Numerical Methods for Option Pricing

Mark Richardson

March 2009

Contents

1 Introduction

Black and Scholes published their seminal work on option pricing in 1973 [3]. In it, they described a mathematical framework for calculating the fair price of a European option in which they used a no-arbitrage argument to derive a partial differential equation which governs the evolution of the option price with respect to the time to expiry, t, and the price of the underlying asset, S.

As a starting point, we will briefly discuss options in general and we will work through the derivation of the Black-Scholes PDE, highlighting along the way some of the key assumptions that the model makes. We will then compare and contrast three of the primary numerical techniques that are currently used by financial professionals for determining the price of an option: Finite difference solution of the Black-Scholes PDE, The Binomial Method, and Monte-Carlo simulation.

2 A brief introduction to derivatives and options

Derivatives are named as they are because they *derive* their value from the behaviour of the underlying asset. Any derivative contract in existence must have been sold by one counterparty (we say that the derivative writer is *short*) and the same derivative must have been purchased by another counterparty (we say that the derivative holder is *long*).

The simplest types of options (called *European* or *Vanilla* options) come in two main brands, *Calls* and *Puts*. If you are long a call, then you have the right, at some known point in the future (called expiry, T) to purchase a unit of the underlying asset, whatever that may be, for a pre-determined price (called the exercise, or strike price, K). If you are long a put, then the same applies, but you instead have the right to sell the underlying. If you are short either of these options, you have received a premium, but may be forced to either buy or sell the underlying in future, according to the terms of the contract. The aim of the mathematics we are about to discuss is to determine what the fair price for this premium should be.

If we denote the current price of the underlying by S, then the payoffs at expiry, T, for a given exercise price, K, of European Calls and Puts is:

$$C(S,T) = \max(S-K,0) \qquad\qquad P(S,T) = \max(K-S,0) \tag{1}$$

3 The Black-Scholes Model

3.1 Asset prices and Ito's Lemma

It is well known that asset prices can be modelled by the following Stochastic Differential Equation (SDE):

$$dS = \sigma S dX + \mu S dt \tag{2}$$

Where S is the asset value, dS and dt are incremental changes in asset value and time respectively, σ is the volatility, μ is the drift and dX is a Wiener process. Because of the dX term, S itself is also a random variable.

Ito's lemma allows us to relate a small change in a random variable, S to a small change in a *function* of the random variable (and another arbitrary variable, t), $V(S,t)$. That is, if S satisfies (2), then $V(S,t)$ satisfies the following (for a sketch proof, see [1], Ch2).

$$dV = \sigma S \frac{\partial V}{\partial S} dX + \left(\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} \right) dt \tag{3}$$

3.2 Derivation of the Black-Scholes PDE

We begin by constructing a portfolio, Π, in which we are long one option, $V(S,t)$ and short an as yet unknown quantity, Δ, of the underlying asset, S. Thus:

$$\Pi = V - \Delta S \qquad \Rightarrow \qquad d\Pi = dV - \Delta dS \tag{4}$$

Upon substituting (2) and (3) into (4), we obtain:

$$\begin{aligned} d\Pi &= \sigma S \frac{\partial V}{\partial S} dX + \left(\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} \right) dt - \Delta \left(\sigma S dX + \mu S dt \right) \\ &= \sigma S \left(\frac{\partial V}{\partial S} - \Delta \right) dX + \left(\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} - \mu \Delta S \right) dt \end{aligned}$$

We now see that it is possible to make a choice for Δ that eliminates the dX term and hence makes the change in the value of the portfolio independent of the random walk taken by the underlying asset. This choice is $\Delta = \frac{\partial V}{\partial S}$, and results in:

$$d\Pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt$$

Black and Scholes' next step was to ingeniously invoke the no-arbitrage principle. This, in essence, says that the value of the portfolio must be equal (on average) to the value of the portfolio invested at the *risk-free interest rate*, r. If this were not the case, savvy arbitrageurs would be able to risklessly profit. They could do this by either shorting the portfolio and putting the proceeds in the bank if the portfolio was undervalued relative to the risk-free rate, and by applying the opposite strategy if the converse was true. Thus:

$$r\Pi dt = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt$$

Then, substituting in 4 together with our value for Δ and cancelling through by dt gives the Black-Scholes PDE:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{5}$$

3.3 Boundary Conditions

The Black-Scholes PDE describes the evolution of *any* derivative whose underlying asset satisfies the Black-Scholes assumptions. In order to price the derivative, we need to solve (5) together with some boundary conditions. In the case of the European Call, we know exactly what the value of the option needs to be at expiry. This is given by (1). Also following from (1), for any strike price, $K > 0$, $\max(0 - K, 0) = 0$. Thus we have, $C(0,t) = 0$. Finally, consider the case where the underlying asset increases without bound. The strike price, K becomes irrelevant and we have: $C(S \to \infty, t) = S$. In the case of other types of options, similar boundary conditions can be derived.

3.4 Closed Form Solutions

When considering some types of option, there sometimes exists an analytical, closed form solution which, under the Black-Scholes assumptions, delivers the 'fair' price of the option with respect to the various input parameters. For some of the more advanced option contracts, the closed form solution may not be so easy to obtain and thus a numerical method may be required. These numerical methods will be the subject of the rest of this essay, but first we present the closed form solution for the European Put option:

$$P(S,t) = Ke^{-r(T-t)}N(-d_2) - SN(-d_1) \qquad \text{where} \qquad d1 = \frac{\log\frac{S}{K} + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

$$\text{and} \qquad N(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\frac{1}{2}s^2}\mathrm{d}s \qquad\qquad d2 = \frac{\log\frac{S}{K} + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

For the derivation of these formulae, see [1] or [2].

