Use of Numerical PDE Software for the Solution of a Classic Problem in Mathematical Finance

By

Jenna Young

A Thesis Submitted to

Saint Mary's University, Halifax, Nova Scotia

In Partial Fulfilment of the Requirements for

a Bachelor of Science, Honours Mathematics

April, 2014, Halifax, Nova Scotia

Approved: Dr. Paul Muir
Supervisor

Approved: Dr. Walt Finden
Reader

Approved: Dr. Bert Hartnell
Reader

Date: April 22, 2014

Use of Numerical PDE Software for the Solution of a Classic Problem in Mathematical Finance

by Jenna Young

Abstract

Mathematical modelling is an important part of the finance industry. These models can be very complex and one often needs to use numerical methods and numerical software packages to get approximate solutions to these models. In this thesis we use a high quality numerical software package called EPDCOL which is designed to solve systems of linear and non-linear partial differential equations (PDEs) and has temporal error control. Specifically, we will use it to numerically solve the Black-Scholes equation, a linear PDE, that can be used to value a financial instrument known as an option. An option is a contract which gives the holder the option to buy or sell a stock at a future time for an agreed upon price. A stock represents ownership of a corporation's assets and gives the opportunity to share in the corporation's earnings. EPDCOL implements a combination of high quality numerical methods which allow us to solve the Black-Scholes equation for two types of options (puts and calls) involving various parameters. Additionally, we are able to get an estimate of the spatial error associated with the numerical solutions.

April, 2014

1 Introduction

Mathematical modelling is an important part of the finance industry [14], [7]. Often the financial models being considered are very complicated and do not have closed form solutions that can be easily obtained by hand. People have often turned to numerical methods and numerical software packages to get approximate solutions to problems in finance.

High quality numerical software packages are usually developed over many years. Algorithms are made increasingly efficient and the packages are highly tested. However, it is not uncommon for professionals and students to develop their own software to solve problems in finance. These programs often do not measure up to the accuracy and efficiency of the numerical software packages. In this thesis we use a high quality numerical software package called EPDCOL [8] which is designed to solve systems of linear and non-linear PDEs and has temporal error control.

To be specific, in this thesis we numerically solve the Black-Scholes equation, a classic financial model. The Black-Scholes equation is used to value a financial instrument known as an option. An option is a contract which gives the holder the option to buy or sell a stock at a future time for an agreed upon price. A stock represents ownership of a corporation's assets and gives the opportunity to share in the corporation's earnings. The Black-Scholes equation is a PDE dependent on both the value of the stock or asset, S, and the current time, t. We will consider both European call and put options. European call options give the holder the option to buy the stock when the contract expires and European put options give the holder the right to sell the stock when the contract expires. One interesting consideration with European options is that they have a non-smooth initial condition which has to be taken into account when using EPDCOL.

EPDCOL implements a combination of high quality numerical methods. Firstly it uses B-spline collocation in the spatial variable to reduce the PDE(s) to a system

of ordinary differential equations (ODEs). (B-spline collocation involves representing the approximate solution as a linear combination of known piecewise polynomials, represented in terms of what are known as B-splines, with unknown coefficients; the coefficients are determined by requiring the approximate solution to exactly satisfy the PDE at a set of points across the spatial domain, where S, the asset price, is the "spatial" variable.) Then a high quality ODE solver is used to get a solution to the ODE system from which the final solution to the PDE can then be obtained. It is through the ODE solver that EPDCOL provides the adaptive temporal error control. As we have observed, the problem depends on both a spatial and temporal variable. This means that we have both a spatial and temporal contribution to the error. EPDCOL is only able to control the temporal error. However, it is possible to compute an estimate of the spatial error from which we can decide whether or not our computed solution is sufficiently accurate. To do this we compute a solution that is more accurate than the one we wish to consider and subtract from it the solution we are interested in. This yields an estimate of the error in our original solution.

We are able to solve the Black-Scholes equation for both call and put options involving various parameters. To do this, we have written a driver program that includes all of the parameters and subroutines required by EPDCOL. The driver program calls EPDCOL which solves the given problem and returns the computed solution. We have written the driver program so that the non-smooth initial condition does not cause an issue for EPDCOL. In this thesis we provide numerical results to show how high quality PDE software performs on this classic problem in finance.

The European option problem generalizes to the American option problem which is generally of more interest. American options are more complicated since the holder can exercise the option at any time leading up to the expiry. The work done in this thesis can be extended in a way that would allow us to consider the American option problem. There are software packages such as BACOL [11] that have both spatial and

temporal error control and may be able to treat both the European and American version of the problem.

In the next section of this thesis we give an introduction to the relevant finance concepts. We derive the Black-Scholes equation and discuss its initial and boundary conditions. In Section 3 we describe the numerical methods used by EPDCOL. We consider important parameters and subroutines included in our driver program and describe how to define them. In Section 4 we discuss the problem implementation and numerical results. We close with a Conclusion section where we indicate further directions of study.

2 Finance Background

The purpose of this chapter is to give the reader an overview of the important finance concepts that relate to this thesis. We will go into some detail about options, what they are, their properties and why they are important. A derivation of the Black-Scholes equation, for the modelling of call and put options, along with the relevant mathematical background will be presented. The primary source of this information is the textbook [14].

2.1 Introduction to Terms and Concepts

A stock is a type of security that shows ownership in a corporation and represents how much of the corporation's assets and earnings the stockholder owns. The company is owned by its shareholders and must pay them part of its profit if any is made. Shareholders are paid by the company in a quantity called a dividend whose value depends on how much profit the company makes. The value of a share depends on how much profit the investors think the company will make. The price at which shares are bought and sold on the stock market determines the stock's value.

The situation can, of course, be more complicated than simply buying and selling shares on a stock exchange. For example, complex contracts known as derivatives can be formed. A derivative is a type of security that allows investors to modify the contract to their specific wants and needs. An option, the financial instrument we are interested in, is a type of derivative. Mathematical modelling and theory are used to study derivatives.

2.2 Options: puts and calls

In this thesis we will consider the simplest type of options, European call and put options. The call option has the following specific conditions:

- There is a set time in the future, the **expiry date**, when the holder of the option *may* purchase the asset, which we call the **underlying asset** (or simply the **underlying**) for a set amount known as the **exercise price**.

The holder can, but is not obligated to, purchase the asset at the expiry date for the previously specified price. On the other side of the contract, the writer MUST sell the asset, at the previously agreed upon price, if the holder chooses to buy it. The value of the asset is associated with the fact that the holder has the option but no obligation to purchase the asset. The holder must pay for this option when the contract is opened.

Call options = the option to BUY an asset. If we are interested in purchasing an asset there are two things we can do. We can buy the asset outright or purchase a call option. When we buy a call option there are two things that can happen, either the price of the underlying asset will rise or fall in the future. For example, let's assume that we purchased a call option for which the exercise price is \$200, and that at expiry the price of the share has risen to \$240. This means the holder can exercise the option and obtain a share that is worth \$240 for only \$200. Then she can immediately sell

the share and make a profit of \$40. On the other hand, if the price of the share were to fall to \$160 at the expiration date the holder would not exercise the option. If she did she would be paying \$200 for a share that could be bought directly for \$160.

If the share has only value either \$160 or \$240 at the expiration date then the expected profit is:

$$(1/2)(\$0) + (1/2)(\$40) = \$20.$$

From this we conclude that the value of the option is \$20. If the share price is \$240 at expiry this means that the holder should have paid \$20 for the option and her net profit is:

$$\$40 - \$20 = \$20.$$

In this case her net profit was her up-front premium. On the other hand if the share price is less than \$200 at expiry, the price paid for the option is lost. We see that option prices respond in a drastic way to the underlying asset price, an effect that is defined as **gearing**.

The greater the share price at the expiry date, the greater the profit. We don't know the future share price in advance but we can predict that if the share price is higher today it will also be higher in the future. So, today's call option value depends on today's share price. The value of a call option also depends on the exercise price. A lower exercise price means we may possibly pay less than the market value for the share at expiry and thus the option value is higher. **Volatility** is the fluctuation of the price of the underlying asset between when the option is purchased and when the option is exercised. Volatility of the underlying asset impacts the price of a call option. We expect to pay more for an option on a volatile asset since the writer of the option does not know how the value of the asset will change before expiry. In

addition, the option price also depends on the interest rate since the option is paid for up front but payoff comes at a later time in the future. Thus the option price needs to account for the profit from investing the premium in the bank.

Put Options = the right to SELL an asset. The profit comes when conditions are opposite from those for a call option. A put option lets the holder sell the asset on a specific date for a set amount. At this point the writer MUST sell the asset. So the holder wants the asset price to fall as much as possible. If the exercise price is higher one gets more for the asset at expiry and thus the value of a put option increases as the exercise price increases.

2.3 The Financial Press

We can now read and understand the financial press generally given in tables that show information about option prices at given times on a given exchange. Let us consider Table 1 below showing the prices at expiry of options for the company BAA on the London International Financial Futures. This table is taken from the *Financial Times* on February 4th 1993; the prices we see for BAA are the option prices for options that expire on the coming third Friday in February, May and August of 1993.

		Calls			Puts		
Options		Feb	May	Aug	Feb	May	Aug
BAA	750	41	61	71	6	16	31
(^{o}786)	800	11	33	45	29	41	55

Table 1: Example of the BAA option on the London International Financial Futures [14].

On the left we see the acronym for the particular option and below, in brackets, we see the closing price (786). Then to the right there are two rows. In the first row, the number 750 represents an exercise price of the option. The six numbers to the right of 750 represent the call option and put option prices respectively (three of each) for expiry in February, March and April respectively. In the second row, we see

the same information for a BAA option with an exercise price of $800. The price of the option is increasing with time because we are less sure about how the option will behave in the future. We also note that call options cost less at higher exercise prices. Since a call option gives the holder the option to buy the asset at the exercise price, she is more likely to make profit when the exercise price is lower. We are less willing to buy an option with a higher exercise price and hence the cost is lower. Conversely if we hold a put option we are more likely to make a profit when the exercise price is higher. So a put option with a higher exercise price is worth more and hence costs more.

2.4 Two Main Uses of Options

2.4.1 Speculation: A use for holders of options

When deciding whether to buy an option we need to make some sort of prediction about the direction and magnitude of the movement in asset price. If we don't properly predict *both* the the direction and magnitude of the movement of the asset we can potentially incur a loss. On the other hand, if our predictions are good we can potentially obtain a return. Due to the nature of options, speculation is risky business.

