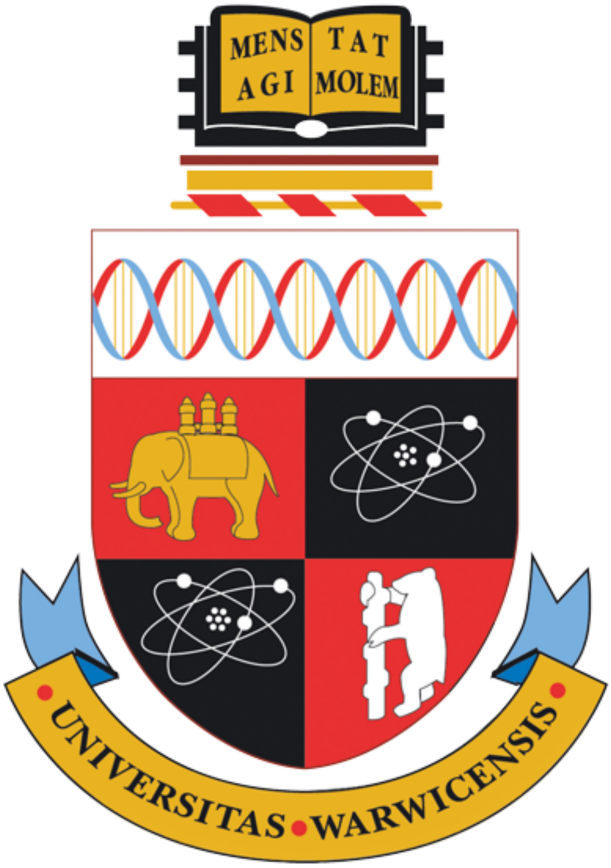


MA9070
Numerical Methods and Programming

Project on Numerical Methods for Option Pricing

Jón Sveinbjörn Halldórsson

1991391

December 11, 2019

Contents

1 Pricing a Vertical Spread

1.1 Transforming the Black-Scholes PDE to the Heat Equation

The Black-Scholes model is used to estimate the theoretical price of European options based on maturity, constant interest rates, constant volatility and the strike price of the option. The partial differential equation and it's conditions are as follows:

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

$$\text{BC}: \quad V(S,t) := e^{-r(T-t)}\big(\max(S-K_1,0) - \max(S-K_2,0)\big) \tag{2}$$

$$V(S_{\min},t) = \lim_{S\to 0^-} V(S,t) = e^{-r(T-t)}(0-0) = 0 \tag{3}$$

$$V(S_{\max},t) = \lim_{S\to\infty} V(S,t) = e^{-r(T-t)}\big(S-K_1-(S-K_2)\big) = \big(K_2-K_1\big)e^{-r(T-t)} \tag{4}$$

where $S > 0$ and $0 \le t \le T$.

We perform the following variable transformations to convert this into a boundary value problem for the heat equation:

$$S = \exp(x) \Rightarrow x = \log(S) \tag{5}$$

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

By substituting the partial derivatives from Appendix A into the Black-Scholes equation we get

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + \left(\frac{2r}{\sigma^2} - 1\right)\frac{\partial v}{\partial x} - \frac{2r}{\sigma^2}v \tag{7}$$

and we set $\kappa = \dfrac{2r}{\sigma^2}$ so the partial differential equation and it's boundary condition becomes

$$\text{PDE:} \quad \frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + (\kappa-1)\frac{\partial v}{\partial x} - \kappa v \tag{8}$$

$$\text{BC:} \quad v(x,\tau) = e^{-r\tau}\big(\max(\exp(x)-K_1,0) - \max(\exp(x)-K_2,0)\big) \tag{9}$$

where the domains of x and t have become $-\infty < x < \infty$ and $0 \le \tau \le \dfrac{\sigma^2}{2}T$ due to the transformations. To finish the transformation to the heat equation, we set

$$v(x,\tau) := \exp\big(\alpha x + \beta\tau\big)u(x,\tau) = \exp\big(-\frac{1}{2}(\kappa-1)x - \frac{1}{4}(\kappa+1)^2\tau\big)u(x,\tau)\big) \tag{10}$$

where α and β are determined in Appendix B such that we can write (7) as the heat equation. We finally end up with the following heat equation and conditions

$$\text{PDE:} \quad \frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2} \tag{11}$$

$$\text{IC:} \quad u(x,0) = \exp(-(\alpha x))\big(\max(\exp(x)-K_1,0) - \max(\exp(x)-K_2,0)\big) \tag{12}$$

$$\text{BC:} \quad u(x,\tau) = \exp(-(\alpha x+\beta\tau))\big(\max(\exp(x)-K_1,0) - \max(\exp(x)-K_2,0)\big) \tag{13}$$

$$u(x_{\max},\tau) = \exp(-(\alpha x_{\max}+\beta\tau))\big(K_2-K_1\big) \tag{14}$$

1.2 Solving the Black-Scholes Equation

After being transformed into the heat equation, the Black Scholes partial differential equation was solved using the Crank-Nicolson scheme with a tridiagonal matrix algorithm, `tridiag.m`.

The number of time steps was chosen as the number of trading days in the six month period, i.e., $N = 130$ and from there the number of grid points, J, was determined by iterating the heat equation solver, `PDE_bullspread.m`, in a while loop until the maximum absolute error was less than the desired amount, in this case

$$||V_{\text{PDE}} - V_{\text{BS}}||_\infty < £0.05 \tag{15}$$

where V_{PDE} is the numerical solution and V_{BS} is the Black-Scholes solution given by the `blsprice` function.

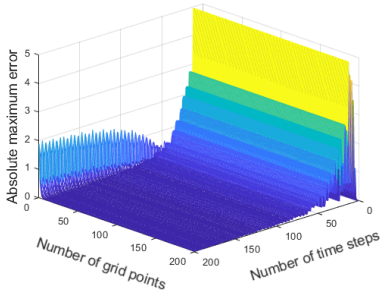


Figure 1.1: The maximum absolute error of the numerical solution as a function of the number of grid points and time steps, ranging from $[0, 200]$. Source: `ErrorSurfacePlot.m`.

From empirical evidence (Figure 1.1) we observed that multiple pairs of N and J fulfill this criteria and that the error was more sensitive to the number of step sizes than the number of grid points as they both tend to 0. The initial value of the number of grid points was chosen at random to be $J = N/10 = 13$ in the function `MinimizeAbsError.m` and 69 was the lowest number which fit the criteria.

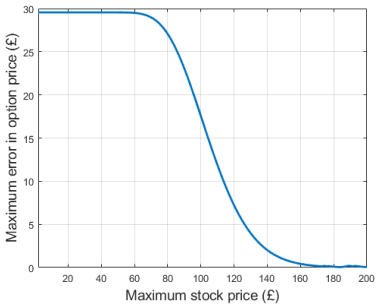


Figure 1.2: The maximum absolute error of the numerical solution as a function of the maximum stock price, ranging from £0 to £200. Source: `ErrorAnalysis.m`.