4 Numerical Methods

European Puts and Call are known as *vanilla* options, that is, they are the most basic type of option, with relatively simple features and payoffs. Indeed, for these options, a closed form solution exists as we have just shown. On the global financial markets, simple vanilla puts and calls are traded along with options that can often have far more complicated payoffs. In general, these are know as *exotic options*. Sometimes, certain exotics (for example European[1] *Barriers* and *Lookbacks*) can be found to have a closed form solution. However, for other products, (for example, *American* or *Asian* options), a closed form solution does not exist. In these cases, the only way a market participant will be able to obtain a price is by using an appropriate numerical method.

In this section, we will discuss three of the main techniques used for pricing options and to test them, we will implement then on a European Put, for which we have the closed form solution (6). In the next section, we will then attempt to apply these techniques to some exotic option contracts.

4.1 Finite Difference Methods

In general, Finite difference (FD) methods are used to numerically approximate the solutions of certain ordinary and partial differential equations. In the case of a bivariate, parabolic PDE, such as (5), we start by establishing a rectangular solution domain in the two variables, S and t. We then form *finite difference approximations* to each of the derivative terms in the PDE.

FD methods[2] create a mathematical relationship which links together every point on the solution domain, like a chain. The first links in the chain are the boundary conditions and from these, we 'discover' what every other point in the domain has to be. Perhaps the most popular FD methods used in computational finance are: *Explicit Euler*, *Implicit Euler*, and the *Crank-Nicolson* method. Using each of these three methods has its advantages and disadvantages. The easiest scheme of the three to implement is the Explicit Euler method. Implicit Euler and Crank-Nicolson are implicit methods, which generally require a system of linear equations to be solved at each time step, which can be computationally intensive on a fine mesh. The main disadvantage to using Explicit Euler is that it is unstable for certain choices of domain discretisation. Though Implicit Euler and Crank-Nicolson involve solving linear systems of equations, they are each unconditionally stable with respect to the domain discretisation. Crank-Nicolson exhibits the greatest accuracy of the three for a given domain discretisation.

The reason that Finite Difference methods are a popular choice for pricing options is that *all* options will satisfy the Black-Scholes PDE, (5) or appropriate variants of it. The difference between each option contract is in determining the boundary conditions that it

[1] *European* meaning: exercised at expiry, T. The other types are *American*, where exercise is permitted at any point before expiry, and *Bermudan*, where exercise is permitted at a fixed number of points before expiry

[2] The following is well known and can be found in any good text on FD methods e.g. [4]

satisfies. Finite Difference methods can be applied to American (early exercise) Options and they can also be used for many exotic contracts.

To illustrate the finite difference method in practice, we will now price a European Put using the Implicit Euler Method. We first define the domain discretisation in S & t:

$$\begin{array}{rlll} \text{Asset Value Discretisation :} & \{0, \delta S, 2\delta S, 3\delta S, ..., M\delta S\} & \text{where} & M\delta S = S_{max} \\ \text{Time Discretisation :} & \{0, \delta t, 2\delta t, 3\delta t, ..., N\delta t\} & \text{where} & N\delta t = T \end{array}$$

$$\text{and} \qquad V_{i,j} = V(i\delta S, j\delta t) \qquad i = 0, 1, ..., M \qquad j = 0, 1, ..., N$$

S_{max} is a maximum value for S that we must choose sufficiently large (we cannot, of course, discretely solve for all values of S up to infinity!). The boundary conditions for the European Put are:

$$V(S, T) = \max(K - S, 0) \qquad V(0, t) = Ke^{-r(T-t)} \qquad V(S_{max}, t) = 0 \tag{6}$$

Translating these into the mesh notation gives:

$$\begin{aligned} V_{i,N} &= \max(K - (i\delta S), 0) \qquad i = 0, 1, ..., M \\ V_{0,j} &= Ke^{-r(N-j)\delta t} \qquad j = 0, 1, ..., N \\ V_{M,j} &= 0 \qquad j = 0, 1, ..., N \end{aligned}$$

Since we are given the payoff at expiry, our problem is to solve the Black-Scholes PDE *backwards* in time from expiry to the present time ($t = 0$). Consider the Black-Scholes PDE (5). There are three derivative terms that we need to approximate, and for the Implicit Euler scheme, a possible discretisation is given by[3]:

$$\frac{V_{i,j} - V_{i,j-1}}{\delta t} + \frac{1}{2}\sigma^2(i\delta S)^2\frac{V_{i+1,j-1} - 2V_{i,j-1} + V_{i-1,j-1}}{\delta S^2} + r(i\delta S)\frac{V_{i+1,j-1} - V_{i-1,j-1}}{2\delta S} - rV_{i,j-1} = 0 \tag{7}$$

This uses a backward difference to approximate the time derivative (since we are solving backward in time from expiry) and a central difference to approximate the first derivative in asset value. The second derivative term is approximated by the usual second difference. With some simple algebra, we can re-write (7) as:

$$V_{i,j} = A_iV_{i-1,j-1} + B_iV_{i,j-1} + C_iV_{i+1,j-1}$$

$$A_i = \frac{1}{2}\delta t(ri - \sigma^2i^2) \qquad B_i = 1 + (\sigma^2i^2 + r)\delta t \qquad C_i = -\frac{1}{2}\delta t(ri + \sigma^2i^2)$$

We are now ready to implement the Implicit Euler scheme, though we should note again that since this is an implicit scheme, we will be inverting a matrix to solve a linear system at each time step. My MATLAB implementation is given in Appendix 7.1 and uses MATLAB's built in Backslash command to invert the tridiagonal matrix. Later, when we price American options, we will need to replace this with an iterative scheme.

Figure 1 shows a sample solution surface using this code for $K = 50$, $r = 0.05$, $\sigma = 0.2$ and $T = 3$.

