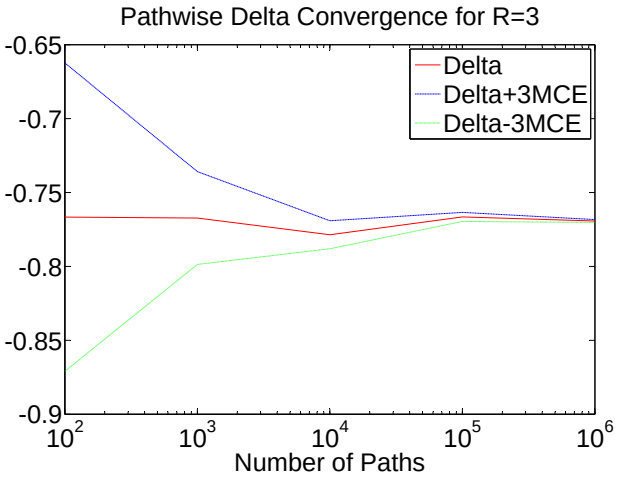


Figure 5.3.: Convergence of Pathwise Sensitivity applied to $1d$ put for R=3

The results for the Likelihood Ratio estimates of the Delta are presented in table 5.5, and plots of the convergence of the Delta and Gamma estimates for $R = 3$ are presented in figure 5.4. The most striking result is that the MCE is much larger for these estimates than in the Pathwise case. It seems that the MCE for the Likelihood ratio method is approximately 20 times larger than that obtained from the same number of paths and basis functions by the Pathwise method. Noting that the MCE is once again decreasing by a factor of 3 each time we increase the number of paths by a factor 10, we can say that we need approximately $10^{\log_3 20} = 533$ times as many paths to obtain the same accuracy.

The convergence properties of the Likelihood-Ratio estimates of the Gamma are similar to those of the Delta case. For this reason, I only include the most accurate (those for $N = 10^6$ paths) results in table 5.6. They are again seen to be reasonably close, though not within $3MCE$, of the control values.

As seen in section 2.1.4, the Pathwise method is not theoretically verified in the case of estimating the *Rho* or the $Vega$. However, the empirical results which I present in

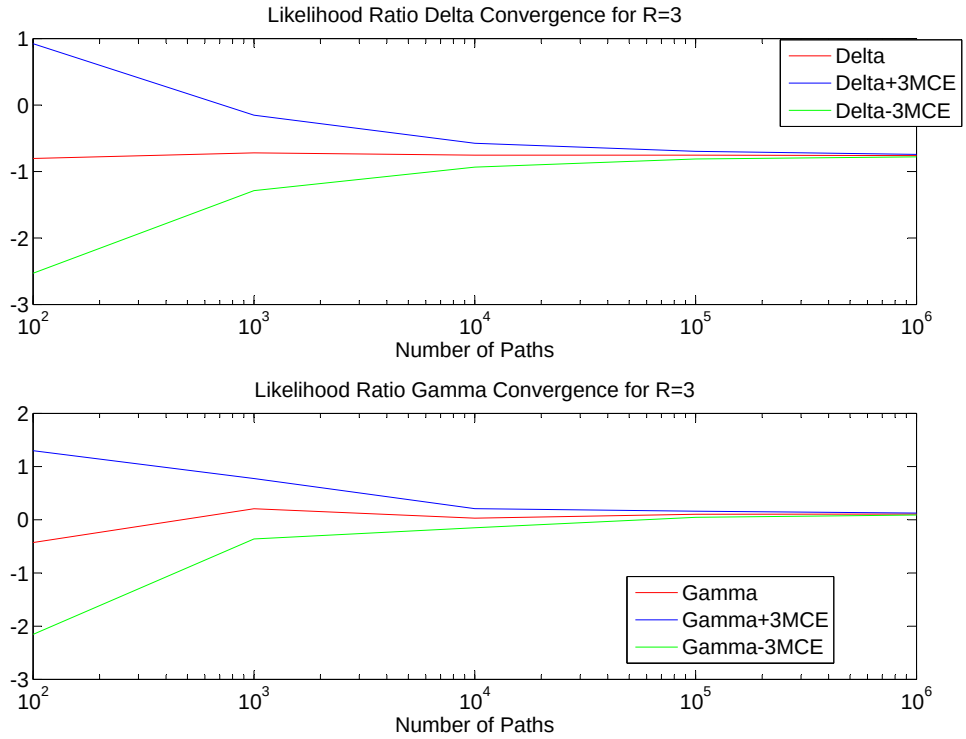


Figure 5.4.: Convergence of Likelihood Ratio method applied to $1d$ put for R=3

Table 5.5.: Convergence Results for the Likelihood Ratio Delta

N	Delta for R=3	MCE	Delta for R=5	MCE	Delta for R=6	MCE
10^2	-0.803436	0.575370	-0.494308	0.543516	-0.694370	0.538568
10^3	-0.719689	0.189256	-0.531730	0.179957	-0.675315	0.185094
10^4	-0.754190	0.059946	-0.773072	0.060796	-0.693407	0.059045
10^5	-0.753492	0.019068	-0.764888	0.019112	-0.783757	0.019225
10^6	-0.760409	0.006044	-0.764224	0.006050	-0.762592	0.006051

Table 5.6.: Convergence Results for the Likelihood Ratio Gamma for 10^6 paths

Gamma at R=3	MCE	*Gamma* at R=5	MCE	*Gamma* at R=6	MCE
0.107497	0.006044	0.104108	0.006050	0.102467	0.006051

tables 5.7 and 5.8 suggest the convergence of the method. It can also be seen that these results are fairly close to the control values, giving strong evidence that the Pathwise method is correct. This warrants further theoretical investigations, as it appears to be a fast converging method (MCE again decreasing by a factor of 3 for each ten-fold increase in paths) for estimating greeks.

Table 5.7.: Convergence Results for the Pathwise Vega

N	Vega for R=3	MCE	Vega for R=5	MCE	Vega for R=6	MCE
10^2	2.768602	0.360631	3.215388	0.295047	2.000074	0.348818
10^3	2.675375	0.078972	2.865397	0.086071	3.113368	0.121532
10^4	2.787354	0.023958	2.780740	0.023837	2.907690	0.027082
10^5	2.909097	0.008127	2.995093	0.008292	2.946135	0.008107
10^6	2.894445	0.002537	2.995693	0.002606	2.972047	0.002599

5.2. Hedging Error

Finding an estimate for the Hedging Error is a different way to measure how accurately the Longstaff-Schwartz algorithm reproduces the 'fair' value of a Bermudan Option. The basic idea is to assume that we have written a Bermudan put option on a stock S with $\mathbf{M}$ exercise dates, and to use our programs to find out how much of the stock and how much cash we should hold at a particular time to replicate the option.