The heat equation solver was used to price a bull call spread with two strike prices $K_1 = £90$ and $K_2 = £120$, a time to maturity of 6 months, constant risk-free interest rate of $r = 3.00\%$ and assuming that the underlying asset follows a geometric Brownian motion,

$$dS(t) = rS(t)dt + \sigma S(t)dW(t) \tag{16}$$

with constant volatility, $\sigma = 25\%$. In order to get an accurate solution, the maximum stock price had to be large enough for $x = \log(S_{\max})$ to be sufficiently large. By running simulations in the function `ErrorAnalysis.m`, we observed that as $S_{\max}$ exceeds £180, the error term dwindles and tends to 0.

The CPU time was measured with `cputime` in MATLAB on a 2016 Dell Latitude E7470 with an Intel(R) Core(TM) i7-6600U CPU @ 2.60 GHz processor and 16.0 GB of RAM. To get a consistent non-zero measurement for the elapsed CPU time we priced the option 1000 times with the heat equation solver. The average elapsed CPU time was 0.0027s.

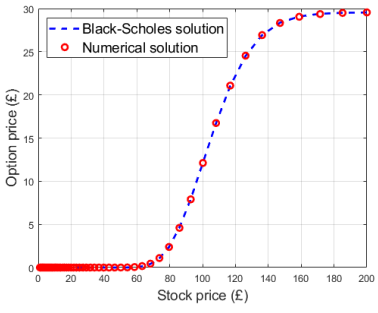


Figure 1.3: The numerical solution of the bull call price compared to the Black Scholes solution obtained by `blsprice.m`. Source: `PlotComparison.m`.

The option prices derived by the heat equation solver were accurate to £0.0499 in comparison to the Black Scholes solution and the shape of the option prices as a function of stock prices reflected the fact that as $S(T)$ tends to 0, the likelihood of the call options being exercised goes to zero, leading to no payoff, and as $S(T)$ tends to infinity the likelihood of both call options being exercised converges to 1, leading to a payoff at maturity of

$$S_T - K_1 - (S_T - K_2) = K_2 - K_1 \tag{17}$$

1.3 Computing the Delta

The delta of the option, i.e. the sensitivity to changes in the stock price, was calculated with the one-sided finite difference method

$$\Delta = \frac{\partial V}{\partial \theta} = \frac{V(\theta + \delta\theta) - V(\theta)}{\delta\theta} + \mathcal{O}(\delta\theta) \tag{18}$$

Due to the transformation, the stock prices were no longer uniformly distributed, so at each time the $\delta\theta$ was calculated as the difference between the stock prices and therefore

$$\Delta = \frac{\partial V}{\partial \theta} = \frac{V(S_{t+1}) - V(S_t)}{S_{t+1} - S_t} + \mathcal{O}(\delta\theta) \tag{19}$$

In comparison to the Black Scholes solution calculated with `blsdelta`, the numerical solution is a good fit. However, the curve seems to be shifted and has an maximum absolute error of $||\Delta_{\text{PDE}} - \Delta_{\text{BS}}||_\infty = 0.06925$ which is large in the context of an option's delta, considering that it only takes values $[0, 1]$ for a call option. The shape of the curve reflects the fact that as the option goes from being out-of-the-money it becomes very sensitive to changes in the stock price, and because the short call option with $K_2 = £120$ limits the upside potential of the trade to a payoff of $K_2 - K_1$ the sensitivity drastically decreases as the stock price exceeds £100.

1.4 Pricing the Bull Call Spread with Monte Carlo

For each of the following methods, we will simulate 1000 stock price paths for each initial stock price ranging from £1 to £200. We will work under the assumption that the underlying asset follows a geometric Brownian motion (16), which has the solution

$$S(T) = S(t) + dS = S(t) \exp\left((r - \frac{1}{2}\sigma^2)(T - t) + \sigma dW(t)\right) \tag{20}$$

where we assume constant annualised interest rates and volatility, $r = 3.00\%$ and $\sigma = 25\%$. We will consider a bull call spread with strike prices $K_1 = £90$ and $K_2 = £120$, a time of maturity, T, of six months. We will refer to the payoff of the bull call spread as f which is defined as

$$f(S(T)) = \max(S(T) - K_1, 0) - \max(S(T) - K_2, 0) \tag{21}$$

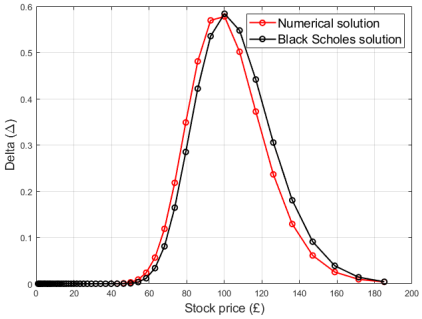


Figure 1.4: The numerical solution and the Black Scholes delta of the bull call spread as a function of stock price. Source: `FiniteDifferenceDelta.m` and `CompareDelta.m`

1.4.1 Naive Method

To simulate the stock price at maturity we assume a time step of T with $t = 0$ and generate $M = 1000$ normally distributed random variables $dW(t)$, i.e. Wiener processes, such that

$$dW(t) = \sqrt{dt}\mathcal{X}; \quad \mathcal{X} \sim \mathcal{N}(0, 1) \tag{22}$$

After simulating the stock price at maturity, the option was priced by taking the average discounted expected payoff at maturity

$$V(S(T)) = \exp(-rT)\,\mathbb{E}[f(S(T))] \tag{23}$$

1.4.2 Antithetic Variance Reduction

This method uses the the fact that for a normally distributed variable $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, the corresponding negated values, $-\mathcal{X} = \{-X_1, -X_2, \ldots, -X_n\}$, are also normally distributed. Therefore,

simulating twice as many stock prices at maturity, using both $\mathcal{X}$ and $-\mathcal{X}$ in the Wiener process to produce

$$S_i(T)^+ = S_i(t)\exp((r - \frac{1}{2}\sigma^2)(T-t) + \sigma dW(t)) \tag{24}$$

$$S_i(T)^- = S_i(t)\exp((r - \frac{1}{2}\sigma^2)(T-t) - \sigma dW(t)) \tag{25}$$

We then calculated the price of the option by taking the average price of the option, obtained for each initial stock price, $S_i(t)$

$$Z = \mathbb{E}[Z_i(S(T))] = \mathbb{E}\left[\frac{V(S_i(T)^+) + V(S_i(T)^-)}{2}\right] \tag{26}$$

Due to the negated values, $-\mathcal{X}$, the variance of $Z_i(S(t,T))$ decreases by at least twofold due to the fact that the covariance is negative and the two processes have the same variance. This accuracy comes at the cost of being twice as costly in computation.

$$\begin{aligned}
\mathrm{Var}[Z_i(S(T))] &= \frac{\mathrm{Var}[V(S_i(T)^+) + V(S_i(T)^-)]}{4} \\
&= \frac{\mathrm{Var}[V(S_i(T)^+)] + \mathrm{Var}[V(S_i(T)^-)] + 2\mathrm{Cov}(V(S_i(T)^+), V(S_i(T)^-))}{4} \\
&= \frac{\mathrm{Var}[V(S_i(T)^+)]}{2} + \frac{\mathrm{Cov}(V(S_i(T)^+), V(S_i(T)^-))}{2}
\end{aligned}$$

1.4.3 Control Variates

In this case, we wanted to make our sample mean an accurate approximation to the true mean of our payoff function. In order to accomplish this, we needed to find a function that was highly correlated (or anti-correlated) with the payoff of a bull call spread, $f(S(T))$, with a known mean and variance.