2.4.2 Hedging: A use for writers of options

We note that holders of an option have the possibility to make a profit with their loss limited by the initial premium they paid on the option. On the other hand, writers of an option incur the possibility of a loss with their profit limited by the initial premium. This must mean that writers of options are expecting the price of the asset to fall. We can reduce the risk of a loss by hedging. When hedging, we construct a portfolio that contains both both put and call options for the same asset. This way if the price of the asset falls we don't incur a large loss due to the combination of both

puts and calls in our portfolio.

Take the following situation for example. We own shares in a company XYZ but we are worried about the short term performance of the company. We do still have confidence in the long term performance of the company though and so we do not want to sell our shares. Instead we buy put options for the company XYZ which will allow us to sell shares in the company. This way if the stock prices do fall the losses caused will be offset by the money we gain from the put options [1].

2.5 Other Types of Options

So far we have been discussing European options but these are not the only types of options available. Another important type of option is the American option. This option, unlike the European option, may be exercised at any given time before the expiry date. More options such as Exotic, Path-dependent, Barrier, Asian and Look-back options also exist. We will not go into detail about any of these types of options, except to say a little more about American options at the end of this chapter.

2.6 Asset Prices: A Simple Model

It is important to note that we are not trying to predict the asset price. As in most option pricing theory we do not know and cannot predict the price of an asset tomorrow or into the future. However, we can study past behaviour of an asset to get important information such as jumps in asset price, mean, variance and likely future distribution.

The **efficient market hypothesis** is often said to cause asset prices to move randomly. This hypothesis says two things:

- the present price of the asset fully reflects its past history
- any new information about an asset causes an immediate market response

Firstly we will note that knowledge of the absolute change of an asset price by itself is not useful. Instead we must associate every change with a return. The return is defined as the change in price divided by the original value and is denoted $\frac{dS}{S}$. This is the relative change in the asset price. If at time t we have an asset price S, we can consider a small time step dt, and suppose over this time period, dt, the price of the asset S changes to $S + dS$. To model the relative return of the asset $\frac{dS}{S}$ we begin by decomposing it into two pieces. Firstly, the **drift** μ which is the average rate of growth of the asset price and contributes the term μdt. Secondly we must consider the random change in the asset price. This term is σdX where σ is the **volatility** and dX is what is known as a **Weiner process** and is explained below. Together we get

$$\frac{dS}{S} = \sigma dX + \mu dt. \tag{1}$$

Taking $\sigma = 0$ in (1) gives an ODE which is easy to solve for S, the asset price, when μ is constant; this gives

$$S = S_0 e^{\mu(t-t_0)}$$

where S_0 is the value of the asset at the initial time $t = t_0$.

The term dX, known as a **Wiener process**, has three properties:

- dX is a random variable,
- the mean of dX is 0,
- the variance of dX is dt.

To clarify, a random variable is a numerical value that represents the random outcomes of an experiment. Exactly one random variable is assigned to each sample point from the experiment. The probability distribution of a random variable gives the

probabilities associated with each numerical value the random variable can have. A standardized normal distribution is the common bell-shaped curve representing the probability distribution of a random variable. This distribution must have mean 0 and variance 1 [10]. We let ϕ be a random variable drawn from a standardized normal distribution. Then one way of writing dX is

$$dX = \phi\sqrt{dt}. \tag{2}$$

As stated in [14], we scale dX by $\sqrt{dt}$ in (2) since without it, or with any other value, we get a meaningless or trivial problem as we consider the limit of dt as it goes to zero. We discuss this further in the next section. This standardized normal distribution with mean zero and unit variance has a probability density function given by

$$P(\phi) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\phi^2} \tag{3}$$

for real number ϕ. A probability density function is such that the area under it between a and b corresponds to the probability that ϕ will assume a value on [a,b]. The given function P(ϕ) is the standard notation of the probability function for a continuous random variable ϕ [10]. We define the expectation operator $E[\cdot]$ applied to a given function $F(\cdot)$ by

$$E[F(\cdot)] = \int_{-\infty}^{\infty} F(\phi)P(\phi)d\phi = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} F(\phi)e^{-\frac{1}{2}\phi^2} d\phi \tag{4}$$

for any function F that we choose. This is F(ϕ), multiplied by the corresponding probability and "summed" over all possible ϕ values. We can show using standard integration techniques and integration by parts that

$$E[\phi] = 0$$

and

$$E[\phi^2] = 1.$$

To see this first we note that the Gaussian Integral [13]

$$\int_{-\infty}^{\infty} e^{-\frac{1}{2}\phi^2} d\phi = \sqrt{2\pi}.$$

Then we have

$$E[\phi] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \phi e^{-\frac{1}{2}\phi^2} d\phi,$$

which we can solve using standard integration techniques. Also

$$E[\phi^2] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \phi^2 e^{-\frac{1}{2}\phi^2} d\phi.$$

We solve this by parts letting $u = \phi$ and $dv = \phi e^{-\frac{1}{2}\phi^2} d\phi$.

Equation (1) is called a **random walk**. It can give useful probabilistic information but cannot give a deterministic path for the share price. The random walk generates a different path each time it is restarted. Each new path that is generated is called a **realization** of the random walk. One important property of (1) is that the next asset price after some time step $S + dS$ depends on today's asset price, but not on any past information. This independence from the past is called the **Markov Property**. Figure 1 shows that random fluctuations of stocks.

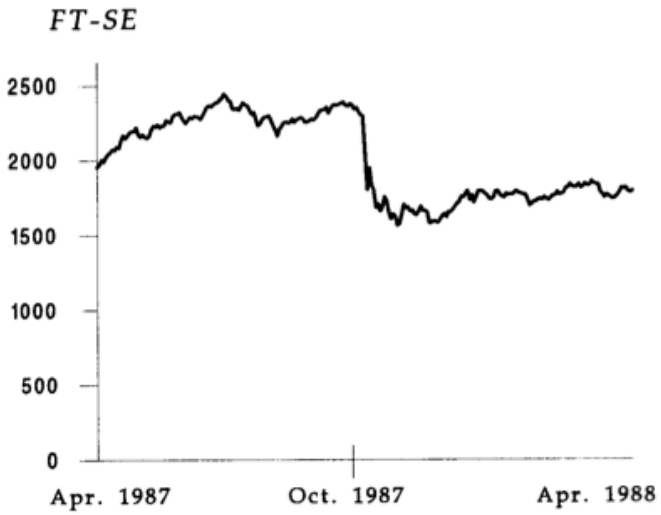

Figure 2.1. *FT-SE* closing prices from April 1987 to April 1988.

Figure 1: Graph showing the Financial Times Stock Exchange closing prices from April 1987 to April 1988 [14].

2.7 Itô's Lemma

Itô's lemma is a very important result regarding random variables that relates a small change in a function of a random variable to the variable itself. We are given that, with probability 1, as

$$dt \to 0 \text{ we have } dX^2 \to dt. \tag{5}$$

Consider a continuous function $f(S)$. Changing S by a small amount dS means that f also changes by a small amount df. This gives

$$\begin{aligned} f(S + dS) &= f(S) + df \\ \Rightarrow \; df &= f(S + dS) - f(S) \end{aligned}$$

Using a Taylor series expansion for $f(S+dS)$ about S we get

$$\begin{aligned} df &= f(S+dS) - f(S) \\ &= \left[f(S) + \frac{df}{dS}dS + \frac{1}{2}\frac{d^2f}{dS^2}dS^2 + \cdots \right] - f(S) \\ &= \frac{df}{dS}dS + \frac{1}{2}\frac{d^2f}{dS^2}dS^2 + \cdots \end{aligned} \tag{6}$$

But we know from (1) what dS is and we can square it to give

$$\begin{aligned} dS^2 &= (\sigma dX + \mu dt)^2 \\ &= \sigma^2 S^2 dX^2 + 2\sigma\ \mu\ dt\ dX + \mu^2 S^2 dt^2. \end{aligned} \tag{7}$$

Given how we defined dX in (2) we see that

$$dX = O(\sqrt{dt}).$$

This means that, considering (7), the first term dominates the other two for small dt giving

$$dS^2 \to \sigma^2 S^2 dX^2 + \cdots .$$

We can now make this substitution in (6) and use the definition of dS from (1) to get

$$\begin{aligned} df &= \frac{df}{dS}\left(\sigma S dX + \mu S dt\right) + \frac{1}{2}\sigma^2 S^2 \frac{d^2f}{dS^2}dt \\ &= \sigma S \frac{df}{dS}dX + \left(\mu S \frac{df}{dS} + \frac{1}{2}\sigma^2 S^2 \frac{d^2f}{dS^2} \right) dt. \end{aligned} \tag{8}$$

This result is what is known as Itô's lemma for $f = f(S)$ and we see that it relates a small change in the function of a random variable to a small change in the variable itself.

We would now like to further generalize this result in (8) by looking at a function of both the random variable S and of time t. We denote this function by $f(S,t)$. We want to consider a small change df in f arising from small changes in S and t, dS and dt respectively. This means

$$\begin{aligned} f(S+dS,t+dt) &= f(S,t)+df, \\ \Rightarrow\ df &= f(S+dS,t+dt)-f(S,t). \end{aligned}$$

Now we have two variables so we expand $f(S+dS,t+dt)$ about $f(S,t)$ in a double Taylor series expansion. To do this we do the Taylor series expansion first for $S+dS$ and then for $t+dt$. This yields the following

$$\begin{aligned} df &= \left[f(S,t)+\frac{d}{dS}f(S+dS,t+dt)+\frac{d}{dt}f(S+dS,t+dt)\right]-f(S,t) \\ df &= \left[\frac{\partial f}{\partial S}dS+\frac{1}{2}\frac{\partial^2 f}{\partial S^2}dS^2+\cdots\right]+\left[\frac{\partial f}{\partial t}dt+\frac{1}{2}\frac{\partial^2 f}{\partial t^2}dt^2+\cdots\right] \\ df &= \frac{\partial f}{\partial S}dS+\frac{\partial f}{\partial S}dt+\frac{1}{2}\frac{\partial^2 f}{\partial S^2}dS^2+\cdots \end{aligned}$$

Now we can use (1) to substitute for dS and (5) to substitute for dX^2 to get the following expression for df

$$df=\sigma S\frac{\partial f}{\partial S}dX+\left(\mu S\frac{\partial f}{\partial S}+\frac{1}{2}\sigma^2S^2\frac{\partial^2 f}{\partial S^2}+\frac{\partial f}{\partial t}\right)dt. \tag{9}$$

This result is Itô's lemma for $f=f(S,t)$

2.8 Eliminating Randomness

We have looked at two random walks that involve the random variable dX, the first in S, (1), and the second in f, (9). From here we can construct a third variable, call it g, with variation dg such that dg is wholly deterministic for the time step dt. *That*

is, dg does not depend on the random variable dX. To do this we define

$$g = f - \Delta S,$$

where Δ, to be specified shortly, is constant over the whole time step *dt.* Then we consider *dg*:

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

Next we choose $\Delta = \frac{df}{dS}$ (this means we assume $\frac{df}{dS}$ is constant over the time step *dt*). We notice that by constructing *g* in this way we can remove the *dX* term from *dg.* By making this substitution we have

$$dg = df - \left(\frac{df}{dS} \right) dS = \left(\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + \frac{\partial f}{\partial t} \right) dt,$$

and we note that *dg* does not depend on *dX*.