One of the key advantages to using finite difference methods is that we are able to automatically determine the option values at every mesh point in the domain. If we require an option value for a particular value of S and t, all we need to do is extract that point from the matrix called *'solution_mesh'*. Since we are using a finite discretisation, we do not have a continuous solution and therefore we do not know the value of the option at any arbitrary

[3]From [5] and [6]

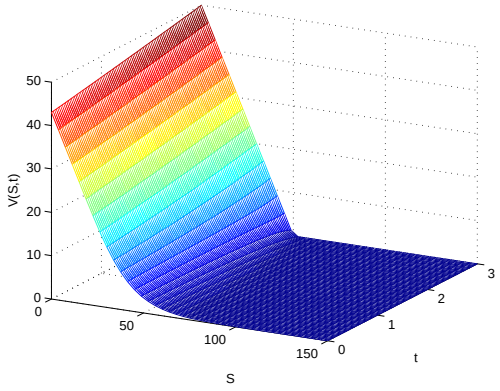

Figure 1: European Put values calculated using the Implicit Euler method

point in the domain, just at the mesh points. If it happens that we require an option value which does not coincide with one of our mesh points, we may use some kind of interpolation technique (linear interpolation is usually good enough, if a fine enough mesh has been used to begin with) to extract an approximation to the value we require.

4.2 Risk-Neutral Valuation

The last section discussed the FD method applied to the Black-Scholes PDE (5). The Black-Scholes model makes an assumption that the underlying asset satisfies the SDE (2). This SDE has two parameters associated with it; the volatility, σ (σ is the standard deviation of the random asset price movements), and the drift μ (μ can be thought of as the expected return from the asset). In the derivation of the (5), we made a choice for the Delta (Δ) of the option that eliminated the random component of the asset price movements. This had the added effect of also cancelling the drift term from the PDE so that the only two remaining parameters were σ and r.

In the next two chapters, we take a different approach which does not rely on solving the Black-Scholes PDE. Instead we will try and model the behaviour of the underlying asset itself, from which we will obtain estimates for the corresponding option values. To allow us to do this, we make an assumption that all investors are 'risk-neutral', that is they do not require a premium to encourage them to take risks. A consequence of this assumption is that the average return on assets must be equal to the risk free interest rate, r. Thus, in (2), we may replace μ (the expected return of the asset) with r.

4.3 Binomial Tree Methods

The Binomial tree method is widely favoured amongst market professionals because of its ease of implementation, its simple feel, and its versatility in applying it to all manner of options, simple and complex. The technique was first introduced by Cox, Ross and Rubenstein in their now famous 1979 paper [7].

The general idea is to build a 'tree' of values up from the current asset price, assuming that at each time step, δt, the asset increases in value by a fixed factor, u, or decreases by a factor, d. We make the additional assumption that $u = \frac{1}{d}$, in order to obtain a tree that recombines so that the effect of a down movement followed by an up movement is the same as the effect of an up movement, followed by a down movement. Each up movement is associated with a probability of occurrence, p and the down movement therefore has probability, $1 - p$.

This process is repeated recursively until we arrive at the expiry time, T, at which point (in the case of European Puts and Calls), we assess the value of the option given by the payoff formulae (1) at each of the 'nodes'. We then work back through the tree by working out the expected value of the option at the previous time step and using the risk-neutral assumption to discount at the risk-free interest rate.

A popular way of setting the parameters u, d and p is given in [8], and this is:

$$A = \frac{1}{2}\left(e^{-r\delta t}e^{(r+\sigma^2)\delta t}\right) \quad u = A + \sqrt{A^2 - 1} \qquad d = \frac{1}{u} \qquad p = \frac{e^{r\delta t} - d}{u - d} \tag{8}$$

This choice ensures that that our Binomial method is compatible with the Black-Scholes assumptions and indeed, the limiting behaviour (as $\delta t \to 0$) of the option values can be shown to converge to the Black-Scholes values [8]. I have implemented this method for the European Put and the MATLAB code is given in Appendix 7.2.

A key advantage to the Binomial tree model is that it can be easily extended to valuing options that can be exercised before expiry e.g. American Options. We shall demonstrate how to do this later on in this report.

There are limitations to this model, and whilst it performs well evaluating Vanilla options such as Puts and Calls, the Binomial method is well known to have difficulty when valuing certain exotics. For example when applying the Binomial method to Barrier options, a well observed phenomenon is the (e.g.) over-valuation of knock-out calls. In this case, this is due to the true barrier of the option being located above the Binomial tree nodes, which is where the Binomial method implicitly assumes that the barrier is. We can also encounter problems when evaluating other types of path-dependency due to the exponential growth of the binomial coefficients causing floating point overflow problems.

An extension to the Binomial tree method is the *Trinomial* tree method, and though we shall not discuss it in detail here, the general idea is that the asset can move up, down or stay stationary, with corresponding probabilities. Once again, the parameters are chosen to mimic the behaviour of the Black-Scholes model.

My implementation of the Binomial tree method for pricing a European Put is given in Appendix 7.2.

4.4 Monte-Carlo Simulation

The Black-Scholes model assumes that asset returns are distributed according to the SDE (2). Under the risk-neutral assumption, this must be equivalent to the average option payoff, discounted back to the present value using the risk-free interest rate, r. Monte-Carlo simulation is the name of the general process by which a large number of asset paths are generated that evolve according to (2). We compute the option payoff for each path in the sample and take the arithmetic mean to compute an approximation to the 'fair' value of the option. By the assumptions of the Black-Scholes model, any Monte-Carlo simulation must converge to the Black-Scholes option value in the limit as the number of paths in the sample, $M \to \infty$.