The function $g(S(T)) = S(T)$ sufficed for our strategy when looking at a range of stock prices which is near both strike prices. In this case we were looking at values of $S(T)$ from £1 to £200 with strike prices at $K_1 = £90$ and $K_2 = £120$. To increase the correlation, we could have either valued the price of the option on a range closer to the strike prices or found another function g with a higher correlation.

Since we determined $g(S(T)) = S(T)$, i.e., the geometric Brownian motion, we knew the mean and variance

$$g_m := \mathbb{E}[S(T)] = S(t)\exp(rT) \tag{27}$$

$$g_v := \mathrm{Var}[S(T)] = \big(S(t)\exp(rT)\big)^2\big(\exp(\sigma^2 T) - 1\big) = g_m^2\big(\exp(\sigma^2 T) - 1\big) \tag{28}$$

Our control variate was defined as

$$f_c = f - c(g - g_m) \tag{29}$$

$$\mathbb{E}[f_c] = \mathbb{E}[f] - \mathbb{E}[c(g - g_m)] = \mathbb{E}[f] - c(\mathbb{E}[g] - g_m) = \mathbb{E}[f] \tag{30}$$

$$\mathrm{Var}[f_c] = \mathrm{Var}[f] - \mathrm{Var}[c(g - g_m)] = \mathrm{Var}[f] - 2c\mathrm{cov}[f,g] + c^2\mathrm{Var}[g] \tag{31}$$

we then minimised the variance by differentiating with respect to c, yielding the optimal constant c as

$$c = \frac{\mathrm{cov}[f,g]}{\mathrm{Var}[g]} \tag{32}$$

Substituting the optimal c into (29) gives the variance of the payoff as $\mathrm{Var}[f]\big(1 - \mathrm{corr}[f,g]^2\big)$, indicating that higher correlation (or anti-correlation) results in lower variance.

1.4.4 Importance Sampling

As the name of this method suggests, the main idea here is to draw samples from stock prices where the corresponding option price is deemed important, i.e., non-zero. Due to the nature of the bull call spread as shown above, it's payoff is £0 when neither of the call options are exercised, i.e. $S(T) < K_1$, so we will consider the range $S(T) \geq$ £90 as important.

We note that the following are equivalent, $\mathcal{X} \sim \mathcal{N}(0,1)$ and $\Phi^{-1}(Y)$, where $Y \sim \mathcal{U}[0,1]$ and Φ is the standard normal cumulative distribution. Now, the stock price and the corresponding option price can be written as

$$S(T) = S(t)\exp\left((r - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T-t}\Phi^{-1}(Y)\right) \tag{33}$$

$$V(S(T)) = \mathbb{E}[S(T)] = \int_0^1 f(S(T,y))p_1(y)dy = \int_{y_0}^1 f(S(T,y))p_1(y)dy \tag{34}$$

where $p_1(y)$ is used to show that we are integrating with respect to the uniform distribution on the $[0,1]$. The two integrals are equal for y_0 defined by $S(T,y_0) = K_1$. We define another probability density function, p_2, as a uniform distribution on $[y_0,1]$, i.e.,

$$p_2(y) = \frac{1}{1-y_0} \tag{35}$$

and we find y_0 by solving (33) for $S(T) = K_1$

$$K_1 = S(t)\exp\left((r - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T-t}\Phi^{-1}(y_0)\right) \tag{36}$$

$$\Rightarrow y_0 = \Phi\left(\frac{\log\left(\frac{K_1}{S(t)}\right) - (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}\right) \tag{37}$$

where Φ is the cumulative distribution function for the standard normal distribution. Calculating this using `norminv` and `normcdf` in MATLAB for small values caused the payoff to jump to infinity. This was mended by setting these values were all set to £0. We can now write the payoff of the option as

$$V(S(t)) = (1-y_0)\int_{y_0}^1 f(S(T),y)p_2(y)dy \tag{38}$$

and therefore the price of the option price is estimated with

$$V(S(T)) = (1-y_0)\exp(-rT)\frac{1}{N}\sum_{i=1}^{N} f(S_i(T)) \tag{39}$$

where N is the number of stock prices.

1.4.5 Monte Carlo Computation of the Delta

For the delta calculations, we used a combination of path recycling and antithetic variance reduction. We considered the centred finite difference approach for computing the delta, i.e.,

$$\frac{\partial V}{\partial S}(S_t) = \frac{V(t+1) - V(t-1)}{2\Delta S} + \mathcal{O}(\Delta S^2) \tag{40}$$

Where $\Delta S = S_{t+1} - S_t$ is constant for all values of t, i.e. the stock prices are uniformly distributed. Taking the variance of both sides yielded

$$\mathrm{Var}\left[\frac{V(t+1) - V(t-1)}{2\Delta S}\right] \approx \mathrm{Var}\left[\frac{\partial V}{\partial S}S_t\right] \tag{41}$$

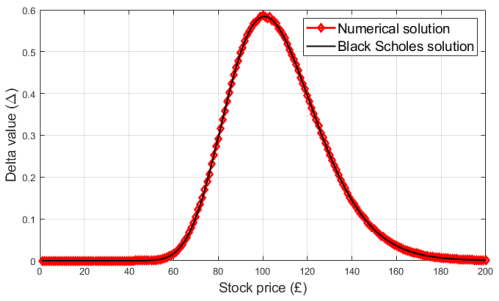


Figure 1.5: Comparison of the Monte Carlo delta and the Black Scholes delta for each stock price. Source: `AntitheticDelta.m` and `PlotDelta.m`

