NUMERICAL METHODS REPORT
Robert Buck
Student number: U1963874

Contents:

Foreword:

Note that For Part 1 of the project unless otherwise stated the parameters used to obtain all outputs are as follows: K_1 = 90, K_2 = 120, T = 0.5, r = 0.03, sigma = 0.25, Smin = 1, Smax = 200, error = 0.05. Where error refers to the largest possible pricing error at a 95% confidence interval. Equally for Part 1 unless otherwise stated the parameters used to obtain all outputs are as follows: K = 50, T = 1, r_0= 0.05, σ_0 = 0.3, Smin = 1, Smax = 100, err = 0.05, Sb = 30, Sbmin = 0 and Sbmax = 50. Where err refers to the largest possible pricing error at a 95% confidence interval. Details of how the graphs and outputs in tables given throughout the report can be replicated are described in appendix section “Table & graph outputs”.

Part 1 Pricing Bull Spread:

Part 1 of the project aims to price a bull call spread by several numeric methods and compare these approaches. A bull call spread is a is a strategy where the trader buys a call option with strike K_1 and simultaneously sells another call option with strike K_2 where $K_1 < K_2$. This strategy gives you positive exposure to the underlying asset capped at a payoff of $K_2 - K_1$.

1.1 Transformation of Black-Scholes PDE

Let V(S,t) be the price of the European bull call spread at time t with underlying price of S. K_1 and K_2 are the strike prices of the calls purchased and sold respectively. Under the various assumptions of the Black Scholes model, e.g. constant volatility, underlying evolves according to geometric Brownian motion e.c.t., the value of the bull spread must satisfy the Black-Scholes PDE:

$$\text{Black-Scholes PDE: } \frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \quad (1)$$

$$\text{Payoff: } V(S,T) = \max(S-K_1,0)-\max(S-K_2,0) \quad (2)$$

$$\text{Boundary Conditions: } V(S_{min},t) = 0 \quad (3)$$

$$V(S_{max}, t) = (K_2 - K_1)\, e^{-r(T-t)} \quad (4)$$

where $S \in [S_{min}, S_{max}]$, σ is the volatility of the underlying, r is the risk-free rate, T is the expiration time and t is the current time.

To implement the transformation, we consider the following change of variables:

$$S = K_1 e^x \Rightarrow x = \log(S/K_1) \quad (5)$$

$$t = T - \frac{2\tau}{\sigma^2} \Rightarrow \tau = \frac{\sigma^2}{2}(T-t) \quad (6)$$

$$V = K_1\omega(x,\tau) = \omega\left(\log(S), \frac{\sigma^2}{2}(T-t)\right) \quad (7)$$

Using this change in variable in addition to setting $k = \frac{2r}{\sigma^2}$ transforms (1) into:

$$\omega_\tau = \omega_{xx} + (k-1)\,\omega_x - k\omega = 0 \quad (8)$$

A second substitution is now used on (8). Here $\omega(x,\tau) = e^{\alpha x+\beta\tau}\, u(x,\tau)$, where $\alpha = -\frac{1}{2}(k-1)$ and $\beta = \frac{1}{4}(k+1)^2$.

This results in: $u_\tau = u_{xx}$ hence, we have obtained the heat equation.

We now need to transformation the Initial and Boundary conditions.

For the IC:

$$u(x,\tau) = e^{-\alpha x}(\max(S-1,0)-\max(S-K_2/K_1,0)) \quad (9)$$

For the BC's:

$$u(x_{max},\tau) = (K_2/K_1 - 1)\, e^{-(\alpha x max+\beta\tau+2r\tau/\sigma^2)} \quad (10)$$

1.2 Solution to the Black-Scholes PDE

The Black-Scholes PDE was solved via firstly using a transformation to turn the Black-Scholes PDE into the heat equation and then implementing the Crank-Nicholson method to solve the heat equation. We use the "tridiag" function to solve the system of equations produced in the numeric solution to the Black-Scholes PDE, rather than gaussian elimination. The number of grid points in time, N, and space, J, were chosen such that the maximum absolute error of the numerical solution and the exact solution to the Black-Scholes formula, "blsprice", is less than £0.05. This is done in the function "Min_err_bullspread". Specifically, this function runs a while loop which increase both N and J by an additional power each loop, where N and J are set as 2 initially, until the error is less than £0.05. For example, in the first loop $N = 2^1$ and $J = 2^1$ in the second $N = 2^2$ and $J = 2^2$ e.c.t. The error in each loop is computed via that the infinity norm of the difference between the numerical approximation and the exact solution of the PDE,

$\|Vexact - Vpde\|_\infty$. While this is larger than 0.05 the loop continues to run. The values of N and J that ensured the pricing error was less than £0.05 were found to be 128. These corresponded with a pricing error of £0.0215.
To calculate the CPU time required to compute the numeric solution to the Black-Scholes PDE for given N and J such that $\|Vexact - Vpde\|_\infty < 0.05$ we use MATLAB's built in function "cputime". The CPU time required is very short. As a loop is used to run the solution 100 times and then we complete the average CPU time. This is done in the function "PDE_euro_call_bull_CPU". The CPU time required is 0.0036 seconds a computer with the following specifications: Processor: Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz 3.41 GHz , RAM: 16.0 GB. A plot of the numeric solution to the Black-Scholes PDE is shown in Figure 1.1. The financial intuition behind why this solution is sensible is as follows: a call bull spread is created by buying a call option with strike price K_1 and at the same time selling another call option with strike price K_2 where $K_1 < K_2$. In our case K_1=£90 and K_2 = £120. The payoff of our bull spread is as follows: max(S-K_1 ,0) -max(S-K_2 ,0). Where S is the spot price at expiration. Therefore, the payoff of the bull spread is capped at K_2 - K_1. In our case this is £30. As the option gets deeper into the money, S(t) > 90, the value of the spread will approach £30 but never actually become £30 for two reasons. Firstly, to account for the possibility no matter how remote that at expiration the option could still expire at a value less than £30. Secondly, due to the fact that the payoff in the future is discounted at the risk-free rate so even if a payoff of £30 was guaranteed with zero risk the value of the bull spread would be less than £30 currently. As the option gets further out of the money S(t) < 90 the value of the spread approaches 0 but never actually becomes zero to account for the fact that at expiration the spread could expire in the money no matter how remote the probability. Finally, we see that the call bull spread is an increasing function of spot price. This makes sense as the spot increases so does the likelihood that the option will expire with a higher payoff.

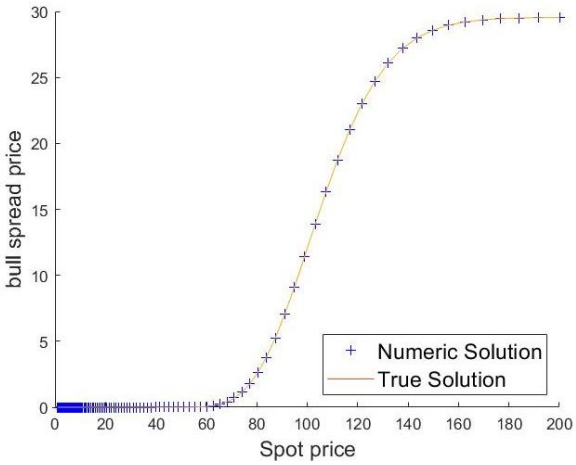

Figure 1.1: The NUMERIC SOLUTION of a bull call spread via Crank-Nicholson solution to Black-Scholes PDE vs the exact solution. Source of plot function "PDE_euro_call_bull_CPU"

1.3 Computing the delta

When computing the delta of the bull spread this is done via the use of a finite difference representation of the first derivative of the bull spread price with respect to a change in the price of the underlying. This can be achieved in two ways either by a one-sided finite difference (forward difference) or a centred difference:
One-sided difference:

$$\Delta = \frac{\partial V}{\partial \theta} = \frac{V(\theta+\delta\theta)-V(\theta)}{\delta\theta} + O(\delta\theta) \quad (11)$$

Adjusted for irregularities in the grid of the underlying prices gives:

$$\Delta = \frac{\partial V}{\partial \theta} = = \frac{V(S(j+1))-V(Sj)}{S(j+1)-S(j)} + O(\delta\theta) \quad (12)$$

Centred difference:

$$\Delta = \frac{\partial V}{\partial \theta} = \frac{V(\theta+\delta\theta)-V(\theta-\delta\theta)}{2\delta\theta} + O(\delta\theta^2) \quad (13)$$

Adjusted for irregularities in the grid of the underlying prices gives:

$$\Delta = \frac{\partial V}{\partial \theta} = = \frac{V(S(j+1))-V(S(j-1))}{S(j+1)-S(j-1)} + O(\delta\theta^2) \quad (14)$$

Fact that the denominator for (12) and (14) depends on the spot price indicates that the denominator is adjusted as you calculate the delta for different spot prices due to the irregularities of the spot price grid.