From basic theory, we know that we should hold Δ_i of the stock at time-step i, where Δ_i is the Delta of the Bermudan option, and an amount of cash B_i such that (ideally)

$$V_i(S_i) = B_i + \Delta_i(S_i)$$

where $V_i(S_i)$ is the value of the Bermudan option. If we are trying to replicate the option, we should adjust this holding in a self-financing way as the stock price S_i evolves over time.

Table 5.8.: Convergence Results for the Pathwise Rho

N	Rho for R=3	MCE	Rho for R=5	MCE	Rho for R=6	MCE
10^2	-6.402235	0.363689	-5.738520	0.429306	-6.197949	0.505527
10^3	-4.817835	0.123382	-5.628809	0.136456	-7.689981	0.142553
10^4	-4.886199	0.037839	-4.718475	0.037952	-5.462408	0.039042
10^5	-5.230901	0.012382	-5.360896	0.012521	-5.231239	0.012321
10^6	-5.160281	0.003886	-5.326362	0.003971	-5.300562	0.003968

For us, this means that we start by calculating the value and the Delta of the Bermudan option at time 0, which are $V_0(S_0)$ and $\Delta_0(S_0)$ respectively. We then find an amount of cash B_0 satisfying

$$B_0 = V_0(S_0) - \Delta_0(S_0)S_0$$

At each reset time i, we find the new $\Delta_i(S_i)$ of stock that we are supposed to hold, and adjust our portfolio by either borrowing or lending cash. In order to see how much cash B_i we end up with, note that our portfolio evolves over one time-step as follows

$$B_{i-1} + \Delta_{i-1}(S_{i-1})S_{i-1} \Rightarrow B_{i-1}e^{r\Delta t} + \Delta_{i-1}(S_{i-1})S_i$$

where r is the risk-free rate. We then require the total value of the portfolio to remain the same

$$B_{i-1}e^{r\Delta t} + \Delta_{i-1}(S_{i-1})S_i = B_i + \Delta_i(S_i)S_i$$

so we can set

$$B_i = B_{i-1}e^{r\Delta t} + (\Delta_{i-1} - \Delta_i)S_i$$

which defines our replicating strategy.

If we are replicating a Bermudan put, it means that the holder has the right to sell a stock to us for some strike K at each exercise time. At this time we receive $S_i - K$, and can liquidate our holding Δ_i in stock. We should be left with (in cash)

$$HedgingError_i = S_i - K + B_i + \Delta_i S_i$$

which is the definition of our Hedging Error. If our strategy replicated the value of the option perfectly, Δ should be -1 at the optimal exercise time, B should be K, and our Hedging error should be 0.

In order to find an estimate of the Hedging error, I simulated P paths and performed the above replication strategy for a Bermudan put, using our Longstaff Schwartz algorithm with N paths and $R+1$ basis functions. For simplicity, I assumed that the exercise dates and the reset dates were the same. I also assumed that the holder of the option (the counterparty) was also using a Longstaff Schwartz regression with N paths and $R+1$ basis functions to determine his stopping rule. For this, I recorded the Hedging Error realised along each of the P paths, discounted back to 0, and averaged to find a Monte-Carlo estimate. The implementation for this is provided in A.6.

I did this with $P = 10000$, $N = 200$ and 4 basis functions, on a Bermudan put with 5 exercise dates and the following parameters

- S_0=11
- K=15
- σ=0.4
- r=0.05
- T=1

I obtained the following results

- Average Hedging Error=0.1127
- MCE=0.00678

I found that my set of discounted simulated Hedging Errors had several outliers, which I removed in order to clarify the results. The histogram of this refined set is given in figure 5.5. From this plot, and the results presented above, we can see that the person doing the hedging can expect to make a profit by this strategy. However, it is important to note the assumption that the holder of the option is also using Longstaff-Schwartz to define his stopping rule, so it does not mean that we can expect to make a profit with this strategy in the real-world.

The shape of this histogram suggests normality of the data (once the outliers have been removed), so I implemented a QQplot for the complete set, and obtained the results in figure 5.6. This shows that although a large set of the discounted Hedging Errors are normal (along the line), the distribution has very fat tails. Although the inputted process to the Hedging Error procedure (Geometric Brownian motion) was normal, the procedure is a non-linear transformation of this input, so it is not unreasonable that the output is non-normal. However, I was unable to find a satisfactory explanation for the fat tails.

5.3. Results for 2 dimensional option

I here present some of the results obtained by the Longstaff-Schwartz method applied to a Bermudan option with **M** exercise dates, and the put-like payoff

$$f(X_t, Y_t) = \max(K - \max(X_t, Y_t))$$

The parameters that I am using are

- X_0=11 (initial X price)
- Y_0=12 (initial Y price
- K=20 (strike)
- M=10 (exercise times)

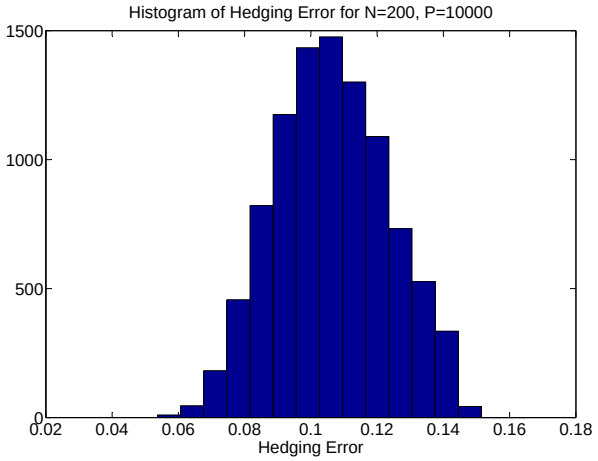


Figure 5.5.: Histogram of Discounted Hedging Errors with Outliers removed, for N=200, P=10000