The main benefits to using Monte-Carlo simulation is that it is generally quite easy to implement and can be used without too much difficulty to value a large range of European style exotics. Also, Monte-Carlo simulation is *reliable*, and providing enough sample paths are taken, we can have a great deal of (statistical) confidence that our prices are accurate. It is therefore often used as the benchmark valuation technique for many complex, European style exotics.

However, this accuracy can often come at a large computational cost. To generate sufficiently accurate prices, a large number of paths must be generated (perhaps in the region of 10^6). If we happen to be pricing some kind of path-dependent exotic, then we

will need to know the value of the asset at every point on the path (and we could typically sample perhaps, 10^3 increments in the Wiener process). We could therefore potentially be looking to store and evaluate 10^9 asset values for one option valuation. Multiply this by the many hundreds of quick evaluations an options trader may need to perform during one day and it is not hard to see how there can be problems.

Lastly, one of the main drawbacks to using Monte-Carlo simulation is that though it works very well for pricing European-style path-dependent options, it is difficult to implement for early exercise (American-style) options. Binomial trees and Finite Difference methods are, in general, far better suited to pricing these types of contract. My implementation of Monte-Carlo simulation for pricing a European Put is given in Appendix 7.3.

Table 1 shows the performance of the three techniques against the 'true' Black-Scholes price for a European Put with $K = 50$, $r = 0.05$, $\sigma = 0.25$, & $T = 3$. The table shows the variation of the option price with the underlying price, S. The results demonstrate that the three techniques perform well, are mutually consistent, and agree with the Black-Scholes value. However, in practice, there is no real need for using such numerical techniques when we have an explicit formula. We will now attempt to price an option contract for which there is no closed formula.

S	Black-Scholes	Binomial Tree	Monte-Carlo	Implicit Euler
10	33.0363	33.0362	33.0345	33.0369
15	28.0619	28.0618	28.0595	28.0629
20	23.2276	23.2273	23.2291	23.2300
25	18.7361	18.7362	18.7339	18.7390
30	14.7739	14.7736	14.7748	14.7749
35	11.4384	11.4388	11.4402	11.4402
40	8.7338	8.7352	8.7374	8.7348
45	6.6021	6.6025	6.6014	6.6012
50	4.9564	4.9556	4.9559	4.9563
55	3.7046	3.7073	3.7076	3.7042
60	2.7621	2.7640	2.7602	2.7612
65	2.0574	2.0592	2.0581	2.0571
70	1.5328	1.5346	1.5324	1.5325
75	1.1430	1.1427	1.1407	1.1426
80	0.8538	0.8549	0.8543	0.8537
85	0.6392	0.6401	0.6405	0.6391
90	0.4797	0.4803	0.4790	0.4794

Table 1: A comparison with the Black-Scholes price for a European Put

5 Pricing American Options

American options allow the holder to exercise the option at any point in time up to and including expiry. Clearly, this somewhat complicates the problem of pricing, and indeed for American Puts and Calls, there is no known pricing formula as there is for European options. Thus, we shall now attempt to apply the numerical methods developed above to pricing these contracts. In particular, we shall deal with pricing the American Put.

As discussed in the previous section, Monte-Carlo methods are generally hard to implement for early-exercise options, so here we shall implement and compare the Finite Difference and Binomial Tree methods.

5.1 Boundary Conditions and Early Exercise Criteria

The first problem to consider is: What is the criteria for early exercise? That is, under which circumstances should the option holder not wait until expiry to exercise, and instead exercise immediately? Recall that at *expiry*, the payoff of a (European or American) Put is:

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

We can therefore infer that since the payoff is the same at expiry for both the European and American options, then the corresponding boundary condition (at $t = T$) is the same. For the boundary condition at $S = 0$, as in the European case, we expect that the payoff will again be K, discounted in time at the risk free rate, so that $P(0,t) = Ke^{-r(T-t)}$. Lastly, for the boundary as $S \to \infty$, we expect that the payoff to be zero, i.e. $P(S \to \infty, t) = 0$. This leads us to conclude that the boundary conditions for the American Put are *identical* to the European case. Knowing this, we must devise a strategy for deciding, at each point in time, whether it is optimal to exercise the option or not.

At each point in time, it makes sense that we should compare the expected value of waiting until expiry to the value of exercising immediately. This is fortunate, since we know each of these quantities at any given point. The first is the Black-Scholes price for the European case, whilst the second is $\max(K - S, 0)$, for any particular value of S. Therefore, the larger of these two quantities is the current value of the American Option i.e. for the American Put:

$$P_{Am}(S,t) = \max(K - S, P_{Eu}(S,t)) \tag{9}$$

We will use this criteria to evaluate the option using the Binomial and Finite Difference method approach.

5.2 Pricing American Options using a Binomial Tree

Out of the two methods that we will be using to price an American Put, the Binomial Tree method extends the most naturally from the European case and deserves the lighter treatment of the two. When we priced a European option using the Binomial Tree, we first built a tree of asset values in steps of δt, until we arrived at expiry, T. At this point, our tree had divided and recombined into $N + 1$ final values, where N was the number of time steps in the tree and $\delta t = \frac{T}{N}$. At this point, we evaluated the Put option at each of the final 'nodes' and then tracked back through the tree, by computing the expectation of the price of the option at each previous time-step, whilst discounting using the risk-free interest rate, r.

To price the American Put, the only modification to the European approach is that at each node, we need to make the comparison given in (9), above. That is, we compare

the value of the European Option, and the value of exercising at that point, an choose the greater of the two. Repeating this process back to time $t = 0$ gives us the present value of the American Put.

My MATLAB implementation for pricing the American Put using the Binomial Tree is given in Appendix 7.5.

5.3 Pricing American Options using Finite Differences

Although the philosophy behind pricing the American Put using Finite Differences is identical to that of the Binomial Tree approach, the implementation is a little more complicated and requires some more carful consideration.