Figure 1.2 below shows plots of the deltas estimated via these finite difference methods and their corresponding errors. We see that the one-sided difference approximation has larger errors than that of the centred difference approximation by a factor of 1000. In addition, the one-sided difference is shifted to the left relative to the true delta.

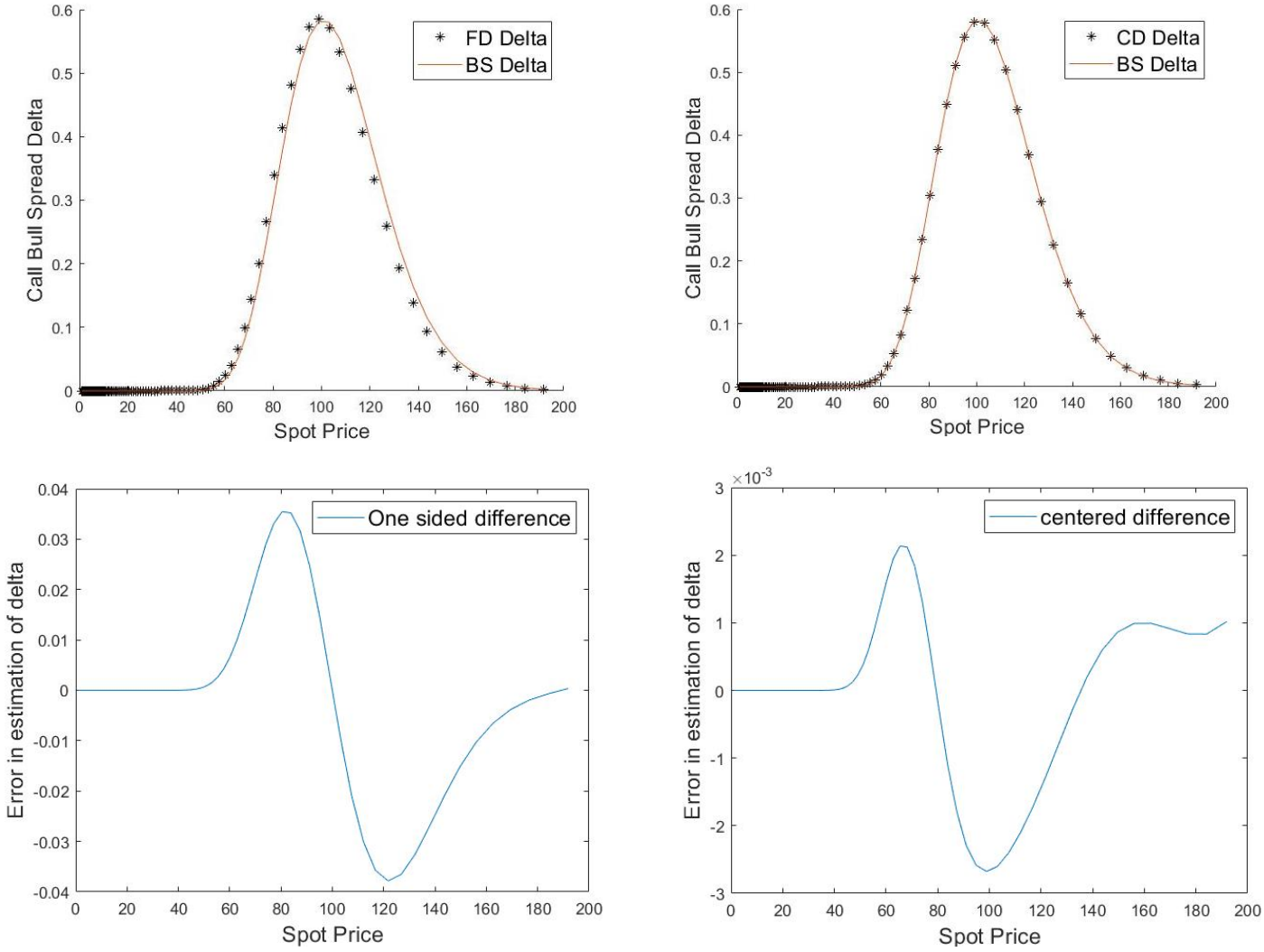

Figure 1.2: Plots of the delta obtained via both centered (CD) and one sided, forward (FD), difference approximation from the Black-Scholes PDE and its exact value. Plots also of the error associated with both the centered and one sided difference approximations of delta. These plots come from the functions "Delta_FD" and "Delta_CD" respectively.

The maximum errors associated with both of these estimations of delta were as follows:

$$\| \Delta \text{One\_sided} - Exact\_Delta \|_{\infty} = 0.0378$$

$$\| \Delta Centered\_difference - Exact\_Delta \|_{\infty} = 0.0027$$

The shape of these delta graphs makes sense from a financial perspective via the following intuition: As an option goes from being out the money to in the money it becomes much more sensitive to the underlying spot price. So initially as spot prices increase the delta of the call option with strike K1 dominates the overall delta of the bull spread. However, as spot increases further above £90 the delta of the call with strike K2, which is negative as this call is sold, begins to have more of an impact. This in turn causes the delta to peak at around a spot of 105. The effect of the sold call option on the spreads delta also explains why the delta never reaches the theoretical maximum seen in call options i.e. 1. Though the larger the difference between K_1 & K_2 the larger your maximum delta of the bull spread becomes.

1.4 Monte Carlo pricing (bull spread):

In order to use Monte Carlo approach to price the bull spread we must firstly assume that the underlying follows a geometric Brownian motion which has the following solution:

$$S(T)=S_0 \exp\left(\left(r-\frac{1}{2}\sigma^2\right)T+\sigma W\right) \Rightarrow S(T)=S_0 \exp\left(\left(r-\frac{1}{2}\sigma^2\right)T+\sigma\sqrt{T}\,X\right) \qquad (15)$$

Where X~N(0,1) and $S_0 = S(t=0)$

We also know that the discounted payoff of the bull spread is given as follows:

$$f(S(T)) = e^{-rT}(\max(S(T) - K_1, 0) - \max(S(T) - K_2, 0)), \quad (16)$$

1.4.1 Naive

Simplest approach. Firstly, generate N i.i.d standard normally distributed random variables. For each of these random variables then compute the corresponding S(T) via (15). The discounted payoff for the bull spread can then be computed via (16). Finally, the average of these values is taken to compute price of the bull spread.

1.4.2 Antithetic variance reduction

The intuition behind this approach is to realise that we are interested in the mean of the discounted payoffs. As such if we can find a sequence of random variables who have expectation equal to $\boldsymbol{E}[f(S(T))]$ but which are negatively correlated with f(S(T)) then a combination of this random variable and f(S(T)) will still have expectation equal to $\boldsymbol{E}[f(S(T))]$ but a lower variance. In our case this random variable can be found by noting that X~N(0,1) is symmetric around 0 so $-X$~N(0,1) but has perfect negative correlation with X. We then generate two values for the spot price at time T, one generated by X the other generated by $-X$, these are given by (17) and (18). Note that reusing the random variables X to form $-X$ is known as path recycling.

$$S(T)^+ = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}\,X\right) \quad (17)$$

$$S(T)^- = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T - \sigma\sqrt{T}\,X\right) \quad (18)$$

We then substitute (17) and (18) into (16), obtaining the payoffs $f(S(T)^+)^+$ and $f(S(T)^-)^-$ respectively. Finally, we average (17) and (18) and take the mean to obtain the price of the bull call spread.