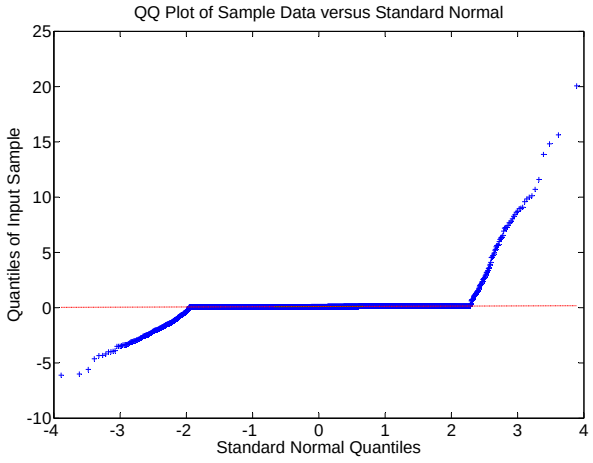


Figure 5.6.: QQplot for Discounted Hedging Errors for N=200, P=10000

- σ_x=0.4 (volatility in X)
- σ_y=0.3 (volatility in Y)
- ρ=0.2 (correlation)
- r=0.05 (risk free rate)
- T=1 (expiry time)

I ran my simulations for a number (the input to the program) $R = 1, 2, 3$ which translates to $L = (R+1)^2 = 4, 9, 16$ two dimensional basis-functions, and $N = 10^2, 10^3, 10^4, 10^5$ paths. The results for the convergence of the value are presented in 5.9, and a plot of the results for $L = 4$ are presented in figure 5.7. The important thing to notice is that the MCEs are not exponentially larger (if comparing for the same number of paths) than in the $1-d$ case, which demonstrates the power of the Longstaff-Schwartz method. This indicates that we can estimate a $2-d$ option with the same amount of accuracy as a 1d option by using only slightly more paths.

Table 5.9.: Convergence Results for Value of Basket-Put by Longstaff-Schwartz

N	Value for L=4	MCE	Value for L=9	MCE	Value for L=16	MCE
10^2	5.904278	0.314147	5.599359	0.315288	5.603622	0.329693
10^3	5.705902	0.104875	5.681589	0.102355	5.792139	0.103930
10^4	5.863186	0.032652	5.912238	0.032639	5.814291	0.032585
10^5	5.877196	0.010312	5.881997	0.010312	5.866350	0.010338

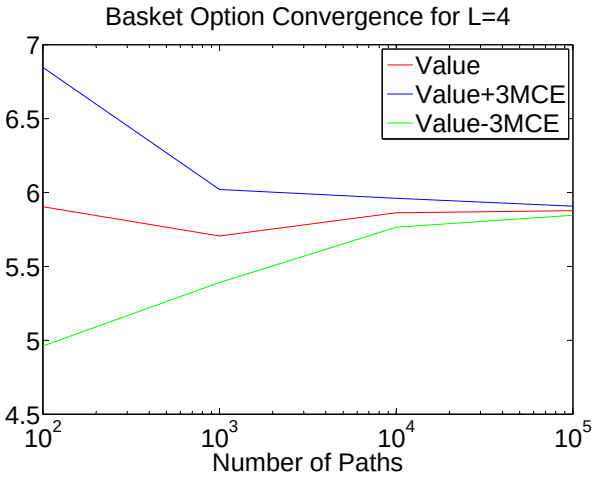


Figure 5.7.: Convergence of Longstaff-Schwartz algorithm in 2-dimensions

Note that, as in the one-dimensional case, the Monte-Carlo estimates are within $3MCE$s of the next more accurate estimate, which shows that our results are reasonably

consistent with each other. A trend that carries over from the $1-d$ is that the MCE decreases by a factor of 3 for every factor of 10 that we increase the number of paths by. A further interesting observation is that the MCEs are relatively independent of the number of basis functions L that we use.

The convergence results for the $Delta_X$s are presented in . Again we see that although the MCEs are slightly higher than in the one dimensional case, they are only higher by at most a factor of 2. This continues even when we increase the number of paths (i.e. The MCE is less than a factor of 2 larger than in the $1-d$ case no matter how many paths we use), which again demonstrates the power of the Longstaff-Schwartz algorithm in higher dimensions. The trend of an MCE decreasing by a factor of 3 continues to be present, and the results are consistent to within $3MCE$s of each other. Also carrying over from the other examples of Longstaff-Schwartz, the MCE does not vary noticeably with the number of basis functions used.

Table 5.10.: Convergence Results for $Delta_X$ of Basket-Put by Pathwise Sensitivity

N	$Delta_X$ for L=4	MCE	$Delta_X$ for L=9	MCE	$Delta_X$ for L=16	MCE
10^2	-0.417197	0.059925	-0.447792	0.061556	-0.322614	0.053313
10^3	-0.408543	0.018376	-0.396829	0.018279	-0.396561	0.018208
10^4	-0.418394	0.005853	-0.400318	0.005748	-0.393235	0.005789
10^5	-0.400273	0.001823	-0.400689	0.001827	-0.402097	0.001829

Table 5.11.: The Greeks by Parameter bumping in 2 dimensions, using ϵ=0.02

Greek	L=4	L=9	L=16
$Delta_X$	-0.40177	-0.40052	-0.39907
$Delta_Y$	-0.59507	-0.59643	-0.59774
$Vega_X$	-1.39118	-1.39389	-1.36003
$Vega_Y$	-0.68017	-0.71802	-0.70964
Rho	-16.53097	-16.51424	-16.54203

The results for $Delta_Y$ have the same good convergence behaviour as the $Delta_X$, and so only the most accurate (those for $N = 10^5$ are given. These are in table 5.12. The values for the different greeks obtained by finite-difference bumping are contained in table 5.11. We can see that our estimates for the Value, $Delta_X$ and $Delta_Y$ are often within $3MCE$ of these results, and so can be seen to very good approximations.