Again, we need to apply (9) at each step in the time discretisation, $\{0, \delta t, 2\delta t, 3\delta t, ..., N\delta t\}$, as we solve the Black-Scholes PDE in time. If we were using an explicit method, this would be straightforward. Consider (9) in terms of the Finite Difference mesh notation:

$$V_{i,j} = \max(K - i\delta S, V_{i,j}) \tag{10}$$

If using Explicit Euler, we would first work out the value $V_{i,j}$, make the above comparison, and there would be no problem. However, let us suppose that we want to use an implicit method in order to circumvent any possible instability. Then, we cannot directly apply (10), since due to us having to solve a linear system to progress from one time step to the next, we do not know the value of $V_{i,j}$ until we get to the next time step. There is in fact no way to solve this problem by inverting the tridiagonal matrix (or by equivalent methods such as LU factorisation).

Instead, we must use an *iterative method*. The simplest such method is called *Jacobi iteration*. A variant upon this is *Gauss-Siedel* iteration and a variant of this method, which we shall use (as suggested in [6]), is called *Successive Over Relaxation* or SOR iteration. The pseudo-code for the SOR iteration applied to the $n \times n$ linear system, $A\mathbf{x} = \mathbf{b}$, is:

for k = 1,2,... **do**
  for i = 1,2,...,n **do**
    $z_i^{k+1} = \frac{1}{a_{ii}} \left\{ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right\}$
    $x_i^{k+1} = \omega z_i^{k+1} + (1 - \omega) x_i^k$
  end for
end for

Where ω is called the relaxation parameter. Thus, we can replace the MATLAB 'backslash' command for inverting the Implicit Euler tridiagonal matrix with the above iteration and, by defining a tolerance parameter, we can obtain convergence to any required degree of accuracy (10^{-3} is typically sufficient).

5.4 Comparing the Performance of the Two Methods

My MATLAB implementation for pricing the American Put using Implicit Euler and Gauss-Siedel SOR iteration is given in Appendix 7.6. Table 2 shows the the results of implementing the two methods whilst varying the underlying price, S. The other parameter values for were: $K = 50$, $r = 0.05$, $\sigma = 0.25$ and $T = 3$. We observe that there is a good agreement between the two sets of values.

S	Binomial Tree	Implicit Euler
15	35.0000	35.0000
20	30.0000	30.0000
25	25.0000	25.0000
30	20.0000	20.0000
35	15.0147	15.0291
40	10.9440	10.9492
45	7.9999	7.9904
50	5.8547	5.8527
55	4.2955	4.2891
60	3.1541	3.1456
65	2.3202	2.3162
70	1.7119	1.7080
75	1.2668	1.2618
80	0.9391	0.9367
85	0.6984	0.6969

Table 2: Binomial tree and Implicit Euler for pricing an American Put

6 Conclusions

I have tried to write this special topic in such a way that a fellow MSc student (or another peer with a similar level of knowledge to myself) could understand this report without having any prior knowledge of Mathematical Finance or Numerical methods. I have therefore explained all the key concepts, such as the basic Black-Scholes theory and the construction of the three numerical methods, under this assumption. I hope therefore, that this report is comprehensible in the way that I intended it to be.

Each of the three numerical methods has its advantages and disadvantages of use. Monte-Carlo simulation is great for pricing European style options and is easy to code, however, it is a lot more challenging to price early exercise options in this way, and the computational cost for some contracts is large. The binomial tree method is also relatively easy to implement and is flexible enough to deal with early exercise, however, it can stuggle with some kinds of exotic contracts, such as Barriers. Finally, Finite Difference methods allow us to obtain an option's value over the entire solution domain and the only real difference between different contracts are the boundary conditions that it must satisfy. However, FD methods are a little harder to code than Monte-Carlo and the Binomial tree, and can be prone to instability (in the case of an explicit method).

Whilst working on this special topic, I considered many exotic contracts before deciding to focus my attention on pricing American Options. The main reason for my decision to do this was that there does not exist a closed form solution for American Options, whilst for certain exotic contracts, such as barriers and lookbacks, there can exist a closed form solution. Nevertheless, I initially gained familiarity with the three techniques discussed above by trying to code up pricing software in MATLAB. I have therefore included my codes for pricing an 'up and out' European Barrier Call and a fixed-strike European Lookback Call using the Binomial Tree and Monte-Carlo methods (though I have not included Monte-Carlo for the Barrier option). These codes can be found in Appendices 7.7 - 7.9. However, I would reccomend treating these codes with caution as I have not been able to test them against any other pricing software to verify if they are producing accurate prices or not.