1.4.3 Control variates

In order to use control variates we seek to adjust the sample mean of the discounted payoff of the bull spread in order to make it a better approximation of the true mean of the discounted payoff. In order to do this, we need to find a function $g(S(T))$ that is highly correlated with f(S(T)) but such that we know both the mean and variance of $g(S(T))$. Given our payoff, $\max(S(T) - K_1,0) - \max(S(T)-K_2,0)$, an obvious choice of $g(S(T))$ would be to use S(T) as this will be perfectly correlated with the payoff between K_1 and K_2 . S(T) has known expectation and variance given by:

$$\bar{g}(S(T)) = \boldsymbol{E}\,[g(S(T))] = \boldsymbol{E}\,[S(T)] = S_0\, e^{rT} \quad (19)$$

$$\text{var}\,[g(S(T))] = \text{var}\,[S(T)] = S_0^2\, e^{2rT}\left(e^{\sigma^2 T} - 1\right) \quad (20)$$

Our control variates estimate is then given by:

$$f_c = f(S(T)) - c\,(g(S(T) - \bar{g}(S(T))\,, \quad (21)$$

Next note the variance of f_c is given by:

$$\text{var}\,[f_c] = \text{var}\,[f(S(T))] - 2c\,\text{cov}\left[f(S(T)), g(S(T)\right] + c^2\,\text{var}\,[g(S(T))] \quad (22)$$

Given that we are seeking to minimise the variance of f_c this will be achieved when we set c as the following:

$$c = \frac{cov(f(S(T)), g(S(T)))}{var(g(S(T)))} \quad (23)$$

Once f_c is calculated then so is its mean which will correspond to the bull spread price.

1.4.4 Importance sampling

The goal of Important Sampling is to ensure that we only generate random variables over the range of the S(T) that will have a “important” impact on the bull spread payoff. Specifically, we want to ensure that our random variables do not produce S(T) values less than £90 as below this value the payoff is zero. To achieve this, we note the following: firstly, we are drawing normally distributed random variables X. Secondly, that the cumulative distribution function will always be uniformly distributed. As such we can rewrite equation (15) as follows:

$$S(T) = S_0 \exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}\,\Phi^{-1}(Y)\right), \qquad Y \sim U[0,1]. \quad (24)$$

Where Φ^{-1} is the inverse of the normal cdf. Now note that computing the value of the bull spread is done via taking the expectation of the discounted payoffs with respect to all of the simulated normally distributed random variables. As such:

$$V\big(S(T)\big) = \boldsymbol{E}[f(S(T))] = \int_0^1 f(S(T)y)p_1(y)dy \quad (25)$$

Where

$$S(T)(y) = \text{S0}\exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}\,\Phi^{-1}(y)\right) \quad (26)$$

and $p_1(y)$ is the probability density function of Y.
We know that $f(S(T))$ = 0 for all $S(T)$ < K_1 . Therefore, (25) can be re written as:

$$\int_0^1 f(S(T)y)p_1(y)dy = \int_{y_0}^1 f(S(T)y)\,p_1(y)dy \quad (26)$$

We can now use the following change in pdf to consider only the pdf for the uniform distribution over [y_0,1]:

$$p_2(y) = \frac{1}{1-y_0} \quad (27)$$

This gives the following for the value of the bull spread:

$$V\big(S(T)\big) = (1-y_0)\int_{y_0}^1 f(S(T)y)\,p_2(y)dy \quad (28)$$

Now solving (26) for y_0 gives:

$$y_0 = \frac{\Phi\left[\ln\left(\frac{K_1}{S_0}\right) - \left(r - \frac{1}{2}\sigma^2\right)T\right]}{\sigma\sqrt{T}}\,. \quad (29)$$

As such the value of the bull spread can be calculated by:

$$V\big(S(T)\big) = (1-y_o)\frac{1}{N}\sum_{i=1}^{N} f(S_i(T)) \quad (30)$$

1.5 Comparison of Monte Carlo approaches:

1.5.1 CPU times and sample sizes

Firstly, we must calculate the number of simulations required to ensure that the price of the bull spread as calculated by all the Monte Carlo methods is within £0.05 of the true price with a 95% confidence interval. In order to do this, we need to recall the central limit theorem and observe the following

$$\|\mathrm{V_{MC}} - \mathrm{V}\|_\infty = 0.05 \geq \frac{1.96\sigma}{\sqrt{N}} \quad (31)$$

Where 1.96 is the corresponding z score for a two-sided 95% confidence interval.
Rearranging for N gives:

$$\mathrm{N} \geq \left(\frac{1.96}{0.05}\right)^2 \sigma^2 \quad (32)$$

In order to calculate the N required to ensure the pricing error is below £0.05 with a 95% confidence interval over the range of spot prices 1≤S≤200 prices the function “MC_Pricing” and “MC_Pricing_recycle” are used. MC_Pricing simulates a new N normally distributed random variables for each spot price in the range of 1≤ S≤200 we are looking at. MC_Pricing_recycle however will use path recycling and so compute N normally distributed random variables once and then use these same random variables for the entire range of 1≤ S≤200. Use of the phrase “path recycling” may seem a bit unusual given that we do not technically estimate the path of the underlying price. However, we believe it is correct terminology as in this context we can view the path as having a single time step and so one output for each random variable simulated. As such it will be less computationally intensive. To determine the N required to ensure the pricing error is below £0.05 both functions will run the 4 types of Monte Carlo simulations with 1000 normally distributed random variables created initially. These will then be used to determine the range of variances produced for each spot price. Each of the corresponding variances are then fed into equation (32) to produce a vector containing the sample sizes required to ensure that there is a pricing error of less than £0.05 with a 95% confidence interval for each spot price. The infinity norm is then taken of this vector and the answer is rounded up using the ceil function. This gives the

N required to ensure a pricing error less than £0.05 occurs 95% of the time over the entire range of 1≤ S≤200. The CPU time required to run each of the Monte Carlo methods with N set such that a pricing
error less than £0.05 occurs 95% of the time was computed over the entire range of spot prices 1≤S≤200, i.e. the CPU time required to price the option across all these spot prices. This was done via computing the CPU time for each spot price and then summing the CPU times taken for each spot price. The CPU times required, and the number of simulations needed to ensure a pricing error less than £0.05 95% of the time both with and without path recycling are shown in the following tables:

	Naive method	Antithetic variance reduction	Control Variates	Importance sampling
Simulations required	2.03×10^5	3.63×10^4	5.23×10^4	1.09×10^5
CPU time	1.69	0.39	1.23	3.59

Table 1: Simulations required to ensure a pricing error of less than £ 0.05 95% of the time and the CPU times required across the the range of spot prices. Calculated using a computer with the following specifications: PROCESSOR: INTEL(R) CORE(TM) I5-7500 CPU @ 3.40GHZ 3.41 GHZ, RAM: 16.0 GB. These results come from the function "MC_Pricing" which does not use path recycling.

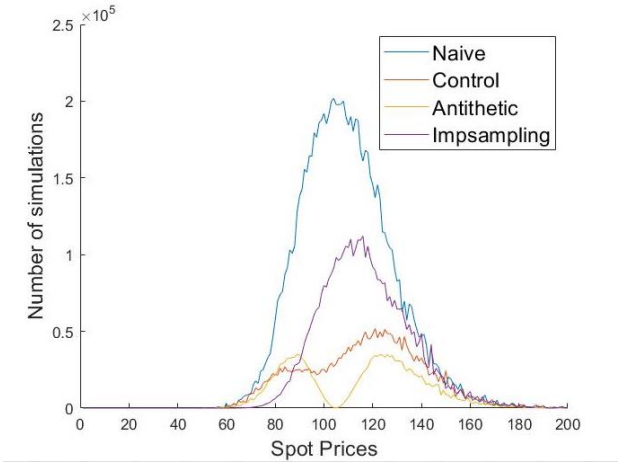

Figure 1.3: Simulations required to ensure a pricing error of less than £0.05 95% of the time not using path recycling. Output of function "MC_Pricing".