Since we wanted to reduce the variance of our delta, we used path recycling to minimise the variance of our left hand side. We set $dS = £1$ and simulated the following stocks paths

$$S_R^+(T) = (S_0 + dS)\exp((r - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}X)$$
$$S_R^-(T) = (S_0 + dS)\exp((r - \frac{1}{2}\sigma^2)T - \sigma\sqrt{T}X)$$
$$S_L^+(T) = (S_0 - dS)\exp((r - \frac{1}{2}\sigma^2)T + \sigma\sqrt{T}X)$$
$$S_L^-(T) = (S_0 - dS)\exp((r - \frac{1}{2}\sigma^2)T - \sigma\sqrt{T}X)$$

where we use left and right (L and R) to mean a shift of the initial price to either $S_0 - dS$ or $S_0 + dS$ respectively, and by the $\pm$ we mean that we take the positive or negative part of the random process respectively. We then took the difference between the payoff of the left and right option to obtain the delta as

$$f(S_R) = \frac{f(S_R^+(T)) + f(S_R^-(T))}{2}$$
$$f(S_L) = \frac{f(S_L^+(T)) + f(S_L^-(T))}{2}$$
$$V_R = \mathbb{E}[f(S_R)]$$
$$V_L = \mathbb{E}[f(S_L)]$$
$$\Delta = \frac{V_R - V_L}{2}$$

where f is the option's payoff. When compared to the Black Scholes delta, i.e. `blsdelta`, the error was $||\Delta_{\text{AT}} - \Delta_{\text{BS}}||_\infty = 0.0038$, much lower than the one calculated with the one-sided finite difference method.

1.4.6 Comparison of the Monte Carlo Methods

To calculate the accuracy of the simulated prices to within $£0.05$ with a 95% confidence level, we required at least N simulations, where N was determined by

$$||V_{\text{MC}} - V||_\infty = £0.05 \geq 1.96\frac{\sigma_{\text{MC}}}{\sqrt{N}}$$
$$\Rightarrow N \geq \left(\frac{1.96\sigma_{\text{MC}}}{0.05}\right)^2$$

where σ_{MC} was measured as the maximum variance after an initial run of 1000 simulations and 1.96 is the approximate value of the 97.5 percentile point of the normal distribution.

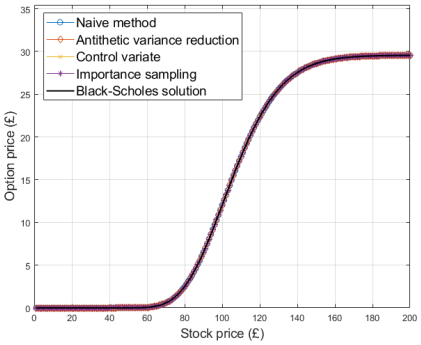


Figure 1.6: Comparison of the Monte Carlo prices for the bull call spread and the Black Scholes price as a function of stock prices. Source: `MonteCarlo.m` and `PlotMonteCarlo.m`

To compare the accuracy and computational cost of the four methods, we obtained the number of simulations required to achieve the specified accuracy for each stock price ranging from $£1$ to $£200$ by first running 1000 simulations. We were only interested in the maximum number of sample sizes required to satisfy the given error criteria over the whole range of stock prices, as it would ensure that the solution is accurate for the entire range. We then ran the Monte Carlo simulations again for each method using the corresponding sample sizes to get the desired accuracy and measured the CPU time with `cputime`.

Table 1: Comparison of the maximum variance, sample size needed, maximum error, and the average CPU time per simulation for the four Monte Carlo methods used to price a bull call spread. These results were obtained by running each Monte Carlo method for the maximum sample size required. Source: `MonteCarlo.m` and `PrintResults.m`.

	Naive method	Antithetic variance	Control Variate	Importance sampling
Maximum variance:	130.79	23.64	35.97	71.81
Maximum sample size:	2.0×10^5	3.6×10^4	5.5×10^4	1.1×10^5
Average CPU time:	0.9×10^{-6}	1.8×10^{-6}	1.9×10^{-6}	0.1×10^{-6}
Maximum error:	0.0878	0.0502	0.0461	0.0570

As may have been expected, the naive method performed the worst, requiring a sample size of 200,986 to give an 95% accurate result within £0.05 for every single stock price and taking the longest to compute, even though each simulation was on average one of the fastest.

The antithetic variance reduction was expected to decrease the variance by at least a half but vastly exceeded expectations. We observed that each simulation took on average twice as long as the naive method, which was to be expected. In a range of stock prices between the two strike prices, the covariance of the two call options almost perfectly canceled out the variance of the payoff.

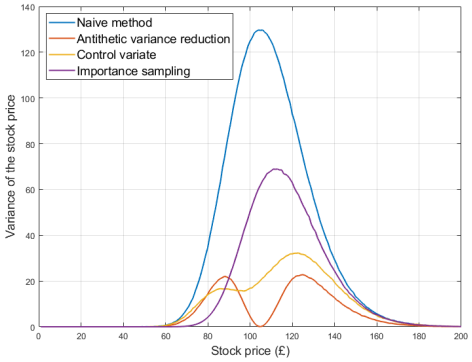


Figure 1.7: Comparison of the variance for the bull call spread and the Black Scholes price as a function of stock prices. Source: `MonteCarlo.m` and `PlotMonteCarlo.m`

The control variates method performed the slowest and did not quite achieve the same variance reduction as the antithetic method, especially between the strike prices. The inferiority in computational speed might be due to the fact that it has to calculate the covariance matrix.

Importance sampling failed to impress in every aspect other than elapsed CPU time, it outperformed the naive method overall but had inferior accuracy in comparison to the other two methods.

In comparison to the heat equation solver, the Monte Carlo methods were on average faster by a factor of 10 or more but required a multitude of simulations and therefore took longer to compute.

2 Pricing a Barrier Option

2.1 Local Volatility Model

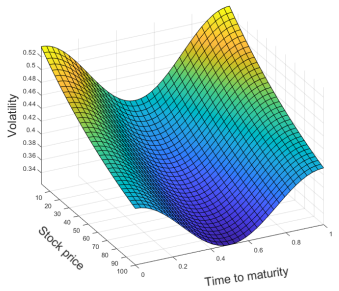


Figure 2.1: A surface plot of the local volatility model ranging from £1 to £100 and a time to maturity of up to 1 year. Source: `PlotVolatility.m`.

For this section, we simulate an asset which follows the geometric Brownian motion with the following local volatility and time-dependent interest rate dynamics:

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

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

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

where the constants are $r_0 = 0.05$, $r_1 = 0.5$, $\sigma_0 = 0.3$, $\sigma_1 = 0.12$, and $\sigma_2 = 0.6$. Both the time-dependent interest rates and the local volatility model were implemented in MATLAB with anonymous functions, i.e., with `volatility = @(S,t) sigma(1)*(1+sigma(2)*cos(2*pi*t))*(1+sigma(3)*exp(-S/100))` and `rate = @(t) r(1)*exp(r(2)*t)` where `sigma` and `r` are arrays of the constants.