7 Appendix

7.1 MATLAB code for European Put: Implicit Euler

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
S = 50;                            % Value of the underlying
K = 50;                           % Strike (exercise price)
r = 0.05;                         % Risk free interest rate
sigma = 0.2;                      % Volatility
T = 3;                            % Time to expiry
%%%%%%%%%% Method Paramaters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
M = 80;                                    % Number of asset mesh points
N = 200;                                   % Number of time mesh points
Szero = 0;                                 % Specify extremes
Smax= 150;
%%%%%%%%%% Setup Mesh and BCs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
solution_mesh=zeros(N+1,M+1);              % Create Mesh
Smesh=0:(Smax/M):Smax;                     % Mesh of equally spaced S values
Tmesh=T:-(T/N):0;                          % Mesh of equally spaced T values
dt=T/N;                                    % Specify timestep, dt
solution_mesh(1,:)= max(K-Smesh,0);        % Payoff BC (Put)
solution_mesh(:,1)= K*exp(-r*(T-Tmesh));   % BC at S=0
solution_mesh(:,M+1)= 0;                   % BC at S=M
A = @(i) 0.5*dt*(r*i-sigma^2*i^2);
B = @(i) 1 + (sigma^2*i^2 + r)*dt;         % Define the functions A, B & C
C = @(i) -0.5*dt*(sigma^2*i^2+r*i);
%%%%%%%%%% Construct Tridiagonal Matrix, Tri %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Acoeffs = zeros(M+1,1);Bcoeffs = zeros(M+1,1);Ccoeffs = zeros(M+1,1);
for i=1:M+1
    Acoeffs(i) = A(i-1); Bcoeffs(i) = B(i-1); Ccoeffs(i) = C(i-1);
end
Tri=diag(Acoeffs(2:end),-1)+diag(Bcoeffs)+diag(Ccoeffs(1:end-1),+1);
Tri_Inv=inv(Tri);                          % Compute inverse
%%%%%%%%%% Implicit Euler Iteration %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for j=1:N
    temp=zeros(M+1,1);
    temp(1)=A(0)*solution_mesh(j+1,1);
    temp(end)=C(M)*solution_mesh(j+1,M+1); % Boundary terms
    RHS=solution_mesh(j,:)'-temp;
    temp=Tri_Inv*RHS;
    solution_mesh(j+1,(2:end-1))=temp(2:end-1);
end
mesh(Smesh,Tmesh,solution_mesh)
xlabel('S');ylabel('t');zlabel('V(S,t)')
%%%%%%%%% Extract Desired Values Using Interpolation %%%%%%%%%%%%%%%%%%%%%%%%
interp1(Smesh,solution_mesh(N+1,:),S)
```

7.2 MATLAB code for European Put: Binomial Tree

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = 55;                          % Value of the underlying
K = 50;                          % Strike (exercise price)
r = 0.05;                        % Risk free interest rate
sigma = 0.25;                    % Volatility
T = 3;                           % Time to expiry

%%%%%%%%%% Binomial method parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N = 256;
dt = T/N; A = 0.5*(exp(-r* dt)+exp((r+sigma^2)*dt));
d = A - sqrt(A^2-1); u = A + sqrt(A^2-1);
p = (exp(r*dt)-d)/(u-d);

%%%%%%%%%% Construct the tree %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tree=S;                          % Initialise tree

for i=1:N
    mult=u*ones(i+1,1);          % Create array of 'up multiplications'
    mult(end)=d;                 % Make the last entry a 'down multiplication'
    Tree(i+1)=Tree(i);
        if i==1
            Tree=Tree';          % Make sure T is in column form
        end
    Tree=Tree.*mult;
end

Tree=max(K-Tree,0);              % Compute Put option values at t=T

Back = p*eye(N+1,N+1) + (1-p)*diag(ones(N,1),1);
Back = sparse(Back);             % Define matrix to track back through tree

%%%%%%%%%% Track back through the tree to time t=0 %%%%%%%%%%%%%%%%%%%%%%%%

for i = N:-1:1
Tree = Back(1:i,1:i+1)*Tree;
end

%%%%%%%%%% Discount under risk-neutral assumption %%%%%%%%%%%%%%%%%%%%%%%%

Option_Value = exp(-r*T)*Tree;
display(Option_Value)
```

7.3 MATLAB code for European Put: Monte-Carlo Simulation

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = 55;                                 % Value of the underlying
K = 50;                                 % Strike (exercise price)
r = 0.05;                               % Risk free interest rate
sigma = 0.25;                           % Volatility
T = 3;                                  % Time to expiry

%%%%%%%%%% Monte-Carlo Method Parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% randn('state',0)                      % Repeatable trials on/off
M=1e7;                                  % Number of Monte-Carlo trials

%%%%%%%%%% Use final values to compute %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

final_vals=S*exp((r-0.5*sigma^2)*T + sigma*sqrt(T)*randn(M,1));

option_values=max(K-final_vals,0);      % Evaluate the Put option options
present_vals=exp(-r*T)*option_values;   % Discount under r-n assumption
int=1.96*std(present_vals)/sqrt(M);     % Compute confidence intervals
put_value=mean(present_vals);           % Take the average

display(put_value)
display([put_value-int put_value+int])
```

7.4 MATLAB code for European Put: Black-Scholes Formula

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = 50;                           % Value of the underlying
K = 50;                           % Strike (exercise price)
r = 0.05;                         % Risk free interest rate
sigma = 0.25;                     % Volatility
T = 3;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

d1 = (log(S/K) + (r + 0.5*sigma^2)*T)/(sigma*sqrt(T));
d2 = d1 - sigma*sqrt(T);
N1 = 0.5*(1+erf(-d1/sqrt(2)));
N2 = 0.5*(1+erf(-d2/sqrt(2)));
value = K.*exp(-r*T).*N2-S.*N1;
```

7.5 MATLAB code for American Put: Binomial Tree

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = 50;                          % Value of the underlying
K = 50;                          % Strike (exercise price)
r = 0.05;                        % Risk free interest rate
sigma = 0.25;                    % Volatility
T = 3;

%%%%%%%%%% Binomial method parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N = 512;
dt = T/N; A = 0.5*(exp(-r* dt)+exp((r+sigma^2)*dt));
up = A + sqrt(A^2-1); down = A - sqrt(A^2-1);
p = (exp(r*dt)-down)/(up-down);

%%%%%%%%%% Construct the tree %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tree = S*up.^((N:-1:0)').*down.^((0:N)');

% Compute Put option values at t=T
Tree=max(K-Tree,0);

% Matrix to compute expectations
Back = p*eye(N+1,N+1) + (1-p)*diag(ones(N,1),1);

%%%%%%%%%% Track back through the tree to time t=0 %%%%%%%%%%%%%%%%%%%%%%%%%

for i = N:-1:1
discounted = Back(1:i,1:i+1)*Tree*exp(-r*dt);   % One step back, discounted
sharevals = S*up.^((i-1:-1:0)').*down.^((0:i-1)');  % Share values at time step
Tree=max(discounted,K-sharevals);               % American Exercise Comparison
end