	Naive method	Antithetic variance reduction	Control Variates	Importance sampling
Simulations required	1.97×10^5	3.45×10^4	5.03×10^4	1.09×10^5
CPU time	0.78	0.20	0.94	4.06

Table 2: Simulations required to ensure a pricing error of less than £0.05 95% of the time and the CPU times required across the the range of spot prices. Calculated for a call bull spread. Calculated using a computer with the following specifications: PROCESSOR: INTEL(R) CORE(TM) I5-7500 CPU @ 3.40GHZ 3.41 GHZ, RAM: 16.0 GB. These results come from the function "MC_Pricing_recycle" which uses path recycling.

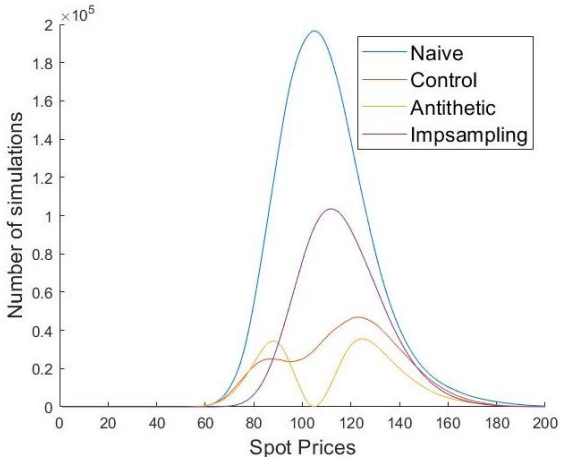

Figure 1.4: Simulations required to ensure a pricing error of less than £0.05 95% of the time, for a bull spread, using path recycling. Output of function "MC_Pricing_recycle".

We also found the price of the bull spread under all 4 of the different Monte Carlo methods. For all of the Monte Carlo methods the number of simulations is set such that the pricing error is less than £0.05 95% of the time. This can be seen in figure 1.5. Note that the function "MC_Pricing" also has the capacity to plot the graph in Figure 1.5 but it is commented out as the plot is identical.

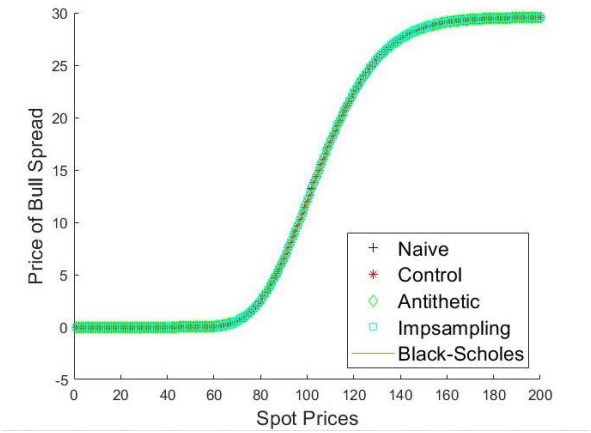

Figure 1.5: Price of the bull spread across a range of spot prices according to all 4 Monte Carlo techniques. Output of function "MC_Pricing_recycle".

1.5.2 Computing the delta with antithetic variance reduction:

The delta of the bull spread is computed using a combination of antithetic variance reduction and path recycling. Specifically, we consider the centred difference approach to compute the delta, recall (13):

$$\Delta = \frac{\partial V}{\partial \theta} = \frac{V(\theta+\delta\theta)-V(\theta-\delta\theta)}{2\delta\theta} + O(\delta\theta\text{^}2)$$

Note that we are now dealing with a uniform grid of Spot prices as such (13) becomes:

$$\Delta = \frac{\partial V}{\partial \theta} = = \frac{V(S(j+1))-V(S(j-1))}{2\Delta S} + O(\Delta S^2) \qquad (33)$$

Also we can observe that:

$$\text{var}\left[\frac{V(S(j+1))-V(S(j-1))}{2\Delta S}\right] \simeq \text{var}\left[\frac{\partial V}{\partial S}(\text{Sj})\right] \quad (34)$$

In order to reduce the variance of our delta estimator antithetic variance reduction can be applied to both $V(S(j+1))$ and $V(S(j-1))$ which will in turn reduce the overall variance of $\frac{\partial V}{\partial S}(\text{Sj})$. For the computation of the antithetic delta we use 1000 simulations and set dS=1 and simulate the following values of the underlying at maturity:

$$(S(T)_{right})^{+} = (S_0 + dS)\exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}\,X\right) \quad (35)$$

$$(S(T)_{left})^{+} = (S_0 - dS)\exp\left(\left(r - \frac{1}{2}\sigma^2\right)T + \sigma\sqrt{T}\,X\right) \quad (36)$$

$$(S(T)_{right})^{-} = (S_0 + dS)\exp\left(\left(r - \frac{1}{2}\sigma^2\right)T - \sigma\sqrt{T}\,X\right) \quad (37)$$

$$(S(T)_{left})^{-} = (S_0 - dS)\exp\left(\left(r - \frac{1}{2}\sigma^2\right)T - \sigma\sqrt{T}\,X\right) \quad (38)$$

Left and right indicate shifts in the initial price by either – dS or + dS respectively. We then proceed to compute the discounted payoff for each of these underlying values via equation (16). We can then compute the value of the left and right payoffs as follows:

$$f(S(T)_{right}) = 0.5*[f((S(T)_{right})^{+}) + f((S(T)_{right})^{-})] \quad (39)$$

$$f(S(T)_{left}) = 0.5*((S(T)_{left})^{+} + (S(T)_{left})^{-}) \quad (40)$$

The by taking the mean of $f(S(T)_{right})$ and $f(S(T)_{left})$ we obtain the value of the bull spread for an incremental increase or decrease in the spot price, V_right and V_left. The delta is then simply estimated by:

$$\Delta = \frac{\text{V\_right - V\_left}}{2dS} \quad (41)$$

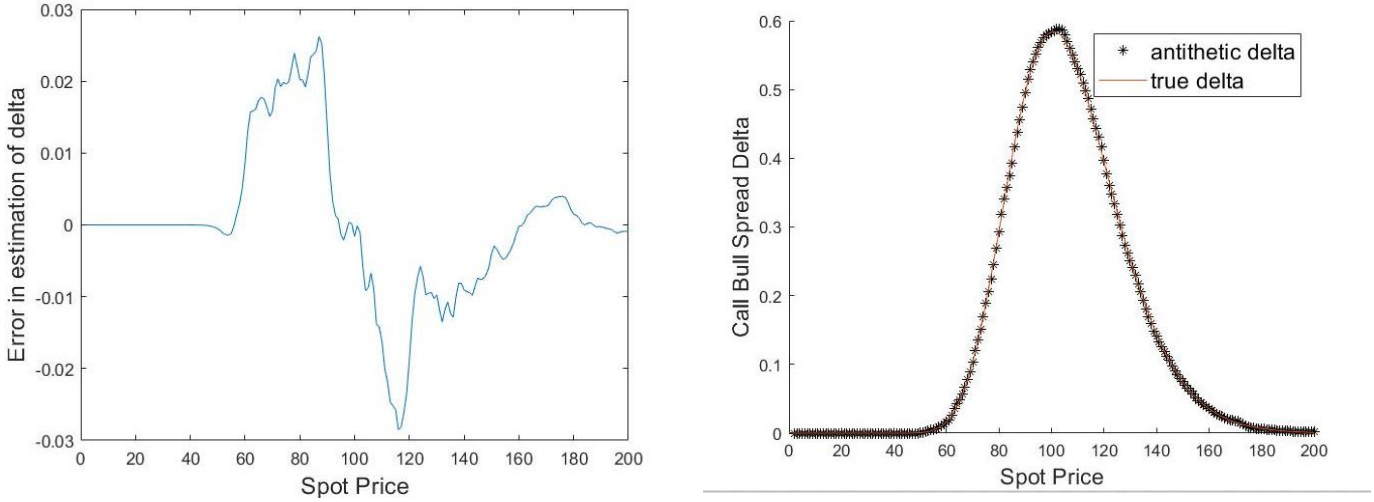

Figure 1.6: Price of the delta of the bull spread across a range of spot prices using antithetic variance reduction and of the error in this delta estimation when compared to the true black scholes delta. Output of function "delta_bull_antithetic" run with 1000 simulations.