2.2 Pricing a European Put Option With Monte Carlo

For each of the following methods, we will consider a strike price of $K = £50$, a time to maturity of 1 year. Additionally, we used the Euler time stepping with a time step of one day for a year, or 260 working days. For each day, the stock price was simulated with equation (43) and using

$$S(t_{i+1}) = S(t_i) + dS(t_i) \tag{45}$$

To begin with, we use these dynamics to price a vanilla put option with a payoff of $\max(K-S,0)$, i.e., with a price of

$$P(t,T) = \exp\left(-\int_t^T r(v)dv\right)\max(K-S(T),0) \quad \text{(Time-dependent interest rates)} \tag{46}$$

We then consider the European put option again but this time with constant interest rates and volatility. We were able to implement this part with the same MATLAB functions as for the local volatility model by having an if-loop which simulated the stock prices with (43) if `rate` and `volatility` were function handles and (16) if they were constants.

2.2.1 Naive method

The naive method for the European put option was similar to the one used when pricing the bull call spread, but instead of following equation (16) and simulating the entire time to maturity in one time step, we simulate the stock price for each day using equations (42) and (45). Another equivalent method would have been to use equation (16) with a time step of $dt = 1/260$. The payoff is then calculated by taking the intrinsic value at maturity, i.e. $\max(K-S(T),0)$ and the option was then priced as the discounted average payoff at maturity.

2.2.2 Antithetic Variance Reduction

In this method, calculating the option price followed the exact same algorithm. The only thing that changed was how the stock price path was simulated with the Euler time stepping, but since we only considered the value of the stock price at maturity, nothing changed other than the stock paths.

2.2.3 Control Variates

As stated in the first chapter, we wanted to find a function, g, which was highly correlated (or anti-correlated) with the option's payoff and whose mean and variance are known. We keep the same function as for the bull call spread, i.e., $g(S(T)) = S(T)$ and acknowledged that for the put (and later the down-and-out put) option, it will be anti-correlated with the payoff. Calculating the mean and variance of the geometric Brownian motion with a local volatility model and time-dependent interest rates, i.e. (40), would require lengthy stochastic calculations which fall outside the scope of this course. Therefore, we assumed constant parameters for the sake of this exercise.

$$dS_{\text{const}} = r_0 S_{\text{const}} dt + \sigma_0 S_{\text{const}} dW(t) \tag{47}$$

The mean and variance of the equation above are

$$g_m := \mathbb{E}[S(T)] = S(t) \exp\left(\int_t^T r_0 dv \right) = S(t) \exp\big(r_0(T-t)\big) \tag{48}$$

$$g_v := \text{Var}[S(T)] = \big(S(t) \exp\big(r_0(T-t)\big)\big)^2 \big(\exp(\sigma_0^2 T) - 1\big) = g_m^2 \big(\exp(\sigma_0^2 T) - 1\big) \tag{49}$$

After determining g_m and g_v, the option price was determined in the exact same way as for the bull call spread. To determine the goodness of fit of our assumption, we simulated both (42) and (47) 100,000 times in `DistributionComparison.m` and compared the distribution of the stock prices at maturity. We observed that our approximation follows a normal distribution, $S_{\text{const}} \sim \mathcal{N}(42.05, 11.98)$, whereas the actual geometric Brownian motion with the local volatility and time-dependent interest rates follows a log-normal distribution, i.e. $\log(S(T)) \sim \mathcal{N}(3.66, 0.43)$. From this comparison, we observe that the difference between the two distributions is discernible but the overall mean and variance is not far off so this should give a good approximation.

2.2.4 Comparison of the Monte Carlo Methods

The three methods were compared in the same way as before except this time we did not compare the prices with the Black Scholes formulas since they make different assumptions about the underlying asset, such as them having constant interest rates and volatility. For each of the three different Monte Carlo simulations, we first ran 1000 simulations to determine the sample size necessary for the desired accuracy. We then proceeded to simulate the European put option with each method and its corresponding maximum sample size. This process was then repeated for a put option with constant volatility and interest rates.

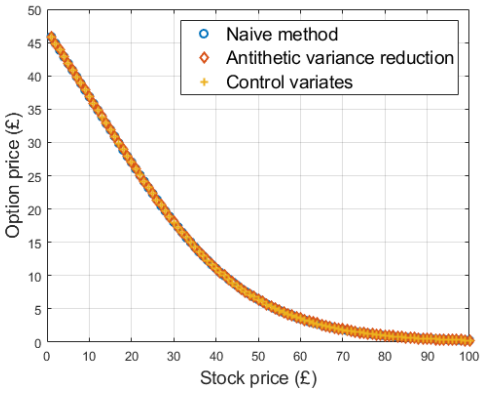
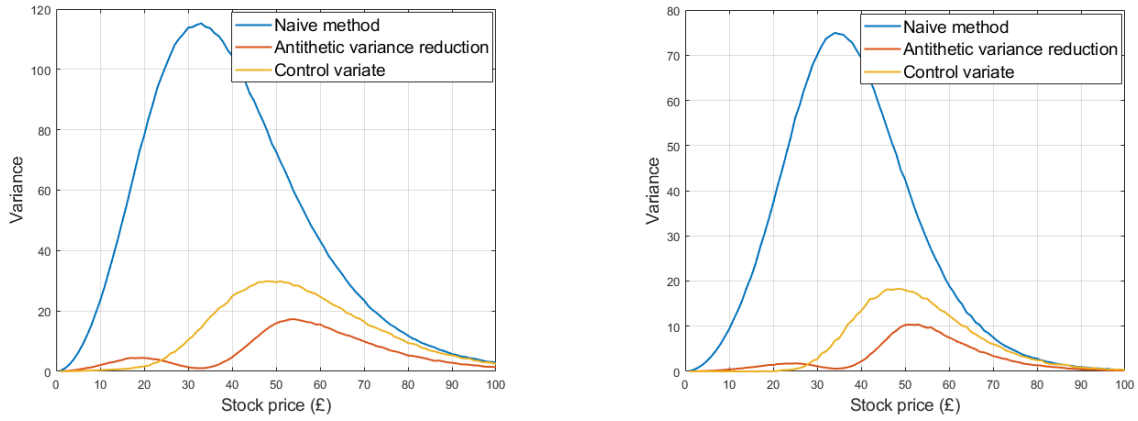


Figure 2.2: The price of a European put option with a strike $K = £50$ obtained by running each Monte Carlo with a 95% accuracy within £0.05 of the true price on the range of stock prices from £1 to £. Source: `MonteCarlo.m` and `PlotMonteCarlo.m`.

The Monte Carlo methods using the local volatility and time-dependent interest rates have to perform 12 calculations for every call to the `volatility` function and three for every call to the `rate`, instead of just one for each in the constant case. In every other aspect, the methods are the same, so we would expect the constant volatility and interest rate simulation to be faster.

The shape of the option's price as a function of the stock price reflects the fact that as S tends to 0, the payoff will at most be the strike price, i.e. $K = £50$, and as the stock prices exceeds the strike price, i.e. becomes out-of-the-money, the likelihood of a payoff goes to zero and therefore the option will be cheap for these stock prices.