2.9 Option Values and Payoffs

We next define some notation that will be used throughout the rest of this thesis.

- Let *C(S,t)* and *P(S,t)* denote the value of a call option and put option respectively for an asset S and time *t.*
- Let σ denote the volatility.
- Let E denote the exercise price.

- Let T denote the expiry time.
- Let r denote the interest rate.

We shall first consider the value of a call option at time $t = T$, that is, just at the moment of expiry. If we have $S > E$ we can exercise the option and pay an amount E for an asset that is worth S. A profit of S - E has been made. If we have $S < E$ we will not exercise the option because we would incur a loss of E - S. From here we get that the value of a call option is

$$C(S, t) = \max(S - E, 0). \tag{10}$$

Graphing this in Figure 2 we have the **payoff diagram** for a call option shown by the bold line and the value of the call option prior to expiry shown by the faint line.

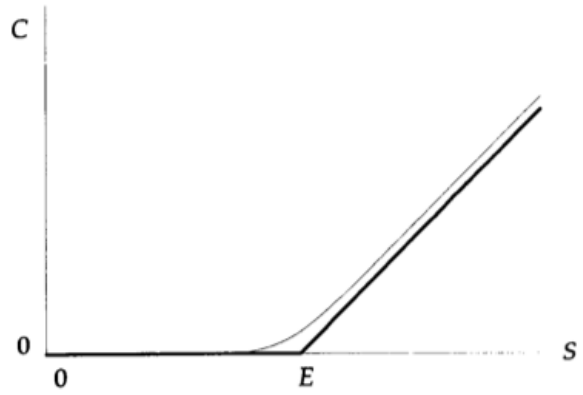

Figure 2: The payoff diagram for a call option and the option value prior to expiry [14].

For the payoff diagram of a put option we follow similar logic. We know that the profit we can make is E - S if $E > S$ and otherwise we make no profit. When the exercise price is more than the value of the asset at expiry it means that the writer of the put option will receive an amount E for an asset that is only worth an amount S. In the other case when $E < S$ at expiry the put option is worthless since the holder

of the option would be receiving an amount E for an asset that is worth a greater value S. In this case she could just sell the asset directly for its value S. So in this case the value of the put option is

$$P(S,t) = \max(E - S, 0).$$

In Figure 3 we have the **payoff diagram** for a put option shown by the bold line and the value of the put option prior to expiry shown by the faint line.

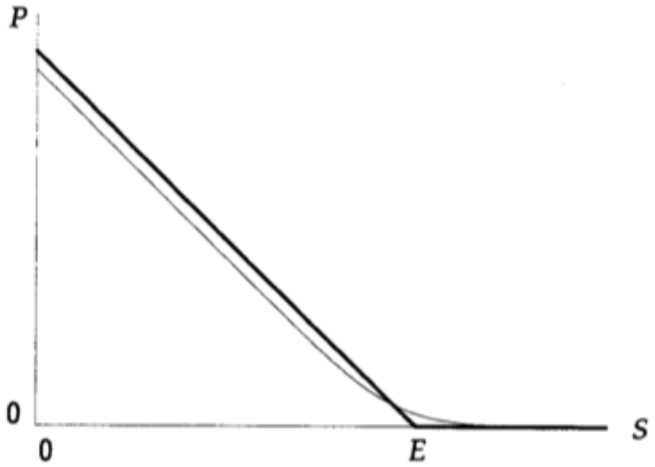

Figure 3: The payoff diagram for a put option and the option value prior to expiry [14].

2.10 The Correlation Between Call and Put Options

In this section we will demonstrate the correlation between put and call options. Let Π denote a portfolio. Suppose that in our portfolio we are set to buy one asset S, buy one put option, P, and sell one call option, C. The put and call options have the same expiry date, T, and the same exercise price, E. If we consider the value of the portfolio Π to be a combination of the value of the assets and options in it then we

can describe it in the following way

$$\Pi = S + P - C.$$

The payoff of such a portfolio at time T is

$$S + \max(E - S, 0) - \max(S - E, 0).$$

Depending on whether S $\geq$ E or E $\geq$ S we have the following:

$$S + 0 - (S - E) = E, \quad \text{if S} \geq \text{E},$$

or

$$S + (E - S) - 0 = E, \quad \text{if E} \geq \text{S}.$$

Now what we want to know is how much we should pay for such a portfolio at expiry, i.e., at time $t = T$. Since we will assume that interest rates are constant, our portfolio has the value Π(t) and grows, in a relative sense, according to

$$\frac{d\Pi}{\Pi} = r dt, \tag{11}$$

the same rate as it would if we deposited the same amount into a bank with interest rate r. This is a differential equation with solution

$$\Pi = ce^{rt},$$

where c is a constant. Since, at expiry, we have the terminal condition $\Pi(t) = E$, this gives a value of $c = \frac{E}{e^{rT}}$. We then have that the value of the portfolio at time t before

$t = T$ is

$$S + P - C = Ee^{-r(T-t)}. \tag{12}$$

This relationship between the asset S and its put and call options is what we call the **put-call parity** of the asset.

2.11 Black-Scholes Analysis

Before beginning this analysis we will list some assumptions made in [14] that we will make throughout the remainder of this thesis:

- The asset price follows (1).
- Both r and σ are known and constant over the life of the option.
- The underlying asset does NOT pay dividends.
- There is no possibility of arbitrage. This means that there is no opportunity for us to take advantage of different prices between markets to make risk-free returns.
- There are no transaction costs associated with the buying and selling of assets. Transaction costs are the added costs incurred when buying or selling an option. They could be things such as market research into whether or not to buy the option or the cost of negotiating with the buyer or seller.
- We can continually trade the underlying asset.
- Short selling is allowed and assets are divisible. Selling short means selling an option that we may not actually own yet. Assets being divisible means that we can buy or sell any number (even a non-integer amount) of the underlying asset.

We will now consider a portfolio *V(S,t)* that can contain both puts and calls. To simplify we can think of *V(S,t)* as containing just one simple call or put. Using Itô's lemma and replacing f with V in (8) we get

$$dV = \sigma S \frac{\partial V}{\partial S} dX + \left(\mu S \frac{\partial V}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} \right) dt. \tag{13}$$

We can now construct a portfolio of this one option V and some amount, Δ, of the underlying asset S. We are allowed to consider this value Δ of S since in one of our assumptions we assumed that the underlying asset is divisible. We also note that constructing our portfolio this way will allow us to use the elimination of randomness technique introduced in section 2.8; we have

$$\Pi = V - \Delta S. \tag{14}$$

We can now consider the change in the value of the portfolio over one small time step dt:

$$d\Pi = dV - \Delta dS. \tag{15}$$

We next substitute (1), (12) and (13) into (15) to get

$$d\Pi = \sigma S \left(\frac{\partial V}{\partial S} - \Delta \right) dX + \left(\mu S \frac{\partial V}{\partial S} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} - \mu \Delta S \right) dt. \tag{16}$$

Now we can use the elimination of randomness technique that we discussed earlier. We will let

$$\Delta = \frac{\partial V}{\partial S} \tag{17}$$

giving a wholly deterministic value for the portfolio after one time step. Simplifying we get

$$d\Pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt. \tag{18}$$

Since there is no arbitrage possibilities and no transaction costs associated with trading assets, the return that we would make on an investment of size Π after some time step dt is $r\Pi dt$ as given by (11). To see this consider the following argument. If the right hand side of (18) was more than this amount, an investor could make a guaranteed risk free profit by borrowing an amount Π from the bank and investing in the portfolio. On the other hand, if the right hand side was less than this amount, the investor would instead invest the amount Π in the bank. In either case the investor would make a guaranteed risk free profit (which we have assumed to be impossible) so the return on the portfolio and the return from a risk free bank account must be equal. Hence,

$$r\Pi dt = \left(\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt. \tag{19}$$

By substituting (14) and (17) into (19) we obtain the **Black-Scholes PDE**,

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0. \tag{20}$$

The Black-Scholes equation is very important as, under the assumptions made, any option that is paid for up front and that depends only on S and t must satisfy this equation.

2.12 Black-Scholes Final and Boundary Conditions

The Black-Scholes equation is what we call a backward parabolic PDE. The term parabolic means that the highest derivative with respect to S is a second derivative and the highest derivative with respect to t is a first derivative. It is said to be a backward parabolic PDE since it is linear and the signs with respect to these derivatives are the same when on the same side of the equation (see [14] pg 45). A PDE itself can have many different solutions so we need to include some other conditions to get the specific desired solution. Since the variable S appears with both first and second derivatives, we need two conditions; since the variable t appears only with a first derivative, we only need to impose one more condition. In our case this time condition is a final condition at time $t = T$ and we solve the problem in the region where $t < T$. In most PDEs the time condition is imposed as an initial condition at time $t = 0$. We note that this is not an issue since we can change from moving forward to moving backward in time by a change of variable.