We can also compute the maximum error in the delta estimated from our antithetic approach which is as follows:

$$\|Estimated\_Delta - Exact\_Delta\|_{\infty} = 0.0270$$

We can see from figure 1.6 that the delta calculated with antithetic variance reduction is a reasonably good approximation of the true delta. However, the estimation can be slightly problematic around the strike prices of the two calls used to produce the spread, i.e. at £90 and £120.

1.5.3 Comparison of Monte Carlo Methods:

Looking at Table 1 and 2 and Figures 1.3 and 1.4 it becomes apparent that unsurprisingly the naïve approach is the worst performer in terms of the number of simulations required to ensure a pricing error less than £0.05 95% of the time. Requiring a sample size of either 2.03×10^5 without using path recycling or 1.97×10^5 using path recycling to price the spread with a pricing error less than £0.05 over the whole range of underlying prices. Antithetic produced the best results in terms of both the number of simulations required, 3.63×10^4 without path recycling and 3.45×10^4 with path recycling.

It was also the least computationally intensive with CPU times of 0.39 seconds without path recycling and 0.2 with path recycling. It is also worth noting that between the strike prices of the options the number of simulations required decreases to close to zero. Control variates also requires sizably less simulations than the naïve method indicating that S(T) was highly correlated with the option payoff. Important sampling was still sizably better than the naïve approach in terms of the number of simulations required, decreasing N by approximately half. However, it was the most computationally intensive by a large margin. The CPU time required to compute the important sampling estimate was over 4 times greater than the next longest CPU time in the case with recycling (control variant) and 2 times larger than the next longest CPU time in the case without recycling (naïve). In comparison to the solution of the Black-Scholes equation using the heat equation solver all the Monte Carlo approaches have a CPU time far greater. However, an advantage of the Monte Carlo approaches is their flexibility in producing prices for options that cannot be priced via the Black-Scholes equation e.g. the barrier put covered in part 2.

Part 2 Pricing a barrier option:

2.1 Implementing the local volatility model for put:

With the local volatility model, we now have the added complexity of both the interest rate and volatility of the underlying are now functions of time. We still assume that the underlying evolves according to geometric Brownian motion. However, now the parameters r and σ are non-constant. This is given by the following equation:

$$dS(t) = r(t)S(t)dt + \sigma(S(t),t)S(t)dW \tag{42}$$

In our case the functions for r and σ are given by:

$$r(t) = r_0 e^{r_1 t} \tag{43}$$

$$\sigma(S,t) = \sigma_0 \left(1 + \sigma_1 \cos(2\pi t)\right)\left(1 + \sigma_2 e^{\frac{-S(t)}{100}}\right) \tag{44}$$

Both (43) and (44) are implemented in MATLAB via the functions “rate” and “volatility”. To price the put we now need to know the entire path the underlying takes rather than simply the end value taken as volatility and the interest rate are now functions of the spot price. This can be done using Euler time stepping to implement (42) i.e.:

$$S_{n+1} = S_n \left(1 + r(t_n)\Delta t + \sigma(S_n, t_n)\varepsilon_n \sqrt{\Delta t}\right) \qquad \varepsilon_n \sim N(0,1) \tag{45}$$

The discounted payoff for the put option can then be calculated by:

$$f\big(S(T)\big) = e^{\int_0^T r(t)dt} \max\left(K - S(T), 0\right) \tag{46}$$

The following code shows how this can be implement in MATLAB, it is taken from the function “MC_Put_local_vol”:

```
% Simulate Npath paths, each with Nsteps time steps (or Nsteps+1 time points
% counting the initial condition).
S = zeros(Npaths,Nsteps+1);
epsilon=randn(Npaths,Nsteps);

% Set initial condition and time step by the Euler method.
S(:,1) = S0;

% Setting up a time vector increasing at rate dt each step
t= linspace(0,T,260*T);

% Computing the paths of the underlying
for n = 1:Nsteps
S(:,n+1) = S(:,n) .* ( 1 + rate(r_0, t(1,n))*dt+volatility(sigma_0,S(:,n)...
,t(n))*sqrt(dt).*epsilon(:,n) );
End

% computing discounted payoff
fs = exp(-sum(rate(r_0, t(1:end))*dt))*max(K-S(:,Nsteps+1),0);
```

Where t is a vector that contains the time as we move each step along the path (260 time steps in each year) and S contains the spot prices simulated across the entire path not just the price at expiration. Note that when calculating the

discounted payoff to compute the discount rate we approximate $\int_0^T r(t)\,dt$ by $\sum_0^T r(t)$ dt. This ensures the discount rate is time variant.

2.2 Monte Carlo Pricing (put):

2.2.1 Antithetic variance reduction for put:

Computing Antithetic variance reduction for the put option with a local volatility model is very similar to that described in 1.4.2. However, the main distinction now is that the underlying price needs to be computed at all points in time rather than just at the expiration time, i.e. the entire path of the underlying price needs to be estimated. This must again be done for S^+ and S^-. The following MATLAB code from the function "MC_Put_local_vol_antithetic" shows how this can be computed:

```
% Simulate Npath paths, each with Nsteps time steps (or Nsteps+1 time points
% counting the initial condition).
S_p = zeros(Npaths,Nsteps+1);
S_m = zeros(Npaths,Nsteps+1);
epsilon=randn(Npaths,Nsteps);

% Set initial condition
S_p(:,1) = S0;
S_m(:,1) = S0;

% Setting up a time vector increasing at rate dt each step
t= linspace(0,T,260*T);

% Computing paths of underlying
for n = 1:Nsteps
S_p(:,n+1) = S_p(:,n) .* ( 1 + rate(r_0, t(n))*dt+volatility...
(sigma_0,S_p(:,n),t(n))*sqrt(dt).*epsilon(:,n) );

S_m(:,n+1) = S_m(:,n) .* ( 1 + rate(r_0, t(n))*dt-volatility...
(sigma_0,S_m(:,n),t(n))*sqrt(dt).*epsilon(:,n) );
end

% Computing discounted payoffs
fS_p = exp(-sum(rate(r_0, t(1:end))*dt))*max(K-S_p(:,Nsteps+1),0);
fS_m = exp(-sum(rate(r_0, t(1:end))*dt))*max(K-S_m(:,Nsteps+1),0);
```

Again, the discount rate is approximated using a sum.

2.2.2 Control variates for put:

Again, this is very similar to the approach taken in 1.4.3. Here we choose our function g to again be the underlying price. However, if the underlying price evolves under equation (42) i.e. with non-constant interest rate and volatility, then we do not know what the expectation or variance of this control variate would be. As such we now choose our control variate to be the underlying price that evolves under geometric Brownian motion with constant interest rate, set as $r_0 = 0.05$, and constant volatility, set as $\sigma_0 = 0.3$.

$$dS(t) = r_0 S_{const} dt + \sigma_0 S_{const} dW \qquad (47)$$

The expectation and variance of S(T) can then be calculated via (19) and (20) respectively. The following code, taken from the function "MC_Put_local_vol_control" shows how this would be implemented in MATLAB:

```
% Simulate Npath paths, each with Nsteps time steps (or Nsteps+1 time points
% counting the initial condition).
S = zeros(Npaths,Nsteps+1);
S_const = zeros(Npaths,Nsteps+1);
epsilon=randn(Npaths,Nsteps);

% Set initial conditions.
S(:,1) = S0;
S_const(:,1) = S0;

% Setting up a time vector increasing at rate dt each step
t= linspace(0,T,260*T);

% known mean and variance of g=S(T)=ST
mean_ST = S0 * exp(r*T);
var_ST = mean_ST^2*(exp(sigma^2*T)-1);
```

```
% Computing paths of Spot under local volatility model and constant r &
% sigma
for n = 1:Nsteps
S(:,n+1) = S(:,n) .* ( 1 + rate(r_0, t(1,n))*dt+volatility(sigma_0,...
S(:,n),t(n))*sqrt(dt).*epsilon(:,n) );

S_const(:,n+1) = S_const(:,n).*(1 + r*dt + sigma*epsilon(:,n)*sqrt(dt));
end

% Discounted payoff
fST = exp(-sum(rate(r_0, t(1:end))*dt)).*max(K-S(:,Nsteps+1),0);

% getting our c
cov_fST_ST = cov(fST,S_const(:,end));
c = cov_fST_ST(1,2)/var_ST;

% implimenting control variance estimator
fc = fST-c*(S_const(:,end)-mean_ST);
```

It is important to note that even though $S_{const}(T)$ could be computed without computing the entire path, i.e. by using a single time step T, it would be a poor control variate for our put option. This is due to the fact that we require $S_{const}(T)$ to be highly correlated with f(S(T)) which will be achieved by ensuring that the same sequence of random variables is used. As such we simulate the entire path of S_{const} and then select the last column from the matrix to get $S_{const}(T)$.