(a) Local volatility and time-dependent interest rates

(b) Constant volatility and interest rates

Figure 2.3: A comparison of the variances as a function of stock prices obtained by running the Monte Carlo methods under the two assumptions for volatility and interest rates. Source: `MonteCarlo.m` and `PlotVariances.m`.

Table 2: Comparison of the maximum variance, sample size needed, and the average CPU time per simulation for the Monte Carlo methods used to price a European put option with local volatility and time-dependent interest rates. These results were obtained by running each Monte Carlo method for the maximum sample size required. Source: `MonteCarlo.m` amd `PrintResults.m`.

	Naive method	Antithetic variance	Control Variate
Maximum variance:	115.79	18.26	29.88
Maximum sample size:	1.8×10^{5}	2.6×10^{4}	4.5×10^{4}
Average CPU time:	1.6×10^{-5}	1.6×10^{-5}	0.9×10^{-5}

Table 3: Comparison of the maximum variance, sample size needed, and the average CPU time per simulation for the Monte Carlo methods used to price a European put option with constant volatility and interest rates. These results were obtained by running each Monte Carlo method for the maximum sample size required. Source: `MonteCarlo.m` amd `PrintResults.m`.

	Naive method	Antithetic variance	Control Variate
Maximum variance:	75.12	10.53	18.21
Maximum sample size:	1.1×10^{5}	1.6×10^{4}	2.7×10^{4}
Average CPU time:	0.5×10^{-5}	0.7×10^{-5}	0.7×10^{-5}

The constant volatility and interest rate simulations were consistently faster on average for all three methods, had a lower maximum variance and therefore required fewer simulations to obtain the desired accuracy.

2.3 Pricing a Down-and-Out Put Option With Monte Carlo

A down-and-out put option, unlike the vanilla put option, is path dependent and has the following payoff dynamics

$$V(S(t,T)) = \begin{cases} \max(K - S(T), 0) & \text{if } \min_t S(t) > S_b \\ 0 & \text{if } \min_t S(t) \leq S_b \end{cases} \tag{50}$$

where S_b is a barrier price. By implementing the option payoff in MATLAB with anonymous functions, i.e. the put payoff as a function of S, `put_payoff = @(S) max(K-S(:,end))`, and the down-and-out barrier put option as a function of S and S_b, `barrier_payoff = @(S, Sb) max(K - S(:,end),0).*(min(S,[],2) > Sb)`, we were able to make the code from the put option work for any type of barrier option as well by providing an optional input for the barrier price into the function for each method. The addition to the barrier payoff was a Boolean operator which returns 1 if the minimum stock price of the path was above the barrier, leading to a put payoff, and otherwise wiping out any payoff.

Once again, we start of by running the Monte Carlo methods 1000 times to determine the sample size needed for the desired accuracy. We then proceeded to price the option again with the relevant maximum sample size for each method. We considered a down-and-out put option with a strike of $K = £50$ and a barrier of $S_b = £30$. The option's price as a function of stock price reflects the fact that as the stock price exceeds the barrier, i.e. making it less likely for the stock path to go under the barrier, the down-and-out put option quickly becomes valuable and then it slowly decreases in value as the stock price approaches the strike price.

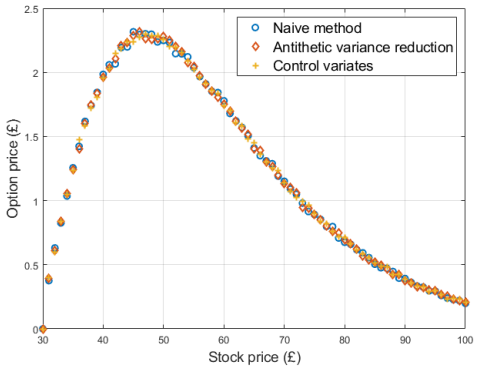


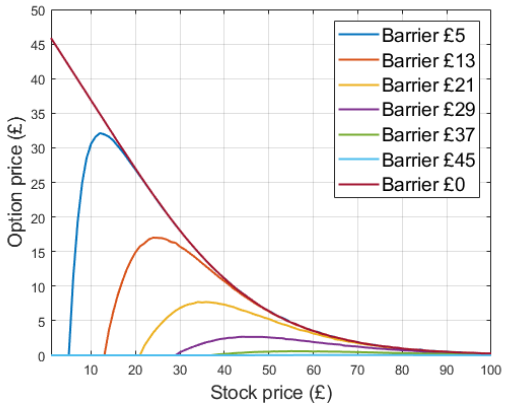
Figure 2.4: The Monte Carlo prices of a down-and-out put option with a strike $K = £50$ and barrier $S_b = £30$, 95% accurate to within $£0.05$ of the true price on the range of stock prices from $£1$ to $£$. Source: `MonteCarlo.m` and `PlotMonteCarlo.m`.

We then priced the down-and-out put option for a few different barrier prices, ranging from $S_b = £0$ to $S_b = £45$, for the maximum sample size needed using the antithetic variance reduction method. We observed that the barrier option with a barrier price of $S_b = £0$ behaves like a vanilla put option, as expected from the definition, and we refrained from using $S_b = K$ as a barrier price as that would not produce a payoff for any stock price.

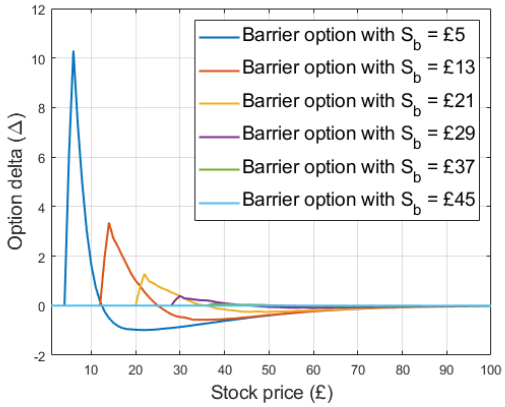
We notice that the Monte Carlo methods for the barrier option had a lower variance due to being bounded by the barrier and the strike, took longer on average due to the payoff being path dependent and needed fewer sample sizes to get the desired accuracy. The control variate method only slightly outperformed the naive method for the barrier option, leading us to believe that our choice of g is not correlated (or anti-correlated) enough with the payoff to reduce the variance.

Table 4: Comparison of the maximum variance, sample size needed, and the average CPU time per simulation for the Monte Carlo methods used to price a down-and-out put option with strike $K = £50$ and a barrier $S_b = £30$ with local volatility and time-dependent interest rates. These results were obtained by running each Monte Carlo method for the maximum sample size required. Source: `MonteCarlo.m` and `PrintResults.m`.

	Naive method	Antithetic variance	Control Variate
Maximum variance:	19.59	10.23	18.45
Maximum sample size:	3.0×10^4	1.6×10^4	2.8×10^4
Average CPU time:	0.9×10^{-5}	1.8×10^{-5}	1.0×10^{-5}



(a) Price of the barrier option with different barriers.