Next we will consider the specific conditions we need to place on European call and put options. Let's first look at a call option *C(S,t)* with exercise price E and expiry date T. The final condition is imposed at time $t = T$ and, as we discussed in Section 2.11, the value of a call option is known to be the payoff:

$$C(S, T) = \max(S - E, 0). \tag{21}$$

Now we need to consider the boundary conditions applied to the call option. The first is applied at S = 0 and the second as S $\to \infty$. First we can note that if S is zero then by (1) dS is also zero and the value of S can never change. Hence if $S = 0$ then we obtain the left boundary condition at $S = 0$:

$$C(0, t) = 0, \text{ for } S = 0. \tag{22}$$

As the price of an asset increases with no upper bound it becomes increasingly likely that the option will be exercised since as the price gets larger the exercise price becomes less meaningful. Thus we have the right boundary condition as $S \to \infty$:

$$C(S,t) \sim S \quad \text{as} \quad S \to \infty. \tag{23}$$

Similarly for a put option the payoff at time $t = T$ is:

$$P(S,T) = \max(E - S, 0). \tag{24}$$

This is the final condition for a put option. Now we will consider the boundary conditions for $S = 0$ and as S $\to \infty$. As previously stated, if S is ever zero it will remain zero and hence we will make a profit of E. To find $P(0,t)$ we need to know the present value of E that will be received at time T. As explained in Section 1.12 we get the value of the put with $S = 0$ to be:

$$P(0,t) = Ee^{-r(T-t)}, \text{ for } S = 0. \tag{25}$$

This gives the left boundary condition. As $S \to \infty$ we become less and less likely to exercise the option because we have less chance to make a profit so the right boundary condition is:

$$P(S,t) = 0 \quad \text{as} \quad S \to \infty. \tag{26}$$

2.13 The Exact Solution to the Black-Scholes Equation for European Options

It turns out there is a closed form solution for the value of European call and put options when the interest rate, r and the volatility, σ are constant. The exact solution for a call option is

$$C(S,t) = SN(d_1) - Ee^{-r(T-t)}N(d_2) \tag{27}$$

and for a put option the exact solution is

$$P(S,t) = Ee^{-r(T-t)}N(-d_2) - SN(-d_1) \tag{28}$$

where *N(·)* is the cumulative distribution for a standard normal random variable. The cumulative distribution for a standard normal random variable $N(x)$ describes the probability that a random variable X assumes a value in the range [0,x]. It is given by

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy.$$

The quantities d_1 and d_2 are given by:

$$d_1 = \frac{\log(S/E) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

and

$$d_2 = \frac{\log(S/E) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}.$$

If we want to consider the value of Δ to eliminate randomness for a European call option we use

$$\Delta = \frac{\partial C}{\partial S} = N(d_1) \tag{29}$$

and for a European put option we use

$$\Delta = \frac{\partial P}{\partial S} = N(d_1) - 1. \tag{30}$$

2.14 Generalizing to the American Option

American options differ from European options in one important way. A European option has a specific expiry date but an American option can be exercised at any time up to and including expiry. This means that for the American option we cannot just compare the asset price to the exercise price at the expiry date. There is a whole range of prices that the asset can assume. This property means that the American option problem is what we call a free boundary problem. Not only do we have to determine the value of the option but we also have to determine whether or not the option should be exercised for any given value of S. Typically for each time t there is some value of S that determines the boundary condition. Being on one side of the S value will imply that the option should be exercised and being on the other side will imply that the option should not be exercised and therefore held until a later period in time. We have what we call a free boundary because we do not know ahead of time where the boundary condition should be applied. We will not consider this option type further in this thesis.

3 Numerical Methods

In this section we will discuss the numerical methods that are implemented in the software package that we used to obtain numerical solutions to the PDE models discussed in the pervious chapter. This software package, called EPDCOL, is designed to solve systems of linear or nonlinear PDEs. The EPDCOL code is based on a numerical approach called the method-of-lines. EPDCOL uses collocation techniques (to be described shortly) for the treatment of the spatial variable x, or in our case S, which reduces the PDE to a system of ODEs. Then EPDCOL computes an approximate solution to the system of ODEs and from there an approximate solution to the original PDE. Also of interest are the linear algebra algorithms that are used within the package to solve the systems of linear equations that arise. The EPDCOL code is a modified version of a widely used package called PDECOL. The main difference between the two codes is the linear algebra software that is used. In EPDCOL superior linear algebra software is employed to save both time and storage space.

3.1 PDE System Structure

As described in [9], PDECOL uses the method-of-lines approach with *i = 1,2, . . . , N*, nonlinear PDEs of second order or less, over an interval $[x_L,x_R]$ in the x domain for values of $t \geq t_0$. PDECOL assumes the following form of the PDE system:

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{f}(t, x, \mathbf{u}, \mathbf{u}_x, \mathbf{u}_{xx}) \quad \text{where } \mathbf{f} = (f_1, f_2, \ldots, f_N) \tag{31}$$

and where

$$\mathbf{u} = (u_1, u_2, \ldots, u_N), \quad \mathbf{u}_x = \left(\frac{\partial u_1}{\partial x}, \frac{\partial u_2}{\partial x}, \ldots, \frac{\partial u_N}{\partial x}\right), \quad \mathbf{u}_{xx} = \left(\frac{\partial^2 u_1}{\partial x^2}, \frac{\partial^2 u_2}{\partial x^2}, \ldots, \frac{\partial^2 u_N}{\partial x^2}\right).$$

Since the order of the PDEs can vary, PDECOL needs either 0, 1 or 2 boundary conditions for each of the PDEs. These boundary conditions are imposed at x_L and/or x_R for each equation and are of the form

$$b(\mathbf{u}, \mathbf{u}_x) = z(t), \tag{32}$$

where $z(t)$ is some known function. An initial condition for each solution component u_i, i = 1,2, ..., N at the initial time $t = t_0$ is also required. These initial conditions have the form

$$u_i(x, t_0) = \alpha_i(x)$$

where $\alpha_i(x)$ is some known function of x.

The EPDCOL software package was obtained through a modification of PDECOL. The paper [8] claims that PDECOL solves PDE systems of only second order. We also note that all examples in both [8] and [9] treat only second order PDEs. The interface for PDECOL allows for PDEs that involve only $\mathbf{u}_x$ (hyperbolic PDEs) or only $\mathbf{u}$ (ODEs), but the use of the software, to our knowledge, has only been for the case of second order PDEs. EPDCOL assumes that the PDEs are all of second order. This gives the same system of PDEs from (31) and the same initial conditions but the boundary conditions can be specified more precisely. The boundary conditions are imposed at the right and left hand endpoints of the spatial interval for each PDE in the system. This gives the following conditions

$$b_{x_L,i}(\mathbf{u}, \mathbf{u}_x) = z_{L,i}(t), \quad b_{x_R,i}(\mathbf{u}, \mathbf{u}_x) = z_{R,i}(t),$$

where i = 1,2, ..., N, and $z_{L,i}(t)$ and $z_{R,i}(t)$ are known functions of time.

To perform the spatial discretization EPDCOL uses collocation with piecewise

polynomials. After the discretization the system of PDEs is represented by a system of ODEs depending only on the time variable t which can then be solved to get an approximate solution to the problem.

3.2 The Method-of-Lines and Collocation

EPDCOL represents the approximate solution to the PDE as a piecewise polynomial. A piecewise polynomial is made up of many individual polynomials that are joined together at a set of mesh points [4]. In the case of EPDCOL, the number of mesh points is decided by the user upon input. The user is able to input the number of subintervals, m, into which the domain $[x_L,x_R]$ is partitioned. This gives $m+1$ mesh points in total. The user also specifies these mesh points, which are stored in an array of strictly increasing order with the following property

$$x_L = x_1 < x_2 < \cdots < x_m < x_{m+1} = x_R.$$

The degree of the piecewise polynomial is also specified by the user upon input and is defined as the value $k+1$ where k is the degree of the polynomial. The user can also specify the continuity of the piecewise polynomial. Let c be the continuity of the polynomial pieces at each internal mesh point. For the case $c = 2$ (which is generally the case with EPDCOL since the PDEs are of second order) the solution approximation that is constructed consists of piecewise polynomials that are continuous and also have a continuous first derivative.

EPDCOL implements the piecewise polynomial approximate solution using B-spline basis functions. These are piecewise polynomials that together span the required linear piecewise polynomial space, and are of degree k and continuity c, over the mesh $\{x_i\}_{i=1}^{m+1}$. The piecewise polynomial space has finite dimension given by the parameter *d = (k+1)m - c(m-1).* For given k, m, and c values, as indicated by the

user, EPDCOL constructs the appropriate B-spline basis for the piecewise polynomial space [4]. This B-spline basis consists of d known piecewise polynomial functions that we will call $\Phi_j(x)$, for $j = 1, 2, \ldots, d$.

EPDCOL represents the approximate solution as a linear combination of the B-spline basis functions. At any time t, an approximate solution component u_i is a piecewise polynomial written in terms of the B-spline basis functions as follows

$$u_i(t, x) = \sum_{j=1}^{d} c_{i,j}(t)\Phi_j(x), \quad i = 1, 2, \ldots, N. \tag{33}$$

Here the coefficient $c_{i,j}$ is unknown and depends only on t and $\Phi_j(x)$ is a known B-spline basis function and depends only on x. In order to find these unknown coefficients, EPDCOL requires the approximate solution component u_i in (33) to satisfy the original system of PDEs in (31) and the boundary conditions in (32) at a specific set of d collocation points. EPDCOL chooses the collocation points such that,

$$x_L = v_1 < v_2 < \cdots < v_d = x_R.$$

By substituting (33) into (31) EPDCOL obtains the following collocation equations at the interior collocation points

$$\sum_{j=1}^{d} \frac{dc_{i,j}(t)}{dt}\Phi_j(z_l) = f_i(t, z_l, \mathbf{u}(t, z_l), \mathbf{u}_x(z_l), \mathbf{u}_{xx}(z_l)),$$

where $l = 2, 3, \ldots, d-1$, and $i = 1, 2, \ldots, N$. To be more specific, the collocation points are chosen to be the endpoints of the problem domain x_L, x_R, and $k+1$ points on each subinterval which are the images of the Gauss points [4] mapped onto each subinterval.

This method-of-lines algorithm with collocation in the spatial variable leads to an

approximation of the system of PDEs by a system of ODEs. We must also consider the boundary conditions. We notice that they are algebraic equations and not ODEs in time. However the ODE solver used by EPDCOL requires that all of the equations are ODEs. This means that the boundary conditions must be differentiated with respect to time to obtain ODEs rather than algebraic equations. Thus the user is required to provide subroutines that give the time derivatives of the boundary conditions.

3.3 Linear Algebra Techniques

In this section we consider the system of linear equations that arises during the process of computing an approximate solution to the system of PDEs using EPDCOL. In PDECOL, the systems of linear equations that arose were solved using a band matrix solver. A band matrix is one where the nonzero entries are located in a band along the main diagonal of the matrix. However with further investigation it was noticed that, due to the properties of B-splines and the value used for c, the linear systems that arise have what is known as an almost block diagonal (ABD) structure [2]. A matrix with an ABD structure is one where the nonzero elements are within a pair of blocks, of potentially different sizes, along the main diagonal of the matrix, except for the top and bottom block rows that have only single blocks. In addition any column intersects no more than two successive blocks and the overlap between blocks is not necessarily constant [2].

There is a subroutine COLROW [5] that is specifically designed for the efficient solution of ABD linear systems. This subroutine uses both row and column elimination. Unlike the methods used in PDECOL, COLROW employs an algorithm that introduces no fill-in during the elimination process. (Fill-in happens when zero elements of a matrix become non-zero elements during the elimination process.) This is an interesting observation that leads to a saving of 50 percent in both execution time and storage requirements for the tested cases [8].