2.2.3 Comparison of Monte Carlo methods for put:

The approach taken to ensure a pricing error less than £0.05 95% of the time was very similar to that used for the bull spread in the case without path recycling. This was computed via the function "MC_Put_Pricing_locvol". Key differences are that the underlying asset prices now range from 1≤ S≤100 and the Monte Carlo functions called are now adjusted such that the payoff is for a put option and a local volatility model is implemented. Furthermore, the number of paths set to initially determine the variance of the pricing estimate for each of the Monte Carlo methods was set to 1000. Again, the CPU times are calculated in the same manner as for the bull spread, i.e. computed for each spot price and then summed. In terms of comparing the CPU times for the naïve Monte Carlo approach, for pricing a put option without a constant volatility model, we first need to determine how we will compute the Monte Carlo simulations. I used two methods: Approach A: We are now dealing with a constant volatility and interest rate of σ_0 and r_0, as such the path of prices of the underlying for each incremental increase in time of dt no longer needs to be estimated. Therefore, a single time step of T can be used in the Euler time step. If this approach is taken, then the CPU time taken to compute the naïve Monte Carlo estimate of prices over 1≤ S≤100 is 0.25 seconds. Approach B: the naïve Monte Carlo approach could be computed using the same time stepping approach as used for the local volatility model, i.e. estimate the same paths of the underlying price with an incremental increase in time of dt each step. If this approach is used the CPU time is 0.39 seconds. This gives a difference in CPU time taken for the local volatility model and approach A of 233.77 seconds and a difference of 233.63 seconds in the case of approach B. The rational for the increase in computational intensity in approach B rather than A is now we need to generate a multitude more random variables and instead of doing a single time step we are now doing 260*T steps. The rational for the difference in computational intensity of approach B and the local volatility approach is that now for each time step of each path both the "volatility" and "rate" function must be called. These functions in turn must perform an additional 12 and 4 calculations each. Furthermore, in the case of the local volatility model we use a summation function to compute the discount rate which further increase the computational intensity.

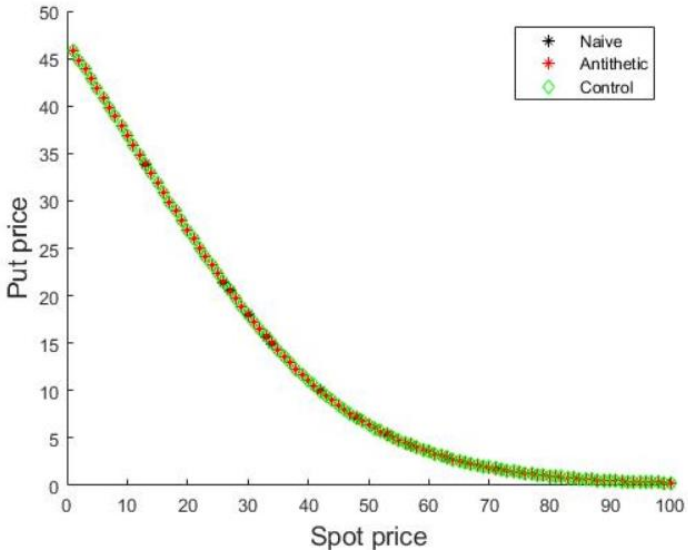

Figure 1.7: Price of the put option with a local volatility model across a range of spot prices. Output of function "MC_Put_Pricing_locvol".

	Naive	Antithetic	Control Variates
Paths required	1.83×10^5	2.75×10^4	4.96×10^4
CPU time	234.02	41.34	52.36

Table 3: Simulations required to ensure a pricing error of less than £0.05 95% of the time and the CPU times required across the the range of spot prices. Calculated for a put option with a local volatility model. Calculated using a computer with the following specifications: PROCESSOR: INTEL(R) CORE(TM) I5-7500 CPU @ 3.40GHZ 3.41 GHZ, RAM: 16.0 GB. These results come from the function "MC_Put_Pricing_locvol".

2.3 Monte Carlo methods for pricing down and out put (barrier):

2.3.1 Implementing down and out put (barrier):

A down and out put is a put which has the following payoff structure:

$\max(K - S(T), 0)$ if $\min_t S(t) > S_b$
0 if $\min_t S(t) \leq S_b$

i.e. if the spot price falls below the barrier S_b at any point during the lifetime of the option then the payoff from the option is zero. The implementation of the down and out put option is very similar to that of the local volatility model for a put. The key difference being the use of the MATLAB function "heaviside" to ensure that the payoff from any path of the option which has dropped below the barrier is zero. A section from the function "MC_barrier_put_local_vol_naive" shows how this would be implemented:

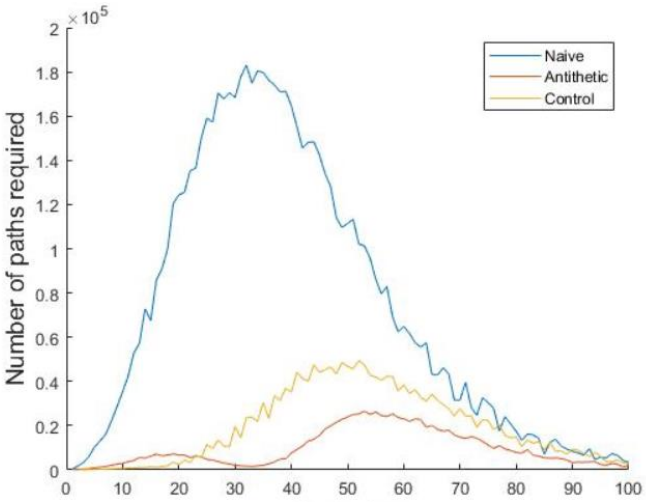

Figure 1.8: Simulations required to ensure a pricing error of less than £0.05 95% of the time, for a put option with a local volatility model. Output of function "MC_Put_Pricing_locvol".

```
%compute discount payoff:

fs = exp(-sum(rate(0.05, t(1,1:end))*dt))*heaviside(min(S , [], 2)...
- Sb).*max(K - S(:,end),0);
```

Here we take the minimum of the spot price for each of the paths. We then subtract that from the barrier price, if this is less than zero the "heaviside" function will set the discounted payoff from this path to zero.