Table 5.12.: Results for $Delta_Y$ of Basket-Put by Pathwise Sensitivity

N	$Delta_Y$ for L=4	MCE	$Delta_Y$ for L=9	MCE	$Delta_Y$ for L=16	MCE
10^5	-0.595241	0.001716	-0.598424	0.001717	-0.596392	0.001719

In order to demonstrate the empirical accuracy of the Pathwise approximations of the Vega and Rho, the results are included in tables 5.13 and 5.14. As can be seen, there

Table 5.13.: Convergence Results for $Vega_X$ of Basket-Put by Pathwise Sensitivity

N	$Vega_X$ for L=4	MCE	$Vega_X$ for L=9	MCE	$Vega_X$ for L=16	MCE
10^2	-1.821712	0.516724	-2.117265	0.541848	-0.863449	0.402725
10^3	-1.332672	0.162768	-1.337647	0.161157	-1.293547	0.155001
10^4	-1.408243	0.050815	-1.247253	0.048649	-1.398844	0.050065
10^5	-1.279599	0.015651	-1.310095	0.015766	-1.324095	0.15777

is the very good convergence property of the MCE decreasing by a factor of 3 for each ten-fold increase in the number of paths. The results are also internally consistent, being within $3MCE$ of the more accurate estimates. By comparison with the control values in table 5.11, we see that the estimates are reasonably close, though not within $3MCE$. This is strong evidence that the Pathwise Method works in 2-dimensions, despite the fact that it remains theoretically unverified.

Table 5.14.: Convergence Results for *Rho* of Basket-Put by Pathwise Sensitivity

N	*Rho* for L=4	MCE	*Rho* for L=9	MCE	*Rho* for L=16	MCE
10^2	-16.514008	0.453863	-16.141074	0.508287	-16.141500	0.508391
10^3	-16.117484	0.162474	-16.234908	0.156715	-16.177474	0.160134
10^4	-16.321556	0.048535	-16.232289	0.050023	-16.210509	0.050177
10^5	-16.268508	0.015621	-16.311794	0.015408	-16.288142	0.015518

6. Conclusions and Final Remarks

In this thesis I have developed and implemented a method for pricing the Greeks of Bermudan options through an adaptation of Likelihood ratio and Pathwise sensitiv ity methods, applied to the Longstaff-Schwartz algorithm. It was found, in chapter 5, that these methods had good convergence properties, and produced results that were close to the control values. I also found that the Likelihood ratio method required considerably larger numbers of paths to obtain the same level of accuracy as the Pathwise sensitivity method.

Thanks to ideas suggested by Gyurko [3], I developed a proof that the Pathwise sensitivity and Likelihood ratio methods were valid in the case of the delta and gamma, and this is provided in sections 2.1.4 and 2.1.5. Although I was unable to prove that the Pathwise sensitivity (or Likelihood ratio) method was valid in the case of rho or vega, I found strong empirical evidence that it was, which suggests a path for future theoretical work.

Furthermore, I was able to use my estimates of the Delta to implement a program to simulate the Hedging Error, which I outlined in section 5.2. In my implementation, I assumed that the holder of the option (a simple Bermudan put) was also using the Longstaff-Schwartz algorithm to determine his stopping time. My results indicated that under this assumption, the expected Hedging Error was slightly positive, meaning that we could expect to make a profit if we were hedging by this method. I also investigated the distribution of the Hedging Error, and found that it was approximately normal, but with fat tails which I was unable to explain.

In addition to these main results, I was able to demonstrate the convergence properties of the Longstaff-Schwartz method proved in Clement et al [4], and suggested by Longstaff and Schwartz [2]. I found good convergence of the estimates across various numbers of basis functions, so long as sufficiently many paths were used. I also implemented a version of the High-Biased estimate proposed by Rogers [8], Glasserman [5], and Haugh and Kogan [9], and showed that it was indeed high-biased (compared with the low-biased estimate), and that it had good convergence properties.

By implementing my techniques for a non-trivial 2-dimensional Bermudan option, I demonstrated the main advantage of Longstaff Schwartz that the number of extra paths required to obtain a certain level of accuracy does not increase exponentially with the dimensionality of the problem. I also showed that the methods for pricing Greeks by the Pathwise sensitivity and Likelihood ratio method continued to work very effectively, even for the unverified case of the rho and vega.

A. The Matlab Code

A.1. Regression code

```
% LaguerreExplicit.m %%%%%%%%%%%%%%

function value=LaguerreExplicit(R,x)
% Generates the (weighted) laguerre polynomial value
    if R==0
        value=1;
    elseif R==1
        value=-x+1;
    elseif R==2
        value=0.5*(x.^2-4*x+2);
    elseif R==3
        value=(1/6)*(-x.^3+9*x.^2-18*x+6);
    elseif R==4
        value=(1/24)*(x.^4-16*x.^3+72*x.^2-96*x+24);
    elseif R==5
        value=(1/120)*(-x.^5+25*x.^4-200*x.^3+600*x.^2-600*x+120);
    elseif R==6
        value=(1/720)*(x.^6-36*x.^5+450*x.^4-2400*x.^3
         +5400*x.^2-4320*x+720);
    else
        disp('Error, R is out of range');
        value=0;
    end
    value=exp(-0.5*x).*value; % weighting used in Longstaff-Schwartz
end

% LSBetaByRegression.m %%%%%%%%%%%%%%

function Beta=LSBetaByRegression(R,N,M,S0,K,sigma,r,T,W)
% Generate the Betas used in the Longstaff-Schwartz
%  algorithm by regression
% Takes in a set of random numbers from which it generates paths
% Weighted Laguerre polynomials are used as the basis functions.
```

```
%R Number of basis functions
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)

payoff = @(x) max(K-x,0); % Put payoff

% Phase 1 : Generate N paths and approximate the coefficients Beta
S=[];X=[];
X=GenerateTheGBMPaths(N,M,S0,sigma,r,T,W);
% Now step backwards from T=1, evaluating the beta at each time
V=payoff(X(M,:));
for y=0:M-1 % might need to fix these indices
    m=M-y;
    for i=0:R
        for s=0:R
            n=[1:N];
            B_psi_psi(i+1,s+1)=
             sum(LaguerreExplicit(i,X(m,n))
             .*LaguerreExplicit(s,X(m,n)));
            B_psi_psi(i+1,s+1)=B_psi_psi(i+1,s+1)/N;
        end
    end
    for i=0:R
        n=[1:N];
        B_V_psi(i+1)=sum(V(n).*LaguerreExplicit(i,X(m,n)));
        B_V_psi(i+1)=B_V_psi(i+1)/N;
    end
    Beta(m,:)=inv(B_psi_psi)*B_V_psi'; % These are the coefficients
    % Now update the value of the option along these paths
    % Include the single-step discounting in the betas
    Beta(m,:)=Beta(m,:)*exp(-r*T/M);
    for n=1:N
        ContinuationVal=0;
        for i=0:R
            ContinuationVal=ContinuationVal
            +LaguerreExplicit(i,X(m,n))*Beta(m,i+1);
        end
        if ContinuationVal<payoff(X(m,n))
            V(n)=payoff(X(m,n));
        else
            V(n)=exp(-r*(T/M))*V(n); % Use Longstaff-Schwartz
        end
    end
```

```
end
end
```