(b) Delta of the barrier option with different barriers.

Figure 2.5: A side-by-side comparison of the down-and-out put option price as a function of stock prices obtained by running the Monte Carlo methods and the corresponding deltas. Source: `MonteCarlo.m`, `BarrierComparison.m` and `BarrierDeltaComparison.m`.

2.3.1 Monte Carlo Computation of the Delta

The same methodology was used to calculate the delta of the down-and-out put option as for the bull call spread in chapter 1. This time, we had to simulate the stock path using Euler time stepping of one day and take into account the fact that the payoff is path-dependent. We used the maximum sample size needed for the desired accuracy in the antithetic variance reduction method to obtain the deltas for the same barrier prices as discussed above.

The barrier option's delta reflects that it's payoff suddenly becomes highly sensitive to changes in the stock prices as the stock price exceeds the barrier price and then for higher stock prices, the barrier property becomes less relevant and the option goes back to behaving like a put option, i.e. having $\Delta \in [-1, 0]$.

Appendix A Differentials for the Black-Scholes Transformation

We have the following variable transformations

$$S = \exp(x) \qquad t = T - \frac{2\tau}{\sigma^2} \qquad V(S,t) := v(x,\tau) = \nu\bigg(\log(S), \frac{\sigma^2}{2}(T-t)\bigg)$$

so the partial derivatives are calculated as follows using the chain rule:

$$\frac{\partial V}{\partial t} = \frac{\partial v}{\partial \tau} \cdot \frac{\partial \tau}{\partial t} = \frac{\partial v}{\partial \tau}\bigg(\frac{\partial t}{\partial \tau}\bigg)^{-1} = \frac{\partial v}{\partial \tau}\bigg(-\frac{2}{\sigma^2}\bigg)^{-1} = -\frac{\sigma^2}{2}\frac{\partial v}{\partial \tau} \tag{A.1}$$

$$\frac{\partial V}{\partial S} = \frac{\partial v}{\partial S} = \frac{\partial v}{\partial x} \cdot \frac{\partial x}{\partial S} = \frac{\partial v}{\partial x}\bigg(\frac{\partial}{\partial S}\log(S)\bigg) = \frac{1}{S}\frac{\partial v}{\partial x} \tag{A.2}$$

$$\frac{\partial^2 V}{S^2} = \frac{\partial}{\partial S}\bigg(\frac{\partial x}{\partial S}\frac{\partial V}{\partial S}\bigg) = \frac{\partial^2 V}{\partial x^2}\bigg(\frac{\partial x}{\partial}\bigg)^2 + \frac{\partial V}{\partial x} \cdot \frac{\partial x^2}{\partial S^2} = \frac{1}{S^2}\bigg(\frac{\partial^2 v}{\partial x^2} - \frac{\partial v}{\partial x}\bigg) \tag{A.3}$$

We substitute these expressions into the Black-Scholes

$$\begin{aligned}
&\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \\
\Leftrightarrow &-\frac{\sigma^2}{2}\frac{\partial v}{\partial \tau} + \frac{1}{2}\sigma^2 S^2\bigg(\frac{1}{S^2}\frac{\partial^2 v}{\partial x^2}\bigg) + rS\bigg(\frac{1}{S}\frac{\partial v}{\partial x}\bigg) - rv = 0 \\
\Leftrightarrow &-\frac{\sigma^2}{2}\frac{\partial v}{\partial \tau} + \frac{1}{2}\sigma^2\bigg(\frac{\partial^2 v}{\partial x^2}\bigg) + r\bigg(\frac{\partial v}{\partial x}\bigg) - rv = 0
\end{aligned}$$

and finally, rearranging the terms gives

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + \bigg(\frac{2r}{\sigma^2} - 1\bigg)\frac{\partial v}{\partial x} - \frac{2r}{\sigma^2}v \tag{A.4}$$

Appendix B Determining α and β for the Heat Equation

We have transformed the Black-Scholes partial differential equation to

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + (\kappa - 1)\frac{\partial v}{\partial x} - \kappa v \tag{B.1}$$

and we want to reduce it to the heat equation

$$\frac{\partial u}{\partial t} = c\frac{\partial^2 u}{\partial x^2} \tag{B.2}$$

Therefore, we set

$$v(x,\tau) := \exp(\alpha x + \beta\tau)u(x,\tau) \tag{B.3}$$

and calculate the partial derivatives

$$\begin{aligned}
\frac{\partial v}{\partial \tau} &= \frac{\partial}{\partial \tau}\big(\exp(\alpha x + \beta\tau)u(x,\tau)\big) = \beta\exp(\alpha x + \beta\tau)u(x,\tau) + \exp(\alpha x + \beta\tau)\frac{\partial}{\partial \tau}\big(u(x,\tau)\big) \\
\frac{\partial v}{\partial x} &= \frac{\partial}{\partial x}\big(\exp(\alpha x + \beta\tau)u(x,\tau)\big) = \alpha\exp(\alpha x + \beta\tau)u(x,\tau) + \exp(\alpha x + \beta\tau)\frac{\partial}{\partial x}\big(u(x,\tau)\big) \\
\frac{\partial^2 v}{\partial x^2} &= \frac{\partial}{\partial x}\big(\alpha\exp(\alpha x + \beta\tau)u(x,\tau) + \exp(\alpha x + \beta\tau)\frac{\partial}{\partial x}\big(u(x,\tau)\big) \\
&= \alpha^2\exp(\alpha x + \beta\tau)u(x,\tau) + 2\alpha\exp(\alpha x + \beta\tau)\frac{\partial}{\partial x}\big(u(x,\tau)\big) + \exp(\alpha x + \beta\tau)\frac{\partial^2}{\partial x^2}\big(u(x,\tau)\big)
\end{aligned}$$

By setting $\Psi := \exp(\alpha x + \beta\tau)$ and denoting $u(x,\tau) = \gamma$, we substitute the expressions into (B.1)

$$\beta\Psi\gamma + \Psi\frac{\partial\gamma}{\partial\tau} = \alpha^2\Psi\gamma + 2\alpha\Psi\frac{\partial\gamma}{\partial x} + \Psi\frac{\partial^2\gamma}{\partial x^2} + (\kappa-1)\big(\alpha\Psi\gamma + \Psi\frac{\partial\gamma}{\partial x}\big) - \kappa\Psi\gamma$$
$$\Psi\frac{\partial\gamma}{\partial\tau} = \Psi\frac{\partial^2\gamma}{\partial x^2} + \Psi\frac{\partial\gamma}{\partial x}\big(2\alpha + (\kappa-1)\big) + \Psi\gamma(\alpha^2 - \beta - \kappa)$$

We now have the following equation and compare the coefficients to determine α and β

$$\frac{\partial\gamma}{\partial\tau} = \frac{\partial^2\gamma}{\partial x^2} + \frac{\partial\gamma}{\partial x}\big(2\alpha + (\kappa-1)\big) + \gamma(\alpha^2 - \beta - \kappa) \tag{B.4}$$
$$2\alpha + (\kappa-1) = 0 \Leftrightarrow \alpha = -\frac{1}{2}(\kappa-1) \tag{B.5}$$
$$\alpha^2 - \beta - \kappa = 0 \Leftrightarrow \frac{1}{4}(\kappa-1)^2 - \beta - \kappa = 0 \Leftrightarrow \beta = -\frac{1}{4}(\kappa+1)^2 \tag{B.6}$$

Appendix C MATLAB Code

The code was written with reusability in mind. Most functions relating to either the option's payoff, the delta, the interest rates or the volatility were written such that they should work for any payoff, with a barrier or without, constant or local volatility and constant or time-dependent interest rates, provided that they are inserted into the functions as a function handle.