The following are images taken from [8]. Figure 4 is an ABD matrix where the degree of the piecewise polynomials is 4 and the number of continuity conditions is 2. Figure 5 is the banded matrix that would have been considered in the PDECOL code.

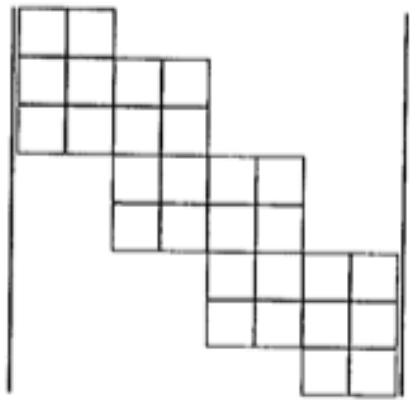

Figure 4: An ABD matrix with $k = 4$ and $c = 2$ [2].

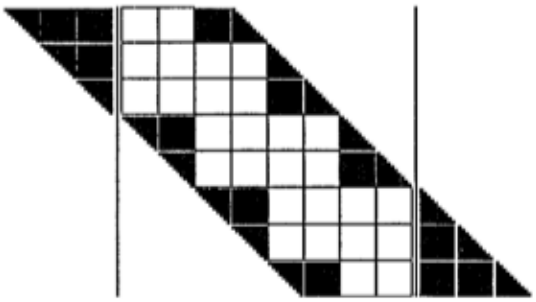

Figure 5: The banded matrix representation for the same system [2].

The computations performed by COLROW inside EPDCOL represent the dominant cost within the EPDCOL algorithm. For a system with 1 PDE, like the problem we are solving, we have that the total cost is $O(m \times k^3)$. In general for a system with i = 1, 2, $\cdots$, N, PDEs the error is $O(m(k \times N)^3)$ [3], [6].

3.4 EPDCOL Parameters and Subroutines

In this section we briefly describe the EPDCOL user interface. The user is tasked

with declaring parameters and writing subroutines that EPDCOL needs in order to solve a given problem.

3.4.1 Parameters

We begin by giving EPDCOL a system of i =1, 2, ..., N = NPDE nonlinear PDEs over the x domain $[x_L, x_R]$. Through the variable m = NINT the user inputs the number of subintervals into which $[x_L, x_R]$ is partitioned. The NPTS = NINT+1 mesh points are stored in the array XBKPT; this array is passed to EPDCOL and satisfies the following property

$$x_L = XBKPT(1) < XBKPT(2) < \cdots < XBKPT(NINT + 1) = x_R.$$

The variable KORD specifies the degree of the piecewise polynomials that will be used by EPDCOL. KORD is defined as $k + 1$ where k is the degree of the piecewise polynomials. The user can change the number of continuity conditions that EPDCOL will impose on the solution by changing the variable c = NCC. The piecewise polynomial space in which the approximate solution is represented has dimension d = NCPTS = KORD*NINT - NCC*(NINT-1). The parameter NCPTS also denotes the number of collocation points that are used by EPDCOL.

Other parameters important to EPDCOL are TO, the initial time, DT, the initial time step, EPS, the time tolerance and MF, the method flag. We will also set r, the interest rate, σ, the volatility, E, the exercise price and TOUT, the expiry time. The latter set of parameters have values determined by the mathematical description of the problem.

3.4.2 Subroutines

There are a number of subroutines that the user must also provide to EPDCOL. The

subroutine F is where we describe the PDEs that we wish to solve. In the subroutine BNDRY we specify the derivatives of the boundary conditions where DBDU and DBDUX correspond to the left hand side of the boundary conditions and the derivatives respectively and DZDT corresponds to the right hand side of the boundary conditions given in (32). The subroutine UINIT is where we specify the initial conditions for the problem. In the optional subroutine DERIVF the user can supply the analytic Jacobian of the functions defined in the subroutine F. After we have correctly chosen and defined all of the necessary information described above, the driver program is used in combination with EPDCOL to compute numerical solutions to the system of PDEs that we have described.

3.5 Black-Scholes Driver Program

In this section we will discuss the software that we wrote specifically for the Black-Scholes problem we are considering. This driver program specifies all of the relevant parameters and subroutines and then calls EPDCOL to compute the approximate solution. The code for this driver program can be found in the appendix of this thesis. The following explains how values were assigned to the parameters described in the above section.

3.5.1 Choosing the Parameter Values for the Driver Program

Both NINT and KORD are chosen by the user when the program is run and can be anything within EPDCOL's accepted range of values. For KORD this is a value larger then 2 and smaller than 21. The only restriction on NINT is that it must be at least 1. The rest of the parameters that the user can control are defined within the driver main program. We specify NPDE, the number of PDEs to be 1 (the Black-Scholes equation) and NCC to be 2. TO is the initial time and we begin at the current time TO = 0.0. The size of the initial time step size can be modified by the user through

the parameter DT and for this problem it was set to 1.0×10^{-7}. The time tolerance EPS was determined experimentally to be 1.0×10^{-5}. These experimental details are explained in the next chapter of this thesis. The method flag MF is associated with the numerical methods used for the time integration and for this problem is set to 21. The time of expiry for the option, TOUT, is set to 1.0 and was chosen based on the problem description and examples shown in [14]. For the specific problem we are solving we choose $r = 0.1$, $\sigma = 0.2$ and $E = 1.0$ but note that these parameters can easily be changed to solve a different instance of the Black-Scholes equation.

3.5.2 Problem Dependent Subroutines

Now we discuss how we wrote the subroutines required by EPDCOL. To reiterate, the following is the general Black-Scholes equation

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

We will remember that the Black-Scholes equation is a backward parabolic PDE with a final condition, as opposed to an initial condition. A linear parabolic PDE is backward if signs with respect to the derivatives in S and t are the same when on the same side of the equation. Otherwise it is a forward parabolic PDE. An issue that arises here is that EPDCOL solves initial value systems and so we need to give it a forward equation, not a backward equation. To do this we make the substitution $t = -t$. This gives the following forward parabolic equation

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

We rearrange it in the following way so that it is in a form that can be used by

EPDCOL

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

Once we have the equation in this form we let $V = U$ and $S = X$. Then we substitute these values into the Black-Scholes equation and write the following equation in the subroutine F

$$FVAL(1) = 0.5^*SIGMA^{**}2^*X^{**}2^*UXX(1) + R^*X^*UX(1) - R^*U(1).$$

Next we will consider the boundary conditions. As we described in Section 2.14, in (21) and (22), the boundary conditions (for a European call option) are

$$C(0,t) = 0 \quad \text{and} \quad C(S,t) \sim S \quad \text{as} \quad S \to \infty.$$

Since we know that EPDCOL considers boundary conditions of the form

$$b(\mathbf{u}, \mathbf{u}_x) = z(t)$$

we write the above boundary conditions in the following way

$$At\ x_L = XBKPT(0), \quad U(1) = 0,$$
$$At\ x_R = XBKPT(NINT + 1), \quad U(1) - X = 0.$$

This means that in the subroutine BNDRY we have the following:

$$\begin{aligned} At\ x_L = XBKPT(0): \quad & DBDU(1,1) = 1.0, \\ & DBDUX(1,1) = 0.0, \\ & DZDT(1) = 0.0, \end{aligned}$$

and

$$\begin{aligned} At\ x_R = XBKPT(NINT+1): \quad & DBDU(1,1) = 1.0, \\ & DBDUX(1,1) = 0.0, \\ & DZDT(1) = 0.0, \end{aligned}$$

where DBDU(1,1) is the partial derivative of the left hand side of of the boundary condition with respect to U(1), DBDUX(1,1) is the partial derivative of the left hand side of the boundary condition with respect to $U_x(1)$ and DZDT is the derivative of the right hand side of the boundary condition with respect to t.

Next we can consider the initial condition, or in our case the final condition, specifically in terms of the variables used in our example problem. We know that the final condition at expiry time T is

$$C(S,T) = \max(S - E, 0).$$

In terms of our variables, where $S = X$, we have the following initial condition that we write in the subroutine UINIT

$$U(1) = \max(X - E, 0.0).$$

We can also input the analytic Jacobian by taking the partial derivatives of FVAL(1) with respect to U, U_x and U_{xx}. This gives the following set of equations that we write in the subroutine DERIVF

$$DFDU(1,1) = -R$$

$$DFDUX(1,1) = R^*X$$

$$DFDUXX(1,1) = (0.5)^*SIGMA^{**}2^*X^{**}2.$$

4 Problem Implementation and Numerical Results

Now that we understand how EPDCOL works and the kinds of systems it can solve, we will use it to get an approximation to the value of European call and put options. This means that we will use EPDCOL to compute an approximate solution to the Black-Scholes equation. Note that all computations are performed on a Linux 2.6.32-52-server #114-Ubuntu SMP using the compiler GNU Fortran (Ubuntu 4.4.3-4ubuntu5.1) 4.4.3. Based on what has been studied and explained in the second section of this paper we know how European options behave and how they are modelled by the Black-Scholes equation. We were also able to determine the boundary and initial conditions for European options. Below is the graph showing the value of a European call option, with the parameters $r = 0.1$, $\sigma = 0.2$ and $E = 1.0$ at different times before its expiry. Our goal is to obtain numerical results that agree with those given in Figure 6.

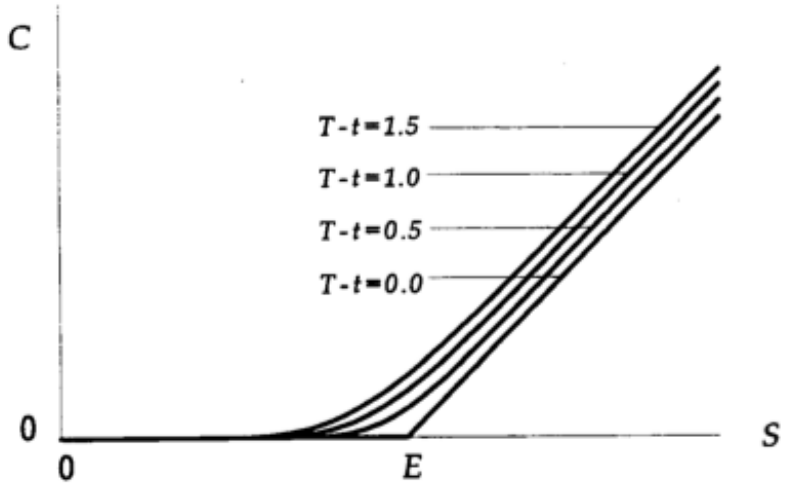

Figure 6: The value of a call option at various times before expiry with $r = 0.1$, $\sigma = 0.2$ and $E = 1.0$ [14].