A.2. Code for Evaluation of Price in 1-dimension

```
% GenerateTheGBMPaths.m %%%%%%%%%%%%%%

function S=GenerateTheGBMPaths(N,M,S0,sigma,r,T,W)
% Generates a set of NxM Geometric Brownian
% paths for a set of random numbers
    deltaT=T/M;
    S=zeros(M,N);
    S(1,:)=S0*ones(1,N);
    for m=1:M-1
        S(m+1,:)=
    S(m,:).*exp((r-0.5*sigma*sigma)
      *deltaT+sigma*W(m,:)*sqrt(deltaT));
    end
end

% LSAmericanPutVal.m  %%%%%%%%%%%%%

function [value variance MC_error]
=LSAmericanPutVal(R,N,M,S0,K,sigma,r,T,Beta,W)
% Uses Longstaff-Schwartz algortihm as outlined in Glasserman's book
% Generate a set of paths, and use the Betas (presumable obtained from
% regression) to evaluate the price of an American put option.
% Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions
%N Number of paths used in evaluation
%M Number of exercise dates (approximate American with Bermudan)

% S0 is initial asset value
% K is strike of the put
% sigma is volatility
% r is risk free rate
% T is expiry time

% W is a set of N(0,1) random numbers to use, this benefits comparisons.

payoff = @(x) max(K-x,0); % Put payoff
```

```
% Phase 2 : Use these Beta values to price the
% bermudan option along a new set of N2 simulated paths.
S=[];X=[];
X=GenerateTheGBMPaths(N,M,S0,sigma,r,T,W);
V=exp(-r*T)*payoff(X(M,:)); % Default value is no-exercise till expiry
for n=1:N
    for m=1:M
        ContinuationVal=0;
        for i=0:R
            ContinuationVal=ContinuationVal
            +LaguerreExplicit(i,X(m,n))*Beta(m,i+1);
        end
        if ContinuationVal<payoff(X(m,n)) || m==M
            V(n)=exp(-r*(T*m/M))*payoff(X(m,n));
            break;
        end
    end
end
value=sum(V)/N;

variance=sum((V-value).^2)/N;
MC_error=sqrt(variance/N);
end
```

A.3. Code for Pathwise Greeks in 1-dimension

```
% LSPathwiseDelta.m %%%%%%%%%%%%%%

function [delta_value delta_variance delta_MCE]
=LSPathwiseDelta(R,N,M,S0,K,sigma,r,T,Beta,W)
% Uses Longstaff-Schwartz algortihm as outlined by in
%  Glasserman's book.
% Calculates the delta based on pathwise sensitivity method
% Also returns the variance and a Monte Carlo error estimate
% Uses a set of Betas obtained (presumably) by regression
% Uses a set of random numbers W to generate the paths
% Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)
```

```
%L Number of time-steps between exercise dates

%r discount factor

payoff = @(x) max(K-x,0); % Put payoff

% Phase 2 : Use these Beta values to price
% the american option along a new set of simulated paths.
X=GenerateTheGBMPaths(N,M,S0,sigma,r,T,W);
dXdS0=X/S0;
V=exp(-r*T)*payoff(X(M,:)); % Default value is no-exercise till expiry
delta=exp(-r*T)*(K>X(M,:)).*dXdS0(M,:);
for n=1:N
    for m=1:M
        ContinuationVal=0;
        for i=0:R
            ContinuationVal=ContinuationVal
             +LaguerreExplicit(i,X(m,n))*Beta(m,i+1);
        end
        if ContinuationVal<payoff(X(m,n)) || m==M
            V(n)=exp(-r*(T*m/M))*payoff(X(m,n));
            delta(n)=-exp(-r*(T*m/M))*(K>X(m,n))*dXdS0(m,n);
            break;
        end
    end
end

delta_value=sum(delta(:))/N;
delta_variance=sum((delta(:)-delta_value).^2)/N; %
delta_MCE=sqrt(delta_variance/N);   % Variance of the estimator
end

% LSPathwiseVega.m %%%%%%%%%%%%%

function [vega_value vega_variance vega_MC_error]
=LSPathwiseVega(R,N,M,S0,K,sigma,r,T,Beta,W)
% Uses Longstaff-Schwartz algortihm as outlined in Glasserman's book
% Calculates the vega based on pathwise sensitivity method
% Also returns the variance and a Monte Carlo error estimate
% Uses a set of Betas obtained (presumably) by regression
% Uses a set of random numbers W to generate the paths
% Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions
```

```
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)

payoff = @(x) max(K-x,0); % Put payoff

% Phase 2 : Use the Beta values to price the american option along a new
% set of simulated paths.
deltaT=T/M;
X=zeros(M,N);
X(1,:)=S0*ones(1,N);

sumW=cumsum(W); % The sum of W along path N up to time M
dXdsigma=zeros(M,N);
for m=1:M-1
    % Generate the process used to define the stopping rule
    X(m+1,:)=X(m,:).*exp((r-0.5*sigma*sigma)
     *deltaT+sigma*W(m,:)*sqrt(deltaT));
    % Generate the process used to find the value of the vega
    dXdsigma(m+1,:)=(sumW(m,:)*sqrt(deltaT)
     -sigma*deltaT*m).*X(m+1,:);
end

V=exp(-r*T)*payoff(X(M,:)); % Default value is no-exercise till expiry
vega=-exp(-r*T)*(K>X(M,:)).*dXdsigma(M,:);
for n=1:N
    for m=1:M
        ContinuationVal=0;
        for i=0:R
            ContinuationVal=ContinuationVal
             +LaguerreExplicit(i,X(m,n))*Beta(m,i+1);
        end
        if ContinuationVal<payoff(X(m,n)) || m==M
            V(n)=exp(-r*(T*m/M))*payoff(X(m,n));
            vega(n)=-exp(-r*T*m/M)*(K>X(m,n)).*dXdsigma(m,n);
            break;
        end
    end
end

vega_value=sum(vega(:))/N;
vega_variance=sum((vega(:)-vega_value).^2)/N;
vega_MC_error=sqrt(vega_variance/N);
end
```

```
% LSPathwiseRho.m %%%%%%%%%%%%%%

function [rho_value rho_variance rho_MC_error]
=LSPathwiseRho(R,N,M,S0,K,sigma,r,T,Beta,W)
% Uses Longstaff-Schwartz algortihm as outlined in Glasserman's book
% Calculates the rho based on pathwise sensitivity method
% Also returns the variance and a Monte Carlo error estimate
% Uses a set of Betas obtained (presumably) by regression
% Uses a set of random numbers W to generate the paths
% Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)

%r discount factor

payoff = @(x) max(K-x,0); % Put payoff

% Phase 2 : Use these Beta values to price
% the american option along a new set of simulated paths.
deltaT=T/M;

X=GenerateTheGBMPaths(N,M,S0,sigma,r,T,W);
dXdr=zeros(M,N); % This sets the initial dXdr to zero
for m=1:M-1
    dXdr(m+1,:)=deltaT*m*X(m+1,:);
end

% This is the (preferred) forward method

V=exp(-r*T)*payoff(X(M,:)); % Default value is no-exercise till expiry
rho=-exp(-r*T)*(K>X(M,:)).*dXdr(M,:);
for n=1:N
    for m=1:M
        ContinuationVal=0;
        for i=0:R
            ContinuationVal=ContinuationVal
             +LaguerreExplicit(i,X(m,n))*Beta(m,i+1);
        end
        if ContinuationVal<payoff(X(m,n)) || m==M
            V(n)=exp(-r*(T*m/M))*payoff(X(m,n));
            rho(n)=exp(-r*(T*m/M))*(-(K>X(m,n))*dXdr(m,n)
             -deltaT*m*payoff(X(m,n)));
```

```
            break;
        end
    end
end


rho_value=sum(rho(:))/N;%-T*value;
rho_variance=sum((rho(:)-rho_value).^2)/N;
rho_MC_error=sqrt(rho_variance/N);
end
```