%%%%%%%%%% Print Option Value %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Option_Value = Tree;
display(Option_Value)
```

7.6 MATLAB code for American Put: Implicit Euler + SOR

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                        T _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
%   V(i,j) -> V(asset,time)                |                             |
%                                       t  |                             |
%   i = 0,1,2....M      bcs at i=0,M       |                             |
%   j = N,N-1,...0      bc at j=N          |_____________________________|
%                                       Szero           S             Smax
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = 50;                                     % Value of the underlying
K = 50;                                     % Strike (exercise price)
r = 0.05;                                   % Risk free interest rate
sigma = 0.25;                               % Volatility
T = 3;

%%%%%%%%%% Method Paramaters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

M = 100;                                    % Number of asset mesh points
N = 400;                                    % Number of time mesh points
Szero = 0;                                  % Specify extremes
Smax= 150;

omega=1.2;tol=0.001;                        % SOR Parameters

%%%%%%%%%% Setup Mesh and BCs %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

solution_mesh=zeros(N+1,M+1);               % Create Mesh
Smesh=0:(Smax/M):Smax;                      % Mesh of equally spaced S values
Tmesh=T:-(T/N):0;                           % Mesh of equally spaced T values
dt=T/N;                                     % Specify timestep, dt
dS=Smax/M;                                  % Specify asset step, dS

solution_mesh(1,:)= max(K-Smesh,0);         % Payoff BC (Put)
solution_mesh(:,1)= K*exp(-r*(T-Tmesh));    % BC at S=0
solution_mesh(:,M+1)= 0;                    % BC at S=M
                                            % (M+1 due to MATLAB indexes)
a = @(i) 0.5*dt*(r*i-sigma^2*i^2);
b = @(i) 1 + (sigma^2*i^2 + r)*dt;          % Define the functions a, b & c
c = @(i) -0.5*dt*(sigma^2*i^2+r*i);

%%%%%%%%%% Construct Tridiagonal Matrix, Tri %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

acoeffs = zeros(M+1,1);bcoeffs = zeros(M+1,1);ccoeffs = zeros(M+1,1);

for i=1:M+1
    acoeffs(i) = a(i-1);
    bcoeffs(i) = b(i-1);
    ccoeffs(i) = c(i-1);
end
```

```
Tri=diag(acoeffs(3:M),-1)+diag(bcoeffs(2:M))+diag(ccoeffs(2:M-1),+1);
Tri_Inv=inv(Tri);                          % Compute inverse

%%%%%%%%% Implicit Euler + SOR Iteration %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for p=1:N
    temp=zeros(M-1,1);
    temp(1)=a(0)*solution_mesh(p+1,1);
    temp(end)=c(M)*solution_mesh(p+1,M+1); % Boundary terms
    RHS=solution_mesh(p,2:M)'-temp;

    % Gauss-Siedel SOR method: Solving Ax=b iteratively
    % Note: A is (M-1)x(M-1), b is (M-1)

    A=Tri;b=RHS;                  % Define matrix, A and RHS vector, b
    x=solution_mesh(p,2:M)';      % Define x
    xold=10000*x;                 % Initialise xold to enter loop
    n=length(x);

    while norm(xold-x)>tol
        xold=x;                     % redefine xold
        for i=1:n                   % for: rows of the matrix
            if i==1
                z=(b(i)-A(i,i+1)*x(i+1))/A(i,i);
                x(i) = max(omega*z + (1-omega)*xold(i),K-i*dS);
            elseif i==n
                z=(b(i)-A(i,i-1)*x(i-1))/A(i,i);
                x(i) = max(omega*z + (1-omega)*xold(i),K-i*dS);
            else
                z=(b(i)-A(i,i-1)*x(i-1)-A(i,i+1)*x(i+1))/ A(i,i);
                x(i) = max(omega*z + (1-omega)*xold(i),K-i*dS);
            end
        end
    end
    solution_mesh(p+1,(2:end-1))=x;
end

%mesh(Smesh,Tmesh,solution_mesh)
%xlabel('S');ylabel('t');zlabel('V(S,t)')

%%%%%%%%% Extract Desired Value Using Interpolation %%%%%%%%%%%%%%%%%%%%%%

interp1(Smesh,solution_mesh(N+1,:),S)
```

7.7 MATLAB code for European Fixed Strike Barrier Call: Binomial

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
S = 3;                      % Value of the underlying
K = 1;                      % Strike (exercise price)
r = 0.05;                   % Risk free interest rate
sigma = 0.25;               % Volatility
T = 3;                      % Time to expiry
B = 5;                      % Barrier
%%%%%%%%%% Binomial method parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N = 512;
dt = T/N; A = 0.5*(exp(-r* dt)+exp((r+sigma^2)*dt));
d = A - sqrt(A^2-1); u = A + sqrt(A^2-1);
p = (exp(r*dt)-d)/(u-d);
%%%%%%%%%% Construct the tree %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Tree=S;                         % Initialise tree
for i=1:N
    mult=u*ones(i+1,1);         % Create array of 'up multiplications'
    mult(end)=d;                % Make the last entry a 'down multiplication'
    Tree(i+1)=Tree(i);
        if i==1
            Tree=Tree';         % Make sure T is in column on the first pass
        end
    Tree=Tree.*mult;

    for j=1:length(Tree)        % Up and out Barrier condition
        if Tree(j)-B>0
            Tree(j)=0;
        end
    end
end

Tree=max(Tree-K,0);             % Compute option values at t=T
Back = p*eye(N+1,N+1) + (1-p)*diag(ones(N,1),1);
Back = sparse(Back);            % Define matrix to track back