2.3.2 pricing the down and out put (barrier):

The approach taken to determine the number of paths required to price the down and out put option with a barrier, S_b set at £30 with a pricing error of £0.05 95% of the time is identical to that discussed in part 2.3. The only difference is that the discounted payoffs for the different Monte Carlo techniques are adjusted via the "heaviside" function in a similar manner shown in 2.3.1. Option price reflects the fact that as the spot price exceeds the barrier price the option initially increases in value. This reflects the fact that as the spot increases it becomes increasingly unlikely that the option will hit the barrier. However, after a certain point the increase in value of the option caused by a decrease in the likelihood of the barrier being hit is outweighed by the likelihood of a higher spot price at expiration and so lower payoff. As such the option price then decreases as the spot price increases. In comparison to the put with a local volatility model we can tell there are some major changes in the performance of the various Monte Carlo methods. Specifically, the CPU times are now far quicker particularly for the naïve method. This could potentially be

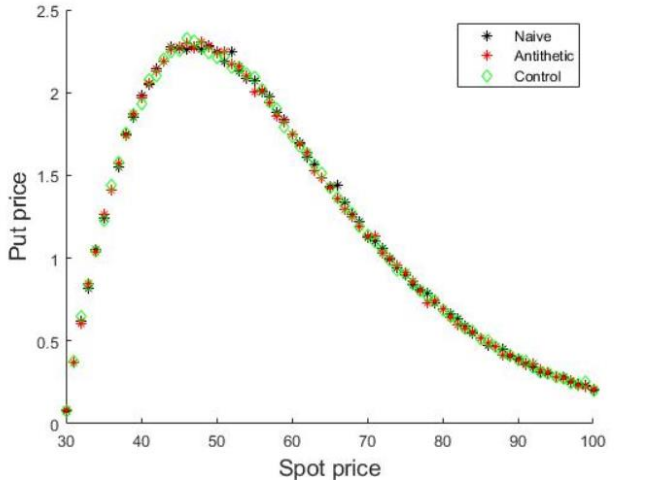

Figure 1.9: Price of the down and out put across a range of spot prices according to all. Output of function "MC_Put_Pricing_locvol_barrier". Note: x axis starts at spot price of 30 i.e. the barrier price.

	Naive	Antithetic	Control Variates
Paths required	3.30×10^4	1.65×10^4	3.17×10^4
CPU time	22.09	21.38	27.36

Table 4: Simulations required to ensure a pricing error of less than £0.05 95% of the time and the CPU times required across the the range of spot prices. Calculated for a put option with a local volatility model and a barrier at £30. Calculated using a computer with the following specifications: PROCESSOR: INTEL(R) CORE(TM) I5-7500 CPU @ 3.40GHZ 3.41 GHZ, RAM: 16.0 GB. These results come from the function "MC_Put_Pricing_locvol_barrier".

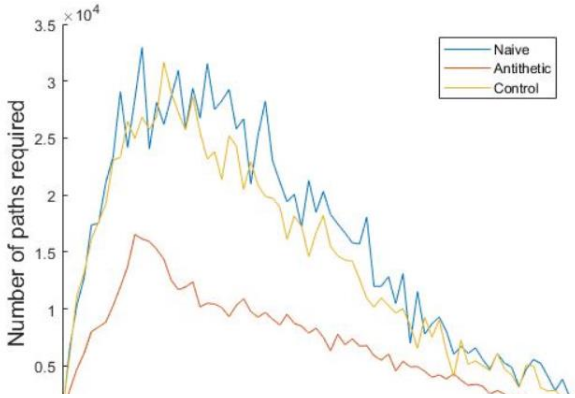

Figure 1.10: Simulations required to ensure a pricing error of less than £0.05 95% of the time, for a down and out put option with a local volatility model. Output of function "MC_Put_Pricing_locvol_barrier".

explained by the fact that for the barrier option the number of paths required to compute the option with a pricing error less than $0.05 95% of the time are many multitudes smaller than in the case of the put. This is explained by the fact that the barrier significantly decreases the variance of the put payoff as it sets a large proportion of the payoffs to zero that would have otherwise taken on non-discounted values between £30 and £50. This in turn bounds the payoff of the option between £0 and £30, vastly decreases the range of potential discounted payoffs. In addition, we are only interested in the payoff between the barrier (£30) and the end of the range of spot prices (£100). This in turn decreases the number of spot prices over which the prices have to be estimated and so vastly reduces the computational intensity. Another noticeable change is the vast decrease in the effectiveness of control variates as a variance reduction technique. This indicates that geometric Brownian motion with constant parameters is no longer strongly correlated with the payoff of the down and out put option. This intuitively makes sense as only the paths of the geometric motion which range between that of £30 and £50 will now be perfectly negatively correlated with the payoff of the option.

2.3.3 pricing the down and out put over a series range of barriers:

Figure 1.11 shows how put prices evolve depending on the value of the barrier we see that as the price of the barrier increases the maximum value of the put decreases. This makes sense from a financial perspective as the higher the value of the barrier the lower the maximum payoff from the option. With a barrier of zero we simply have a payoff exactly the same as that of a standard put option with a local volatility model. With a barrier of 50 the option has zero payoff as the barrier is now the same value as that of the strike. We see that for all the barriers the put price behaves in the same way described in 2.3.2 increasing initially as the spot price exceeds the barrier price and then decreasing as spot price increases.

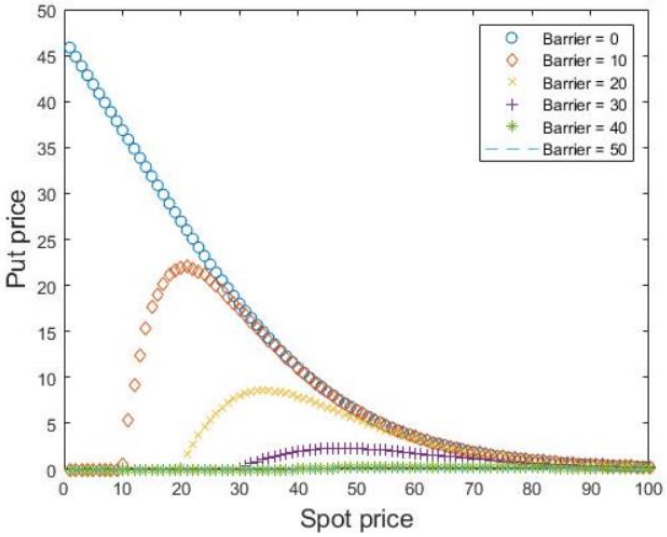

Figure 1.11: Price of a down and out put over a range of barriers using antithetic variance reduction and enough paths to ensure that there is a pricing error of less than £0.05 95% of the tiem. Output of function “MC_barrier_range _local_antithetic”.

2.3.4 Deltas of down and out put options over a range of barrier prices:

Implementing the calculation of the delta for a down and out put option using Monte Carlo methods with antithetic variance reduction is very similar to the approach taken when pricing the bull spread. The key differences being that we calculate the path of the underlying using Euler time stepping of one day. This rational for doing this is due to the payoff: $f((S(T)_{right})^{+})$, $f((S(T)_{left})^{+})$, $f((S(T)_{right})^{-})$, $f((S(T)_{left})^{-})$ being path dependent both because of the local volatility model and the barrier component. The financial intuition behind these deltas is as follows: initially as the spot price is less than the barrier the delta is zero as all the paths have zero payoff and an incremental increase in spot price will not affect this. As the option initially exceeds the barrier the value of the put increases greatly. This is reflected in a very high positive delta. This is because as the spot increases the put now has a far greater likelihood of not hitting the barrier and so this increases the value of the option. However, as the spot price continues to increase the increase in probability of not hitting the barrier, which increases the value of the option, begins to be outweighed by the decrease in value caused by the likelihood of a lower payoff at expiration due to a higher value of the spot at time T. This is reflected in a delta that decreases and then eventually becomes negative. Once the spot is sufficiently high the delta mimics that of a put without a barrier.

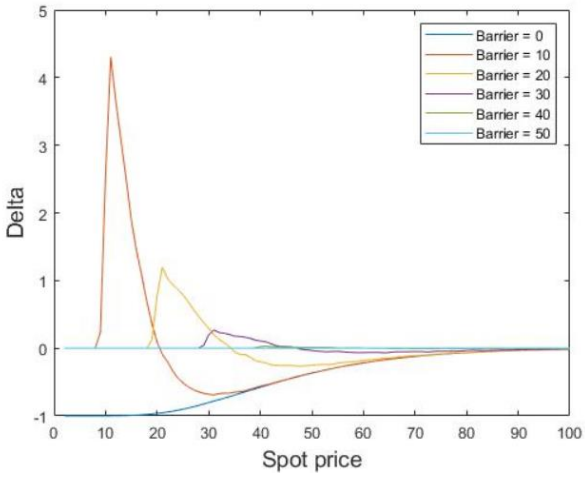

Figure 1.12: Plot of the deltas of a down and out put option over a range of barriers using antithetic variance reduction and with a sufficent number of simulations to ensure a pricing error of less £0.05 95% of the time. Output of function “MC_barrier_put _local_vol_antithetic_delta_Sbrange”.