To begin the numerical experiments we will follow the example in Figure 6. That means we will have $r = 0.1$, $\sigma = 0.2$ and $E = 1.0$. We first run EPDCOL on a standard instance of the problem where we set $x_R = 20$, TOUT = 1.0, EPS = 1.0×10^{-6}, KORD = 4, NINT = 40 and a uniform mesh. Recall that the left boundary is fixed at x_L while the right boundary theoretically goes to ∞ and thus has to be approximated by a"large" number, in this case $x_R = 20$. TOUT is the output time, EPS is the temporal error tolerance, NINT is the number of subintervals in the spatial mesh and KORD - 1 is the degree of the B-spline basis functions. The graph of the numerical solution computed by EPDCOL is given below in Figure 7 and Figure 8 shows the graph zoomed in to the region [0,2].

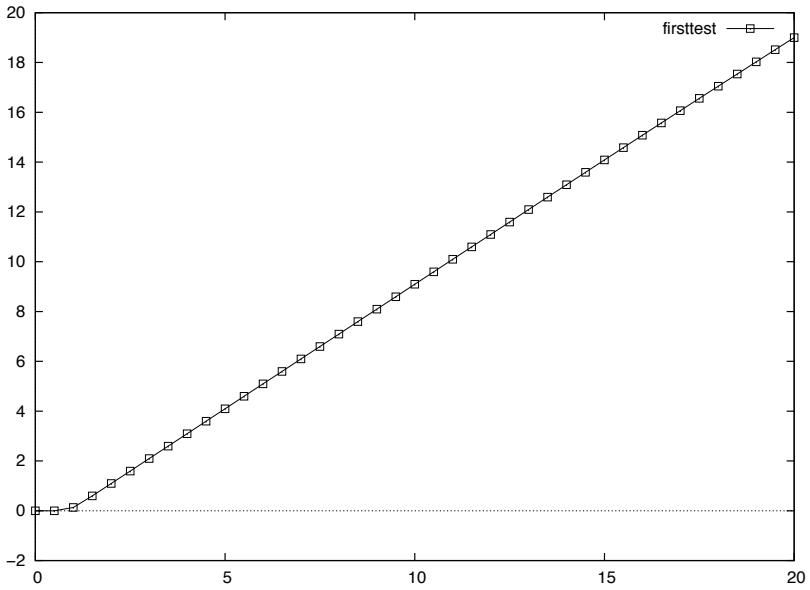

Figure 7: Graph of the solution to the Black-Scholes problem on [0,20] where $r = 0.1$, $\sigma = 0.2$ and $E = 1$ at time TOUT = 1.0.

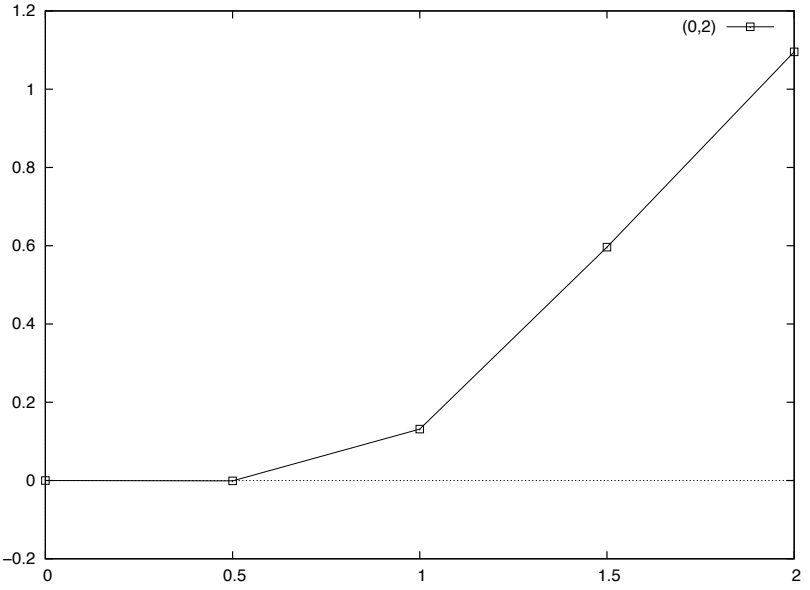

Figure 8: Graph of the solution to the Black-Scholes problem on [0,2] where $r = 0.1$, $\sigma = 0.2$ and $E = 1$ at time TOUT = 1.0.

From these figures we can see that on a basic level, EPDCOL is giving suitable results. We now disuss a number of issues associated with the computation.

4.1 Parameter Choice Issues

Now we move on to consider the problem dependent and software dependent parameters that may cause issues for EPDCOL or may impact on the accuracy of the solution. The values that we chose above for the parameters x_R, TOUT, EPS, NINT, and KORD and the use of a uniform mesh can all affect the accuracy of the solution that EPDCOL computes.

The first parameter we consider is the value of T, the expiry time, that we will use in our calculations. In EPDCOL this is the variable TOUT that we must set. We see in Figure 6 the value of a call option at a few different times before expiry. From here we can conclude that setting TOUT to 1.0 is reasonable. It will give us a snapshot of the value of an option at time TOUT = 1.0 before expiry. We can always change TOUT if we wish to see the value of an option at some different time before it expires.

The next parameter to consider is the right hand endpoint of the problem domain x_R. As the boundary conditions are defined in (23) we see that the left boundary condition is always imposed at $x_L = 0$ and thus does not cause any issues for EPDCOL. The right hand boundary condition, though, is imposed (theoretically) as x_R approaches infinity. This is an issue from a computational standpoint since we have to choose x_R to be finite. This could potentially limit the accuracy of the problem because we are forced to restrict the domain. We discuss the choice of x_R in the next section.

The next parameter that we need to consider is EPS, the time tolerance. This parameter controls the time error in the solution at every time step. The software package PDECOL, from which EPDCOL was developed, uses time integration meth-

ods that use an adaptively chosen time step sequence and integration formula to efficiently solve the problem to within the user specified time tolerance of EPS [9]. The issue here is that we want EPS to be strict enough that our solutions are sufficiently accurate but not so strict that the run time of the program will be significantly increased. Setting a strict tolerance causes EPDCOL to spend increased time trying to compute a solution that satisfies the given tolerance.

There are two more parameters that also affect accuracy. The first is the parameter KORD $= k +1$ where k is the degree of the piecewise polynomials being used. We want to choose a large enough degree that the error is reasonably small but not so large a degree that EPDCOL's runtime is greatly increased. We also realize that we have a non-smooth initial condition which does not have a first derivative at $S = E$. Hence using high degree piecewise polynomials in that area will not give more accurate results there and high degree piecewise polynomials will be less efficient.

The second parameter associated with the accuracy versus computational time tradeoff is NINT, which is used to tell EPDCOL how many subintervals the spatial domain should be partitioned into. Again, we want enough subintervals that EPDCOL computes a sufficiently accurate solution but not so many that EPDCOL will take too long to run.

During the collocation process EPDCOL uses a set of points called Gauss points. We note that the Black-Scholes equation is a 1D parabolic PDE and that analysis in [3] and [6] has investigated the spatial error associated with Gaussian collocation applied to this type of PDE. The analysis states that the spatial error is $O(h^{k+1})$ where k is the degree of the polynomial (which implies $k-1$ collocation points) and h is the maximum spatial subinterval size. Associated with the choice of NINT is the type of mesh we are using. That is, we can have a uniform or nonuniform mesh depending on whether the subintervals are all of the same size or not. EPDCOL conveniently lets us define the mesh. Another important thing to note is that every

mesh must include the point $S = E$ due to the non-smoothness in the initial condition at that point. These observations and analyses give us important information that will allow us to choose appropriate values for these parameters.

4.2 Choice of Parameters

Now that we have identified the issues associated with the choice of the parameters we need to run some numerical experiments to determine which values we should choose so that we get sufficiently accurate results with reasonable computation times. We note that EPDCOL has error control only in the time domain and since we do not have an exact solution to the problem it is hard to identify exactly how much spatial error there is. There are ways to get an estimate of the error, which we will consider later in this section, once we have experimentally determined some of the parameters.

The first parameter we will look to determine experimentally is x_R. We recognize that the interesting results are for S around the exercise price E since we already know how the value of the option behaves far away from this point. If S is much less than the exercise price the option is worthless and as S gets much larger than the exercise price the value of the option is approximately equal to S. For us this means that we are not particularly interested in the value of the option away from the exercise price. We begin by picking a large x_R that should not impact the solution around E. We keep a strict time tolerance of EPS $= 1.0 \times 10^{-12}$ and choose NINT so that there are 9 mesh points between each integer in the range [x_L,x_R]. For example between 0 and 1 we have 0.1, 0.2, $\cdots$, 0.9 as mesh points as well as the mesh points 0.0 and 1.0. We choose a strict time tolerance and a relatively large number of mesh points so that influence from these parameters is minimal in comparison with the effect of changing x_R. This will help us choose a problem domain that does not negatively impact on the accuracy of the computed solution.

Performing this experimental analysis showed us that that size of the interval does not have much effect on the solution values. Specifically, the solution around $E = 1$, where we are most interested, is not affected by the size of the interval unless x_R is very close to E. The following tables show the solutions at $x_R = 3$ (Table 2) and $x_R = 10$ (Table 3) when KORD = 4. We see here that even when reducing the right hand end point down to $x_R = 3$ there are small differences in the solution values compared with the results for $x_R = 10$ near $S = E$. We note from our numerical experiments that significant differences are not observed until $x_R = 1.6$.

S-value	Solution value	1.50000000	0.59556939
0.00000000	0.00000000	1.60000000	0.69524643
0.10000000	0.00000000	1.70000000	0.79501998
0.20000000	0.00000000	1.80000000	0.89469992
0.30000000	0.00000002	1.90000000	0.99411556
0.40000000	0.00000112	2.00000000	1.09306656
0.50000000	0.00006051	2.10000000	1.19130801
0.60000000	0.00108245	2.20000000	1.28855567
0.70000000	0.00748455	2.30000000	1.38450385
0.80000000	0.02789626	2.40000000	1.47884961
0.90000000	0.06948256	2.50000000	1.57131804
1.00000000	0.13268963	2.60000000	1.66168402
1.10000000	0.21248394	2.70000000	1.74978743
1.20000000	0.30258442	2.80000000	1.83554056
1.30000000	0.39815743	2.90000000	1.91892794
1.40000000	0.49631100	3.00000000	2.00000000

Table 2: Solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 3.0$ and EPS = 1.0×10^{-12} at time TOUT = 1.0

S-value	Solution value	1.50000000	0.59559153
0.00000000	0.00000000	1.60000000	0.69531777
0.10000000	0.00000000	1.70000000	0.79521764
0.20000000	0.00000000	1.80000000	0.89518186
0.30000000	0.00000002	1.90000000	0.99516927
0.40000000	0.00000112	2.00000000	1.09516489
0.50000000	0.00006051	2.10000000	1.19516338
0.60000000	0.00108245	2.20000000	1.29516286
0.70000000	0.00748455	2.30000000	1.39516268
0.80000000	0.02789626	2.40000000	1.49516262
0.90000000	0.06948256	2.50000000	1.59516259
1.00000000	0.13268963	2.60000000	1.69516259
1.10000000	0.21248397	2.70000000	1.79516258
1.20000000	0.30258462	2.80000000	1.89516258
1.30000000	0.39815864	2.90000000	1.99516258
1.40000000	0.49631675	3.00000000	2.09516258

Table 3: Solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 10.0$ and EPS = 1.0×10^{-12} at time TOUT = 1.0 on the interval [0,3].