A.4. Code for Likelihood Greeks in 1-dimension

```
% LSLikelihoodGamma.m %%%%%%%%%%%%%%%%%%%

function [delta_value delta_variance
delta_MC_error gamma_value gamma_variance
 gamma_MC_error]
  =LSLikelihoodGamma(R,N,M,S0,K,sigma,r,T,Beta,W)
% Uses Longstaff-Schwartz algortihm as outlined in Glasserman's
% book. Uses the Likelihood-Ratio method to find esimates
% of the Delta and the Gamma for a Bermudan put.
% Also returns the variance and a Monte Carlo error estimate for these.
% Uses a set of Betas obtained (presumably) by regression
% Uses a set of random numbers W to generate the paths
% Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)

payoff = @(x) max(K-x,0); % Put payoff

dloggdX0 = @(x) (1/S0)*(log(x)-log(S0)
-(r-0.5*sigma*sigma)*(T/M))*(1/(sigma*sigma*(T/M)));
d2loggd2X0 = @(x) -(1/(S0^2))*(1/(sigma*sigma*(T/M)))*(1+log(x)
-log(S0)-(r-0.5*sigma*sigma)*(T/M));

% Phase 2 : Use these Beta values to
% price the american option along a new set of simulated paths.
```

```
% The following is the (preffered) forward method
S=[];X=[];
X=GenerateTheGBMPaths(N,M,S0,sigma,r,T,W);
 % Default value is no-exercise till expiry
V=exp(-r*T)*payoff(X(M,:));
% Only need value of derivative at first time-step
dlgdX0=dloggdX0(X(2,:));
d2lgd2X0=d2loggd2X0(X(2,:));
delta=exp(-r*T)*payoff(X(M,:)).*dlgdX0;
gamma=exp(-r*T)*payoff(X(M,:)).*(d2lgd2X0+dlgdX0.*dlgdX0);
for n=1:N
    for m=1:M
        ContinuationVal=0;
        for i=0:R
            ContinuationVal=ContinuationVal
             +LaguerreExplicit(i,X(m,n))*Beta(m,i+1);
        end
        if ContinuationVal<payoff(X(m,n)) || m==M
            V(n)=exp(-r*(T*m/M))*payoff(X(m,n));
            delta(n)=exp(-r*m*T/M)*payoff(X(m,n)).*dlgdX0(n);
            gamma(n)=exp(-r*m*T/M)*payoff(X(m,n))
             .*(d2lgd2X0(n)+dlgdX0(n).*dlgdX0(n));
            break;
        end
    end
end

value=sum(V(:))/N;
delta_value=sum(delta(:))/N;
delta_variance=sum((delta(:)-value).^2)/N;
delta_MC_error=sqrt(delta_variance/N);
gamma_value=sum(gamma(:))/N;
gamma_variance=sum((delta(:)-value).^2)/N;
gamma_MC_error=sqrt(gamma_variance/N);
end
```

A.5. Code for Longstaff-Schwartz in 2-dimensions

A.5.1. Code for Regression in 2-dimensions

```
% LS2dBetaByRegression.m %%%%%%%%%%%%%
```

```
function Beta
=LS2dBetaByRegression(R,N,M,X0,Y0,K,sigmax,sigmay,corr,r,T,B1,B2)

% Uses adapted Longstaff-Schwartz algorithm
% Takes in two sets of random numbers from which to generate paths
% This programuses regression to fit the beta coefficients.
% 2d Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions. In algorithm, ‘
% it uses (R+1)^2 basis functions.
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)

payoff = @(x,y) max(K-max(x,y),0); % Max-put payoff

% Phase 1 : Generate N paths and approximate the coefficients Beta
X=[];Y=[];
[X Y]=Generate2DGBMPaths(N,M,X0,Y0,sigmax,sigmay,corr,r,T,B1,B1);
% Now step backwards from T=1, evaluating the beta at each time
V=payoff(X(M,:),Y(M,:));
for y=0:M-1 % might need to fix these indices
    m=M-y;
%    disp(N+m); % This lets the user know how long is left.
    for j=0:R
    for k=0:R
        for i=0:R
        for s=0:R
            n=[1:N];
            B_psi_psi((j+1)*(k+1),(i+1)*(s+1))
             =sum(Laguerre2d(j,k,X(m,n),Y(m,n))
             .*Laguerre2d(s,i,X(m,n),Y(m,n)));
            B_psi_psi((j+1)*(k+1),(i+1)*(s+1))
             =B_psi_psi((j+1)*(k+1),(i+1)*(s+1))/N;
        end
        end
    end
    end
    for j=0:R
    for k=0:R
        n=[1:N];
        B_V_psi((j+1)*(k+1))=sum(V(n).*Laguerre2d(j,k,X(m,n),Y(m,n)));
        B_V_psi((j+1)*(k+1))=B_V_psi((j+1)*(k+1))/N;
    end
    end
```

```
    Beta(m,:)=inv(B_psi_psi)*B_V_psi'; % These are the coefficients
    % Now update the value of the option along these paths
    for n=1:N
        ContinuationVal=0;
        for j=0:R
        for k=0:R
            ContinuationVal=ContinuationVal
             +Laguerre2d(j,k,X(m,n),Y(m,n))*Beta(m,(j+1)*(k+1));
        end
        end
        if ContinuationVal<payoff(X(m,n),Y(m,n))
            V(n)=payoff(X(m,n),Y(m,n));
        else
            V(n)=exp(-r*(T/M))*V(n); % Use Longstaff-Schwartz
        end
    end
end
end
```