Appendix:

Table & graph outputs:

This section gives further information as to where the various information presented in the various tables and graphs shown throughout the report are outputted from the code.

Part 1:

Note that as mentioned in the foreword all graphs and table outputs for part 1 of the report can be produced by running the function “Main_1”. Specific details of the variables that correspond to each output given in the tables throughout the report are given below:

	Naive method	Antithetic variance reduction	Control Variates	Importance sampling
Simulations required	N_max_naive	N_max_antithetic	N_max_control	N_max_impsampling
CPU time	CPU_naive	CPU_antithetic	CPU_control	CPU_impsampling

Table 1

	Naive method	Antithetic variance reduction	Control Variates	Importance sampling
Simulations required	N_max_naive_r	N_max_antithetic_r	N_max_control_r	N_max_impsampling_r
CPU time	CPU_naive_r	CPU_antithetic_r	CPU_control_r	CPU_impsampling_r

Table 2

Part 2:

Note that as mentioned in the foreword all graphs and table outputs for part 2 of the report can be produced by running the function “Main_2”. Specific details of the variables that correspond to each output given in the tables throughout the report are given below:

	Naive	Antithetic	Control Variates
Paths required	Npath_naive	Npath_antithetic	Npath_control
CPU time	CPU_naive	CPU_antithetic	CPU_control

Table 3

	Naive	Antithetic	Control Variates
Paths required	Npath_naive_b	Npath_antithetic_b	Npath_control_b
CPU time	CPU_naive_b	CPU_antithetic_b	CPU_control_b

Table 4

Functions:

Below is a list of all functions used with a brief description of their purpose:

Part 1:

Main_1 – produces all of the graphs and data used in part 1 of the project

Initalisation_A – Script to set the parameters needed for part 1 of the project, $K_1 = 90$, $K_2 = 120$, T = 0.5, r = 0.03, sigma = 0.25, Smin = 1, Smax = 200 and error = 0.05. Where error refers to the largest possible pricing error at a 95% confidence interval.

delta_bull_antithetic - Calculates the delta of a call bull spread using MC simulations and antithetic variance reduction. Also produces a plots of the deltas estimated vs true deltas and of the error in deltas estimated.

Delta_CD - Calculates the delta of the bull spread using a centred difference approach adjusted for irregularities in the grid. In addition, produces a plot of the deltas predicted by the centred difference approach and those produced by the Back-Scholes function. Also produces a plot of the error in the deltas produced.

Delta_FD - Calculates the delta of the bull spread using a one-sided difference approach (forward difference) adjusted for irregularities in the grid. In addition, produces a plot of the deltas predicted by the one-sided difference approach and those produced by the Back-Scholes function. Also produces a plot of the error in the deltas produced.

heat_CN - Solve heat equation d_t u = d_xx u by CN scheme with general Dirichlet boundary conditions.

MC_antithetic_EURO_CALL_bull - Price Euro call bull spread by Monte Carlo appraoch using antiithetic variance reduction.

MC_control_euro_call_bull - Price Euro call bull spread by the Monte Carlo method with variance reduction using the control variates approach.

MC_euro_call_bull - Price Euro call bull spread by naive Monte Carlo approach

MC_impsampling_euro_call_bull - Price Euro call bull spread by the Monte Carlo method with variance reduction using the important sampling approach.

MC_Pricing - prices a bull spread using Monte Carlo simulations and various variance reduction techniques. In addition, produces graphs of the required number of simulations to ensure a pricing error less than £0.05 for each approach. Also plots the solution to the bull spread for this given pricing error for each Monte Carlo method. NOTE: This approach does not use path recycling when computing the bull spread price for each spot price.

MC_Pricing_recycle - prices a bull spread using Monte Carlo simulations and various variance reduction techniques. In addition produces graphsof the required number of simulations to ensure a pricing error less than 0.05 for each approach. Also plots the solution to the bull spread for this given pricing error for each Monte Carlo method. NOTE: This approach allows for path recycling when computing the bull spread price for each spot price, i.e. the same N normally distributed random variables are used for the entire range of S.

Min_err_bullspread - determines the values of N & J to ensure that the numeric solution to a bull spread solved via the heat equation is within a given margin of error.

EURO_CALL_PDE - Pricing of Euro bull Spread via PDE (heat equation).

PDE_euro_call_bull_CPU - Calculates the CPU time taken to compute the PDE solution to the BS equation for a bull spread for a given N & J. Also produces a plot of the solution to the bull spread.

tridiag - solves a system of linear equations Ax=b where A is a tridiagonal matrix.

Part 2:

Main_1 – produces all of the graphs and data used in part 2 of the project

Initalisation_A – Script to set the parameters needed for part 2 of the project, K = 50, T = 1, r_0= 0.05, σ_0 = 0.3, Smin = 1, Smax = 100, err = 0.05, Sb = 30, Sbmin = 0 and Sbmax = 50. Where err refers to the largest possible pricing error at a 95% confidence interval.

MC_barrier_put_local_vol_antithetic - Uses Monte Carlo simulation to compute the price of a barrier put option with a local volatility model using antithetic variance reduction.

MC_barrier_put_local_vol_antithetic_delta - Computes the delta of a barrier put with a local volatility model using antithetic variance reduction.

MC_barrier_put_local_vol_antithetic_delta_Sbrange - Computes the delta for the barrier put with local volatility model using antithetic variance reduction. This is done over a range of barriers and is also plotted.

MC_barrier_put_local_vol_control - Uses Monte Carlo simulation to compute the price of a barrier put option with a local volatility model using control variates. The underlying price, that evolves according to geometric Brownian motion with drift with constant interest rate and volatility, is used as the control variate.

MC_barrier_put_local_vol_naive - Uses Monte Carlo simulation to compute the price of a barrier put option without any variance reduction.

MC_barrier_range_put_local_vol_antithetic - Computes the price of a barrier put with a local volatility model for a range of spot prices and a range of barrier prices. This will also produce a graph the price of these various put options with different barriers. Pricing is done using Monte Carlo simulations with antithetic variance reduction.

MC_const_vol_Put_CPU - computes the output of a constant volatility naïve Monte Carlo approach to pricing a European put option with pricing error less than err.

MC_const_vol_Put_Euler_CPU - computes the output of a constant volatility naive MC approach to pricing a European put option with pricing error less than err. This method uses Euler time stepping to generate a complete path of the underlying price.

MC_EURO_PUT - Price European put option by Monte Carlo approach

MC_euro_put_euler - Prices a European put option by the Monte Carlo with Euler time stepping i.e. generating the whole path for the underlying.

MC_Put_local_vol - Uses Monte Carlo simulation to compute the price of a put option with a local volatility model.

Uses Monte Carlo simulation to compute the price of a put option with a local volatility model using antithetic variance reduction.

MC_Put_local_vol_control - Uses Monte Carlo simulation to compute the price of a put option with a local volatility model using control variates as a variance reduction technique.

MC_Put_Pricing_locvol - computes for Monte Carlo simulations of a put with a local volatility model the number of paths required to ensure a pricing error less than the input "err". The price of the put for this corresponding number of paths over a give price range (Smin to Smax) and the CPU time required to compute these prices. This is done for the naive Monte Carlo method and with antithetic variance reduction and control variates. It will also produce a plot of the put prices for these various pricing methods and a plot of the number of paths needed to be simulated to ensure a pricing error less than "err".

MC_Put_Pricing_locvol_barrier - computes for Monte Carlo simulations of a barrier put with a local volatility model the number of paths required to ensure a pricing error of less than the input "err". The price of the put for this corresponding number of paths over a give price range (Smin to Smax) and the CPU time required to compute these prices. This is done for the naive Monte Carlo method and with antithetic and control variates. It will also produce a plot of the put prices for these various pricing methods and a plot of the number of paths needed to be simulated to ensure a pricing error less than "err".

rate - gives interest rate as a function of time, i.e. a non-constant interest rate across time

volatility - Returns the local volatility.