In general, any difference that we see between solution values on different sized intervals differs only at values near the end of the interval. This is good since it implies that unless the right hand end point is very near E, the solution there will not be affected. If we choose $x_R = 10$ when $E = 1$, the size of the interval will have very little impact on the values of the solution we get near E. Since EPDCOL can handle other variations of this problem where E, r, and σ are changed, $x_R = 10$ allows accurate solutions for a large range of different E values. That is, if $E = 0, \cdots, 8$ we should not have to modify the value for x_R in our computations.

The next thing we want to determine experimentally is the time tolerance EPS. Since we have already determined all of the other necessary parameters we will start with the strict time tolerance of EPS = 1.0×10^{-11} and repeatedly increase it until we see unwanted changes in our solution approximations. We will keep the same number of subintervals that we used when determining x_R. Since we use 9 mesh points between each integer, in total when $x_R = 10$, that means we have 100 subintervals. We will also consider a few different KORD values at each time tolerance to see

the effect of higher order piecewise polynomials on the solution. We will consider KORD = 4, 6 and 8. As we make EPS less strict we see that the solution values begin to change. Noting the values near E we begin to see very small changes when EPS = 1.0×10^{-9}. Increasing EPS to 1.0×10^{-5} shows that the difference between the solutions values with EPS = 1.0×10^{-11} is within the 5th decimal place for KORD = 4, 6 and 8. Increasing EPS by a factor of 10 to EPS = 1.0×10^{-4}, we see that for all values of KORD the difference in the solution values is in the 4th decimal place for values around $E = 1$. Thus, the most coarse time tolerance we can choose and still have sufficiently accurate solutions is EPS = 1.0×10^{-5}. In Tables 4, 5 and 6 we see the solution values to the problem for the different cases described above. That is, solution values around $S = 1.0$ for EPS = 1.0×10^{-12}, 1.0×10^{-5} and 1.0×10^{-4} at KORD = 4, 6 and 8.

KORD = 4	EPS = 1.0×10^{-11}	EPS = 1.0×10^{-5}	EPS = 1.0×10^{-4}
S-value	Solution value	Solution value	Solution value
0.00000000	0.00000000	0.00000000	0.00000000
0.10000000	0.00000000	0.00000000	0.00000000
0.20000000	0.00000000	0.00000000	0.00000000
0.30000000	0.00000002	0.00000002	0.00000005
0.40000000	0.00000112	0.00000132	0.00000293
0.50000000	0.00006051	0.00006176	0.00009031
0.60000000	0.00108245	0.00107705	0.00117359
0.70000000	0.00748455	0.00748220	0.00748970
0.80000000	0.02789627	0.02790989	0.02776793
0.90000000	0.06948256	0.06948536	0.06944838
1.00000000	0.13268963	0.13267932	0.13275421
1.10000000	0.21248397	0.21247836	0.21251187
1.20000000	0.30258462	0.30258675	0.30254656
1.30000000	0.39815864	0.39816213	0.39810208
1.40000000	0.49631675	0.49631822	0.49627293
1.50000000	0.59559153	0.59559139	0.59556228
1.60000000	0.69531777	0.69531720	0.69529576
1.70000000	0.79521764	0.79521722	0.79519747
1.80000000	0.89518186	0.89518166	0.89516134
1.90000000	0.99516927	0.99516921	0.99514797
2.00000000	1.09516489	1.09516488	1.09514296
2.10000000	1.19516338	1.19516338	1.19514105
2.20000000	1.29516286	1.29516285	1.29514030
2.30000000	1.39516268	1.39516267	1.39514001
2.40000000	1.49516262	1.49516260	1.49513989
2.50000000	1.59516259	1.59516258	1.59513984
2.60000000	1.69516259	1.69516257	1.69513982
2.70000000	1.79516258	1.79516256	1.79513981
2.80000000	1.89516258	1.89516256	1.89513980
2.90000000	1.99516258	1.99516256	1.99513980
3.00000000	2.09516258	2.09516256	2.09513979

Table 4: Solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 10.0$ and TOUT = 1.0 for various values of EPS and KORD = 4.

KORD = 6	EPS = 1.0×10^{-11}	EPS = 1.0×10^{-5}	EPS = 1.0×10^{-4}
S-value	Solution value	Solution value	Solution value
0.00000000	0.00000000	0.00000000	0.00000000
0.10000000	0.00000000	0.00000000	0.00000000
0.20000000	0.00000000	0.00000000	0.00000000
0.30000000	0.00000000	0.00000000	0.00000002
0.40000000	0.00000059	0.00000081	0.00000274
0.50000000	0.00005774	0.00005911	0.00009132
0.60000000	0.00107673	0.00107095	0.00117689
0.70000000	0.00748090	0.00747806	0.00747675
0.80000000	0.02789921	0.02791405	0.02774079
0.90000000	0.06948980	0.06949310	0.06945117
1.00000000	0.13269677	0.13268533	0.13278376
1.10000000	0.21248772	0.21248163	0.21253025
1.20000000	0.30258472	0.30258721	0.30254640
1.30000000	0.39815674	0.39816060	0.39809501
1.40000000	0.49631448	0.49631606	0.49626702
1.50000000	0.59558972	0.59558954	0.59555911
1.60000000	0.69531659	0.69531597	0.69529445
1.70000000	0.79521697	0.79521652	0.79519706
1.80000000	0.89518150	0.89518131	0.89516129
1.90000000	0.99516909	0.99516905	0.99514803
2.00000000	1.09516481	1.09516481	1.09514305
2.10000000	1.19516334	1.19516335	1.19514113
2.20000000	1.29516284	1.29516285	1.29514038
2.30000000	1.39516267	1.39516267	1.39514008
2.40000000	1.49516261	1.49516261	1.49513996
2.50000000	1.59516259	1.59516259	1.59513991
2.60000000	1.69516259	1.69516258	1.69513988
2.70000000	1.79516258	1.79516257	1.79513987
2.80000000	1.89516258	1.89516257	1.89513987
2.90000000	1.99516258	1.99516257	1.99513986
3.00000000	2.09516258	2.09516257	2.09513986

Table 5: Solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 10.0$ and TOUT = 1.0 for various values of EPS and KORD = 6.

KORD = 8	EPS = 1.0×10^{-11}	EPS = 1.0×10^{-5}	EPS = 1.0×10^{-4}
S-value	Solution value	Solution value	Solution value
0.00000000	0.00000000	0.00000000	0.00000000
0.10000000	-0.00000000	0.00000000	0.00000000
0.20000000	0.00000000	0.00000000	0.00000000
0.30000000	0.00000000	0.00000000	0.00000003
0.40000000	0.00000059	0.00000084	0.00000310
0.50000000	0.00005774	0.00005983	0.00009297
0.60000000	0.00107673	0.00107186	0.00116731
0.70000000	0.00748090	0.00747559	0.00743034
0.80000000	0.02789921	0.02791140	0.02770385
0.90000000	0.06948980	0.06949276	0.06946252
1.00000000	0.13269677	0.13268764	0.13281571
1.10000000	0.21248772	0.21248319	0.21256528
1.20000000	0.30258472	0.30258760	0.30256973
1.30000000	0.39815674	0.39816051	0.39810556
1.40000000	0.49631448	0.49631573	0.49627337
1.50000000	0.59558972	0.59558927	0.59556627
1.60000000	0.69531659	0.69531589	0.69530319
1.70000000	0.79521697	0.79521659	0.79520682
1.80000000	0.89518150	0.89518142	0.89517150
1.90000000	0.99516909	0.99516916	0.99515842
2.00000000	1.09516481	1.09516491	1.09515348
2.10000000	1.19516334	1.19516344	1.19515157
2.20000000	1.29516284	1.29516293	1.29515082
2.30000000	1.39516267	1.39516275	1.39515051
2.40000000	1.49516261	1.49516268	1.49515038
2.50000000	1.59516259	1.59516266	1.59515033
2.60000000	1.69516259	1.69516265	1.69515030
2.70000000	1.79516258	1.79516265	1.79515029
2.80000000	1.89516258	1.89516265	1.89515028
2.90000000	1.99516258	1.99516265	1.99515028
3.00000000	2.09516258	2.09516265	2.09515027

Table 6: Solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 10.0$ and TOUT = 1.0 for various values of EPS and KORD = 8.

We note that EPDCOL outputs a value called TOTAL STEPS which indicates the total number of time steps it had to take before obtaining the solution at TOUT. The analysis for choosing EPS also shows us that KORD being 4 or 6 give sufficiently accurate solutions when EPS = 1.0×10^{-5}. When KORD = 8 we see that the solutions

have the same order of accuracy but the number of time steps (TOTAL STEPS) taken by EPDCOL is increased. The small amount of accuracy we gain does not outweigh the increase of work required by EPDCOL; hence using KORD = 4 or KORD = 6 appears to be preferable. This result is not surprising since, as we mentioned before, we do not expect a higher degree polynomial to give more accurate results for this problem due to the non-smooth initial condition at $S = E$.

The next thing we have to consider is the value for NINT, the number of subintervals. We want the number of subintervals to be as small as possible without affecting the accuracy of the solution. We have been using NINT = 100 (and a uniform mesh) so we will decrease NINT until the error in the solution is unsatisfactory. Reducing NINT to 80 shows a discrepancy in the fifth decimal place for the solution at $x = 1.0$. For KORD = 4 reducing NINT from 100 to 80 only reduced the total number of time steps from 36 to 34 and for KORD = 6 the total number of time steps was reduced from 45 to 44. The number of time steps saved does not outweigh the increase in the error in the solution and hence it is best to continue to use NINT = 100 subintervals to ensure a sufficiently accurate solution. When a uniform mesh is employed, output for KORD = 4 and KORD = 6 as we change NINT is displayed below in Tables 7 and 8.