%%%%%%%%%% Track back through the tree to time t=0 %%%%%%%%%%%%%%%%%%%%%%%%%
for i = N:-1:1
Tree = Back(1:i,1:i+1)*Tree;
end

%%%%%%%%%% Discount under risk-neutral assumption %%%%%%%%%%%%%%%%%%%%%%%%%%
Option_Value = exp(-r*T)*Tree;
display(Option_Value)
```

7.8 MATLAB code for European Lookback Call: Binomial

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = 3;                        % Value of the underlying
K = 1;                        % Strike (exercise price)
r = 0.05;                     % Risk free interest rate
sigma = 0.25;                 % Volatility
T = 3;                        % Time to expiry

%%%%%%%%%% Binomial Method Parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N = 50;                        % Number of time steps (length of binomial tree)

dt = T/N;                      % Length of each time-step (automatically computed)
A = 0.5*(exp(-r* dt)+exp((r+sigma^2)*dt));
u = A + sqrt(A^2-1); d = A - sqrt(A^2-1);
p = (exp(r*dt)-d)/(u-d);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Underlying values at time T

P=pascal(N+1);                     % Pascal's triangle

binomial_coeffs=zeros(N+1,1);      % Preallocate vector for Binomial coeffs

% Extract Binomial Coefficients
for i=1:N+1
    binomial_coeffs(i)=P(N+2-i,i);
end

%binomial_coeffs                    % This is for debugging, can comment out
%max(binomial_coeffs)

% Construct matrix for the cumulative paths (which we will later take the
% average of)

% First work out the width of the matrix (cols):

if mod(N,2)==0
    cols=N/2+1;
else
    cols=(N-1)/2+1;
end
% We can now preallocate the arrays:
paths=zeros(N+1,cols);
max_vals=zeros(N+1,cols);

% Now fill this matrix with the (truncated) binomial coeffients:
% Also create a matrix which holds the maximum values of the paths these
```

```
% correspond to the coefficients which were created in in 'paths':

for j=1:cols
    for i=j:N+2-j
        if j==1
            paths(i,j)=1;
            max_vals(i,j)=u^(N+1-i);
        else
            paths(i,j)=binomial_coeffs(j)-sum(paths(i,1:j));
            max_vals(i,j)=u^(max_vals(i,j-1)-1);
        end
    end
end

%paths
%max_vals

option_vals=max(max_vals*S-K,0);        % Option values for each maxima

% final_vals is the (.*) product of option_vals and paths, summed across
% the columns and then (mean) averaged by divig through with binomial_coeffs:

Tree=sum(option_vals.*paths,2)./binomial_coeffs;

Back = p*eye(N+1,N+1) + (1-p)*diag(ones(N,1),1);
Back = sparse(Back);            % Define matrix to track back through tree

%%%%%%%%% Track back through the tree to time t=0 %%%%%%%%%%%%%%%%%%%%%%%%

for i = N:-1:1
Tree = Back(1:i,1:i+1)*Tree;
end

%%%%%%%%% Discount under risk-neutral assumption %%%%%%%%%%%%%%%%%%%%%%%%
Lookback_Value = exp(-r*T)*Tree;
display(Lookback_Value)
```

7.9 MATLAB code for European Lookback Call: Monte-Carlo

```
%%%%%%%%%% Option parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

S = 3;                        % Value of the underlying
K = 1;                        % Strike (exercise price)
r = 0.05;                     % Risk free interest rate
sigma = 0.25;                 % Volatility
T = 3;                        % Time to expiry

%%%%%%%%%% Monte-Carlo Method Parameters %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% randn('state',0)
M=5000;                         % Number of Monte-Carlo trials
n=1000;                         % Set number of steps to compute at in [0,T]
dt=T/n;                         % Compute the time step

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

dW=sqrt(dt)*randn(M,n);     % Generate array of brownian movements drawn from N(0,dt^2)
W=cumsum(dW,2);             % Sum the array cummulativley to obtain Wiener process

t=0:dt:T;                   % Array of equal time steps
W=[zeros(M,1),W];           % Set first values to be zero

tt=repmat(t,M,1);           % Create matrix for t vals

% Compute the asset path using the solution of the asset path SDE
asset_path=S*exp((r-0.5*sigma^2)*tt+sigma*W);

% Compute the maximum value the asset reaches over the life of the option:
max_vals=max(asset_path,[],2);

% Evaluate the fixed strike lookback call option in each case:
option_values=max(max_vals-K,0);

% Discount under risk-neutral assumption
present_vals=exp(-r*T)*option_values;

call_value=mean(present_vals);
display(call_value)
```

References

[1] Paul Wilmott, Sam Howison, and Jeff Dewynne.
The Mathematics of Financial Derivatives: A student introduction
Cambridge University Press, Cambridge, UK, First edition, 1995.

[2] John C. Hull.
Options, Futures, and Other Derivatives.
Prentice-Hall, New Jersey, international edition, 1997.

[3] Fisher Black & Myron Scholes
The Pricing of Options and Corporate Liabilities.
The Journal of Political Economy. Vol 81, Issue 3, 1973.

[4] Morton & Mayers
Numerical Solution of Partial Differential Equations: An Introduction
Cambridge University Press, Second edition, 2005

[5] Prof. Dr. Markus Leippold
An option pricing tool, Semester Thesis in Quantitative Finance (2004)
Swiss Banking Institut, University of Zurich

[6] Paolo Brandimarte
Numerical Methods in Finance & Economics (2004)
John Wiley & Sons Publications, Second edition, 2006.

[7] Cox, Ross & Rubinstein
Option pricing: A simplified approach
The Journal of Financial Economics, Vol. 7, (July 1979), pp. 229 - 263

[8] Desmond J. Higham
Nine Ways to Implement the Binomial Method for Option Valuation in MATLAB
SIAM REVIEW, Vol. 44, No. 4, pp. 661677