A.5.2. Code for Evaluation in 2-dimensions

```
% LSMultiDimensionalVal.m %%%%%%%%%%%%%%%%%%%%%5

function [value variance MCerror]
=LSMultiDimensionalVal(R,N,M,X0,Y0,K,sigmax,sigmay,corr,r,T,Beta,B1,B2)

% Uses adapted Longstaff-Schwartz algorithm
% Takes in 2 sets of random numbers, and generates a 2d path from them
% Evaluates the Bermudan option with M exercise dates and maxput payoff
% 2d Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions. In algorithm, it uses R^2 basis functions.
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)

payoff = @(x,y) max(K-max(x,y),0); % Max-put payoff

% Phase 2 : Use these Beta values to price the american
% option along a new set of simulated paths.
X=[];Y=[];
[X Y]=Generate2DGBMPaths(N,M,X0,Y0,sigmax,sigmay,corr,r,T,B1,B2);
% This is the (preferred) forward method
 % Default value is no-exercise till expiry
```

```
V=exp(-r*T)*payoff(X(M,:),Y(M,:));
for n=1:N
    for m=1:M
        ContinuationVal=0;
        for j=0:R
        for k=0:R
            ContinuationVal=ContinuationVal
             +Laguerre2d(j,k,X(m,n),Y(m,n))*Beta(m,(j+1)*(k+1));
        end
        end
        if ContinuationVal<payoff(X(m,n),Y(m,n))
            V(n)=exp(-r*(T*m/M))*payoff(X(m,n),Y(m,n));
            break;
        end
    end
end

value=sum(V)/N;
variance=sum((V-value).^2)/N;
MCerror=sqrt(variance/N);
end
```

A.5.3. Code for Pathwise Greeks in 2-dimensions

```
% LS2dPathwiseGreeks.m %%%%%%%%%%%%%%%%%%%%

function [value valueMCE deltaXval
deltaxMCE deltaYval deltayMCE vegaXval vegaxMCE
 vegaYval vegayMCE rhoval rhoMCE]=
LS2dPathwiseGreeks(R,N2,M,X0,Y0,K,sigmax,sigmay,corr,r,T,Beta,B1,B2)

% Uses adapted Longstaff-Schwartz algorithm
% Takes in 2 sets of random number from which it creates 2d paths
% Evaluates various greeks by a pathwise-sensitivity method
% Weighted Laguerre polynomials are used as the basis functions.

%R Number of basis functions. In algorithm, it uses R^2 basis functions.
%N Number of paths
%M Number of exercise dates (approximate American with Bermudan)

payoff = @(x,y) max(K-max(x,y),0); % Max-put payoff
% These are used for Pathwise Delta
dPayoffdX = @(x,y) (-(K>max(x,y)).*(x>y));
dPayoffdY = @(x,y) (-(K>max(x,y)).*(y>x));
```

```
% Phase 2 : Use these Beta values to price the
% bermudan option and its greeks along a new set of N2 simulated paths.
X=[];Y=[];dXdsigmaX=[];dXdsigmaY=[];
%[X Y]=Generate2DGBMPaths(N,M,X0,Y0,sigmax,sigmay,corr,r,T);
% In order to calculate the greeks, I will need to derive special processes
deltaT=T/M;
X=zeros(M,N2);
X(1,:)=X0*ones(1,N2);
Y=zeros(M,N2);
Y(1,:)=Y0*ones(1,N2);
Bhat=corr*B1(:,:)+sqrt(1-corr*corr)*B2(:,:);
sumB1=cumsum(B1); % The sum of B1 along path N up to time M
sumBhat=cumsum(Bhat);
dXdsigmaX=zeros(M,N2);
dYdsigmaY=zeros(M,N2);
dXdr=zeros(M,N2);
dYdr=zeros(M,N2);
for m=1:M-1
    X(m+1,:)=
    X(m,:).*exp((r-0.5*sigmax*sigmax)
     *deltaT+sigmax*B1(m,:)*sqrt(deltaT));
    Y(m+1,:)=
    Y(m,:).*exp((r-0.5*sigmay*sigmay)
     *deltaT+sigmay*(Bhat(m,:))*sqrt(deltaT));
    dXdsigmaX(m+1,:)=
    (sumB1(m,:)*sqrt(deltaT)-sigmax*deltaT*m).*X(m+1,:);
    dYdsigmaY(m+1,:)=
    (sumBhat(m,:)*sqrt(deltaT)-sigmax*deltaT*m).*Y(m+1,:);
    dXdr(m+1,:)=deltaT*m*X(m+1,:);
    dYdr(m+1,:)=deltaT*m*Y(m+1,:);
end
dXdX0=X/X0;
dYdY0=Y/Y0;
% Default value is no-exercise till expiry
V=exp(-r*T)*payoff(X(M,:),Y(M,:));
deltaX=exp(-r*T)*dPayoffdX(X(M,:),Y(M,:)).*dXdX0(M,:);
deltaY=exp(-r*T)*dPayoffdY(X(M,:),Y(M,:)).*dYdY0(M,:);
vegaX=exp(-r*T)*dPayoffdX(X(M,:),Y(M,:)).*dXdsigmaX(M,:);
vegaY=exp(-r*T)*dPayoffdY(X(M,:),Y(M,:)).*dYdsigmaY(M,:);
rho=exp(-r*T)*(dPayoffdX(X(M,:),Y(M,:)).*dXdr(M,:)
+dPayoffdY(X(M,:),Y(M,:)).*dYdr(M,:)-T*payoff(X(M,:),Y(M,:)));
for n=1:N2
    for m=1:M
```

```
        ContinuationVal=0;
        for j=0:R
        for k=0:R
            ContinuationVal=ContinuationVal
            +Laguerre2d(j,k,X(m,n),Y(m,n))*Beta(m,(j+1)*(k+1));
        end
        end
        if ContinuationVal<payoff(X(m,n),Y(m,n))
            V(n)=exp(-r*(deltaT*m))*payoff(X(m,n),Y(m,n));
            deltaX(n)=exp(-r*deltaT*m)
             *dPayoffdX(X(m,n),Y(m,n))*dXdX0(m,n);
            deltaY(n)=exp(-r*deltaT*m)
             *dPayoffdY(X(m,n),Y(m,n))*dYdY0(m,n);
            vegaX(n)=exp(-r*deltaT*m)
             *dPayoffdX(X(m,n),Y(m,n))*dXdsigmaX(m,n);
            vegaY(n)=exp(-r*deltaT*m)
             *dPayoffdY(X(m,n),Y(m,n))*dYdsigmaY(m,n);
            rho(n)=exp(-r*deltaT*m)
             *(dPayoffdX(X(m,n),Y(m,n))*dXdr(m,n)
             +dPayoffdY(X(m,n),Y(m,n))*dYdr(m,n)
             -deltT*m*payoff(X(m,n),Y(m,n)));
            break;
        end
    end
end

value=sum(V)/N2;
deltaXval=sum(deltaX)/N2;
deltaYval=sum(deltaY)/N2;
vegaXval=sum(vegaX)/N2;
vegaYval=sum(vegaY)/N2;
rhoval=sum(rho)/N2;

valueVariance=sum((V-value).^2)/N2;
valueMCE=sqrt(valueVariance/N2);

deltaxVariance=sum((deltaX-deltaXval).^2)/N2;
deltaxMCE=sqrt(deltaxVariance/N2);

deltayVariance=sum((deltaY-deltaYval).^2)/N2;
deltayMCE=sqrt(deltayVariance/N2);

vegaxVariance=sum((vegaX-vegaXval).^2)/N2;
vegaxMCE=sqrt(vegaxVariance/N2);
```

```
vegayVariance=sum((vegaY-vegaYval).^2)/N2;
vegayMCE=sqrt(vegayVariance/N2);

rhoVariance=sum((rho-rhoval).^2)/N2;
rhoMCE=sqrt(rhoVariance/N2);

end
```