Many of the functions from Part I were also used in Part II so some functions may be identical. We described the functions from both parts for completeness.

C.1 Part I

- `main.m`: Script to run the project. Each question is split into a section which can be run independently.
- `SetParameters.m`: Initialises the parameters used in part I.
- `MinimizeAbsError(K1, K2, T, rate, sigma, Smin, Smax, N, option_price)`: Finds the optimal J such that our PDE solution is accurate to within £0.05 with a 95% confidence level given our initial N = 130.
- `PDE_bullspread(K1, K2, T, r, sigma, Smin, Smax, N, J)`: Solves the heat equation for a set of parameters and returns the option price as a function of stock prices. Uses `heat_CN.m` and `tridiag.m`.
- `ErrorSurfacePlot(Smin, Smax, K1, K2, T, rate, volatility, option_price)`: Plots Figure 1.1, i.e. a surface plot of the maximum absolute error of the PDE solver for pairs of N, J on $[0, 200]$.
- `ErrorAnalysis(K1, K2, T, rate, volatility, Smin, Smax, N, J, option_price)`: Plots Figure 1.2, the absolute error of our PDE solver as a function of Smax.
- `PlotComparison(S_PDE, V_PDE, V_BS)`: Plots Figure 1.3, i.e. a comparison of the solution obtained by `PDE_bullspread.m` and the Black Scholes solution, i.e. `blsprice`.
- `FiniteDifferenceDelta(S_PDE,V_PDE)`: Plots Figure 1.3, i.e. a comparison of the solution obtained by `PDE_bullspread.m` and the Black Scholes solution, i.e. `blsprice`.
- `CompareDelta(S_PDE, finDelta, option_delta)`: Plots Figure 1.4, i.e. a comparison of the finite difference delta and the Black Scholes delta obtained by `blsdelta`.

- `MonteCarlo(K, Smin, Smax, rate, volatility, dt, T, M, option_payoff, option_price, barrier)`: Sends a command to run the four Monte Carlo methods in part I and returns the average CPU times, prices, variances and sample sizes. The option payoff should be a function handle and the M can be an array of 4 numbers or just one number.
- `AntitheticVarianceReduction(Smin, Smax, rate, volatility, dt, T, M, option_payoff, barrier)`: Runs the antithetic variance reduction method and returns the average cpu times, prices, variances and sample sizes. The option payoff should be a function handle.
- `NaiveMethod(K, Smin, Smax, rate, volatility, dt, T, M, option_payoff, barrier)`: Runs the naive method and returns the average cpu times, prices, variances and sample sizes. The option payoff should be a function handle.
- `ControlVariates(Smin, Smax, rate, volatility, dt, T, M, option_payoff, barrier)`: Runs the control variates method and returns the average cpu times, prices, variances and sample sizes. The option payoff should be a function handle.
- `ImportanceSampling(K, Smin, Smax, rate, volatility, dt, T, M, option_payoff)`: Runs the important sampling method and returns the average cpu times, prices, variances and sample sizes. The option payoff should be a function handle.
- `PrintResults(times, variances, sample_sizes, error_MC)`: Prints a table of the CPU times, variances, sample sizes and an error comparison to the Black Scholes prices. Used for convenience when writing the report.
- `PlotMonteCarlo(Smin, Smax, prices, option_price)`: Plots the option prices obtained by the Monte Carlo methods and compares them to the Black Scholes prices (in part I only).
- `AntitheticDelta(M, rate, volatility, dt, T, Smin, Smax, option_payoff)`: Calculates the delta using antithetic variance reduction and path recycling.
- `GeometricBrownianMotion(S0, rate, volatility, dt, T)`: Simulates a single stock path of geometric Brownian motion for the Monte Carlo methods.

C.2 Part II

- `Main.m`: Script to run the project. Each question is split into a section which can be run independently.
- `SetParameters.m`: Initialises the parameters used in part II.
- `PlotVolatility(Smin, Smax, T, volatility)`: Plots Figure 2.1, i.e. the surface of the local volatility model.
- `MonteCarlo(K, Smin, Smax, rate, volatility, dt, T, M, put_payoff, 0)`: Sends a command to run the three Monte Carlo methods in part II and returns the average CPU times, prices, variances and sample sizes. The option payoff should be a function handle and the M can be an array of 4 numbers or just one number.
- `AntitheticVarianceReduction(Smin, Smax, rate, volatility, dt, T, M, option_payoff, barrier)`: Runs the antithetic variance reduction method and returns the average cpu times, prices, variances and sample sizes. The option payoff should be a function handle.
- `NaiveMethod(K, Smin, Smax, rate, volatility, dt, T, M, option_payoff, barrier)`: Runs the naive method and returns the average cpu times, prices, variances and sample sizes. The option payoff should be a function handle.

- `ControlVariates(Smin, Smax, rate, volatility, dt, T, M, option_payoff, barrier)`: Runs the control variates method and returns the average cpu times, prices, variances and sample sizes. The option payoff should be a function handle.
- `PlotMonteCarlo(Smin, Smax, prices, barrier)`: Plots the prices obtained by the Monte Carlo simulations.
- `PrintResults(times, variances, sample_sizes)`: Prints the Monte Carlo results which are put into tables in the report.
- `PlotVariances(Smin, Smax, variances, barrier)`: Plots the variance for each of the Monte Carlo methods
- `BarrierDeltaComparison(Smin, Smax, rate, volatility, dt, T, M, option_payoff)`: Plots the down-and-out put option's delta for a few barrier prices, 0 and 5:8:45.
- `BarrierComparison(Smin, Smax, rate, volatility, dt, T, M, barrier_payoff)`: Plots the price of the down-and-out barrier with Sb ranging from 0 and 5:8:45
- `AntitheticDelta(M, rate, volatility, dt, T, Smin, Smax, option_payoff, barrier)`: Calculates the delta of the bull call spread with the antithetic variance reduction method with path recycling.

List of Figures

List of Tables