A.6. Code for Hedging Error

```
% Find the hedging error of the Longstaff-Schwartz algorithm
% Assume holder of option and counterparty (person doing hedging)
% are using the same stopping rule.

clear;

P=10000; % Number of paths over which to find error
N=200; % Number of sub paths used in evaluation in hedging,
       % and the number of paths used to define the stopping
       % rule for the holder of the option.

M=5;
R=3;

S0=11;
K=15;
sigma=0.4;
r=0.05;
T=1;

payoff = @(x) max(K-x,0); % Put payoff

W=randn(M,N);
Beta=LSBetaByRegression(R,N,M,S0,K,sigma,r,T,W);

% Paths we simulate hedging error for
X=GenerateGBMPaths(P,M,S0,sigma,r,T);

% At each time-step find the average hedging error
HedgingError=zeros(M,1);
Exercises=zeros(M,1);
```

```
tic;
HE=zeros(P,1);
for n=1:P
    % At the start of each path, you initialise the portfolio
    W=randn(M,N);
    BetaSub=LSBetaByRegression(R,N,M,S0,K,sigma,r,T,W);
    W=randn(M,N);
    [delta delta_variance delta_MCE]
     =LSPathwiseDelta(R,N,M,S0,K,sigma,r,T,BetaSub,W);
    [Vhedged variance MCE]
     =LSAmericanPutVal(R,N,M,S0,K,sigma,r,T,BetaSub,W);
    B=Vhedged-delta*S0;

    for m=2:M
    % The cash position of the portfolio grows at the risk-free rate.
        B=B*exp(r*(T/M));

        % Use the stopping rule to check if option holder exercises
        ContinuationVal=0;
        for i=0:R
            ContinuationVal=ContinuationVal
             +LaguerreExplicit(i,X(m,n))*Beta(m,i+1);
        end
        if ContinuationVal<payoff(X(m,n)) || m==M
            V(n)=exp(-r*(T*m/M))*payoff(X(m,n));
          % If the option is exercised, realise your errors, and discount
            % to 0.
            HE(n)=exp(-r*m*T/M)*(-K+B+delta*X(m,n)+X(m,n));
            break;
        end

        % Adjust the hedging portfolio if the option is not exercised
        W=rand(M-m,N);
        BetaSub=LSBetaByRegression(R,N,M-m,X(m,n),K,sigma,r,T-(T/M)*m,W);
        W=randn(M-m,N);
        delta_old=delta;
        [delta delta_variance delta_MCE]
         =LSPathwiseDelta(R,N,M-m,X(m,n),K,sigma,r,T-(T/M)*m,BetaSub,W);
        % Set the new cash position
        B=B+(delta_old-delta)*X(m,n);

    end
end
toc;
```

```
meanHE=mean(HE);
MCE_HE=sqrt(var(HE)/P);
```

Bibliography

[1] P. Glasserman, B. Yu, Number of Paths Versus Number of Basis Functions in American Option Pricing (Columbia University) (2003).

[2] 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.

[3] Lajos Gyurko, Private conversations with me concerning pathwise and likelihood ratio method

[4] Emanuelle Clément, Damien Lamberton, Philip Protter, An Analysis of a Least Squares Regression Method for American Option Pricing(Finance and Stochastics) (2002)

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

[6] Mark Joshi, The Concepts and Practice of Mathematical Finance(Cambridge) (2003)

[7] J.C.Cox,S.Ross,M.Rubinstein, Option Pricing: a simplified appraoch(Journal of Financial Economics 7)(1979)(229-63)

[8] L.C.G. Rogers, Monte Carlo Valuation of American Options(Mathematical Finance)(2002)(12:271-286)

[9] Martin B. Haugh, Leonid Kogan, Pricing American Options: A Duality Approach(Operations Research)(2004)(Vol 52, 258-270)

[10] Christoph Reisinger, Finite Difference Methods in Computational Fincance (Lecture course delivered 2008-2009 to MSc Mathematical and Computational Finance, University of Oxford)