NINT=80,KORD=4	Total Steps=34	NINT=80,KORD=6	Total Steps=44
S-value	Solution value	S-value	Solution value
0.00000000	0.00000000	0.00000000	0.00000000
0.12500000	0.00000000	0.12500000	0.00000000
0.25000000	0.00000002	0.25000000	0.00000000
0.37500000	0.00000100	0.37500000	0.00000022
0.50000000	0.00006505	0.50000000	0.00005935
0.62500000	0.00189026	0.62500000	0.00187476
0.75000000	0.01537890	0.75000000	0.01537510
0.87500000	0.05695056	0.87500000	0.05696928
1.00000000	0.13266639	1.00000000	0.13268292
1.12500000	0.23425789	1.12500000	0.23426508
1.25000000	0.34991483	1.25000000	0.34991316
1.37500000	0.47163855	1.37500000	0.47163335
1.50000000	0.59559379	1.50000000	0.59558945

Table 7: Solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 10.0$, NINT = 80 and TOUT = 1.0 for KORD = 4 and KORD = 6.

NINT=100,KORD=4	Total Steps=36	NINT=100,KORD=6	Total Steps=45
S-value	Solution value	S-value	Solution value
0.00000000	0.00000000	0.00000000	0.00000000
0.10000000	0.00000000	0.10000000	0.00000000
0.20000000	0.00000000	0.20000000	0.00000000
0.30000000	0.00000002	0.30000000	0.00000000
0.40000000	0.00000132	0.40000000	0.00000081
0.50000000	0.00006176	0.50000000	0.00005911
0.60000000	0.00107705	0.60000000	0.00107095
0.70000000	0.00748220	0.70000000	0.00747806
0.80000000	0.02790989	0.80000000	0.02791405
0.90000000	0.06948536	0.90000000	0.06949310
1.00000000	0.13267932	1.00000000	0.13268533
1.10000000	0.21247836	1.10000000	0.21248163
1.20000000	0.30258675	1.20000000	0.30258721
1.30000000	0.39816213	1.30000000	0.39816060
1.40000000	0.49631822	1.40000000	0.49631606
1.50000000	0.59559139	1.50000000	0.59558954

Table 8: Solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 10.0$, NINT = 100 and TOUT = 1.0 for KORD = 4 and KORD = 6.

4.3 Estimation of the Spatial Error for a Uniform Mesh

Now that we have determined appropriate values for the parameters, we can get an estimate of the spatial error. To perform this analysis we set EPS $= 1.0 \times 10^{-7}$. This way we know that the time error is at most EPS and is small enough that it will allow us to see the spatial error. To get an estimate of the spatial error we will proceed in the following way. We would like to compare our solution to the exact solution as given in Section 2. The issue is that this exact solution involves a improper integral that does not have a closed form solution and hence would have to be approximated numerically. Instead, we use an even larger number of mesh points to reduce the size of the subintervals since we know that the spatial error is $O(h^{k+1})$. We will compute a solution for NINT = 200 and let this be our more accurate solution. To get an estimate of the spatial error when NINT = 100 we consider the absolute value of the difference between the solution values at the mesh points obtained from EPDCOL using NINT = 100 and NINT = 200. It turns out that the maximum spatial error for KORD = 4 is 6.82×10^{-6}. This gives us an estimate of how close our computed solution is to the actual solution. A more detailed explanation of how to do this can be found in the appendix.

4.4 A Non-Uniform Mesh

As we have already mentioned we are particularly interested in the solution around $S = E$. We are also aware that the size of the subintervals has an impact on the computed solution. Since we are able to define the mesh that is passed to EPDCOL, we will construct a mesh that has points clustered around $S = E = 1$. We also need to ensure that $S = 1$ is one of the mesh points to avoid issues with the afore mentioned non-smooth initial condition. Additionally we want to be able to use the same number of subintervals (NINT = 100) and the same KORD values that we used on the uniform mesh to get some sense of how the solutions compare.

To construct a mesh that is clustered around $S = 1$ we use the inverse of the function

$$f = 1 + \frac{100}{12}\left(1 + S + \tanh\left(\frac{S-1}{4 \times 10^{-2}}\right)\right).$$

This function maps from [0,10] onto [1,101] and has a sharp vertical layer region at $S = 1$. The mesh generation function we desire must map from the mesh point indexes, 1 to 101, to the spatial domain [0,10]. The inverse of f provides this capability since the nearly vertical layer region in f corresponds to a nearly horizontal layer region in f^{-1}, leading to a clustering of mesh points near $S = 1$. Since f^{-1} is not available in closed form (to our knowledge) we approximated the f^{-1} values, for the mesh index inputs, to obtain the mesh points as outputs. The mesh points are given in Table 9.

Mesh Points	1.960000000	6.040000000
0.0	2.080000000	6.160000000
0.1200000000	2.200000000	6.280000000
0.2400000000	2.320000000	6.400000000
0.3600000000	2.440000000	6.520000000
0.4800000000	2.560000000	6.640000000
0.5999999959	2.680000000	6.760000000
0.7199983371	2.800000000	6.880000000
0.8393507169	2.920000000	7.000000000
0.9214240563	3.040000000	7.120000000
0.9471422875	3.160000000	7.240000000
0.9601382971	3.280000000	7.360000000
0.9690535379	3.400000000	7.480000000
0.9760558198	3.520000000	7.600000000
0.9819953133	3.640000000	7.720000000
0.9872976591	3.760000000	7.840000000
0.9922145724	3.880000000	7.960000000
0.9969172220	4.000000000	8.080000000
1.001539192	4.120000000	8.200000000
1.006201160	4.240000000	8.320000000
1.011030121	4.360000000	8.440000000
1.016181222	4.480000000	8.560000000
1.021872503	4.600000000	8.680000000
1.028455400	4.720000000	8.800000000
1.036590441	4.840000000	8.920000000
1.047808847	4.960000000	9.040000000
1.067185501	5.080000000	9.160000000
1.124042134	5.200000000	9.280000000
1.240012281	5.320000000	9.400000000
1.360000030	5.440000000	9.520000000
1.480000000	5.560000000	9.640000000
1.600000000	5.680000000	9.760000000
1.720000000	5.800000000	9.880000000
1.840000000	5.920000000	10.00000000

Table 9: The non-uniform mesh we have generated; note the clustering of mesh points near $S = 1$.

The only thing missing from this mesh is the point $E = S = 1$. We will take the mesh point that is closest to it and assign it the value 1.0. That means we change the mesh point $S = 1.001539192$ to $S = 1.0$. Now we have a mesh with the 101

points needed for NINT = 100 with a mesh point at E to deal with the issue of the non-smooth initial condition. We pass this mesh to EPDCOL through the driver program and EPDCOL computes a solution using these mesh points. The changes made to the driver program so that EPDCOL receives this non-uniform mesh can be found in the appendix.

4.5 Estimation of the Spatial Error for a Non-Uniform Mesh

In order to estimate the spatial error for the computed solution on the non-uniform mesh using NINT = 100 and KORD = 4, we will compute a second numerical solution on the same mesh but choose KORD = 6. We expect the solution with KORD = 6 to be more accurate since the spatial error is $O(h^{k+1})$. (Recall that KORD = k+1). Since h, the maximum size of the subintervals is less than 1, increasing k, the degree of the piecewise polynomials, will decrease the spatial error. To get an estimate of the spatial error for the solution where KORD = 4 we consider the absolute value of the difference between the solution values at the mesh points obtained from EPDCOL for KORD = 6 and KORD = 4. Again, we will set EPS = 1.0×10^{-7} so the temporal error is less prominent and we can focus on the spatial error. The largest difference is the maximum spatial error which turns out to be 1.224×10^{-5}. This gives us an idea of how close the computed solution on a non-uniform mesh with NINT = 100 and KORD = 4 is to the exact solution to the problem. A more detailed explanation of how to do this can be found in the appendix.

4.6 "Exact" Spatial Error

We have computed estimates of the spatial error for the approximate solutions computed on both a uniform and non-uniform mesh. Now we will determine a high accuracy approximation to the exact spatial error. To do this we create a high precision reference solution and compare the approximate solutions for the uniform and

non-uniform meshes to it. To get this high precision reference solution we take the uniform mesh case with NINT = 800, KORD = 10 and EPS = 1.0×10^{-8} and sampled the solution at 10001 points along the domain. This means we were not just sampling at the mesh points and NINT, KORD and EPS are such that the subintervals are small, the degree is large and the time error is sufficiently small. Then we compute a solution on a uniform mesh with NINT = 100, KORD = 4 and EPS = 1.0×10^{-7} at 10001 points along the domain. We compare this solution to the reference solution to find that the maximum error is 1.645×10^{-5}. We do the same thing sampling the solution on a non-uniform mesh where NINT = 100, KORD = 4 and EPS = 1.0×10^{-7} at 10001 points along the domain and then compare it with the reference solution. This gives a maximum error of 3.453×10^{-5} for the non-uniform mesh case.

4.7 An Analysis of the Results

We wanted to look at a non-uniform mesh where the mesh points were clustered near $S = 1.0$. We were hoping that putting more mesh points in this area of the domain would give more accurate results.

The fact that the spatial error estimates indicate that the approximate solution on the uniform mesh has a smaller spatial error than the approximate solution on the non-uniform mesh seem contradictory to what we expect. We considered a non-uniform mesh with the intention of making the solution more accurate but our error estimates indicate that the opposite is true.

We also generated a highly accurate reference solution that is very close to the exact solution. This enables us to compare the solution on the uniform mesh and the solution on the non-uniform mesh to what we consider the exact solution to the problem. Comparing these solutions to the reference solution gave the "exact" spatial error for both the uniform and non-uniform mesh solution approximations. Interestingly, again, the uniform mesh gives a smaller error than the non-uniform

mesh.

The fact that the spatial error is larger for the non-uniform mesh is contradictory to what we expect. This is an issue that can be looked into further in future work.

4.8 Solutions to Other Variations of the Problem

The code that was written will not only solve the specific problem we have been discussing where $r = 0.1$, $\sigma = 0.2$, $E = 1$, $x_R = 10$ and EPS $= 1.0 \times 10^{-5}$ at time TOUT $= 1.0$. For example, we can consider the problem for different TOUT values. From Figure 6 we know what the problem should look like at different times before expiry. We can use EPDCOL to solve the problem when TOUT $= 0.5$, 1.0 and 1.5. The following figure shows the graph of the computed solutions for all three TOUT values where we have zoomed in on the solutions on [0.0,2.0]. This plot corresponds very well to what the theoretical results indicate that we should get.

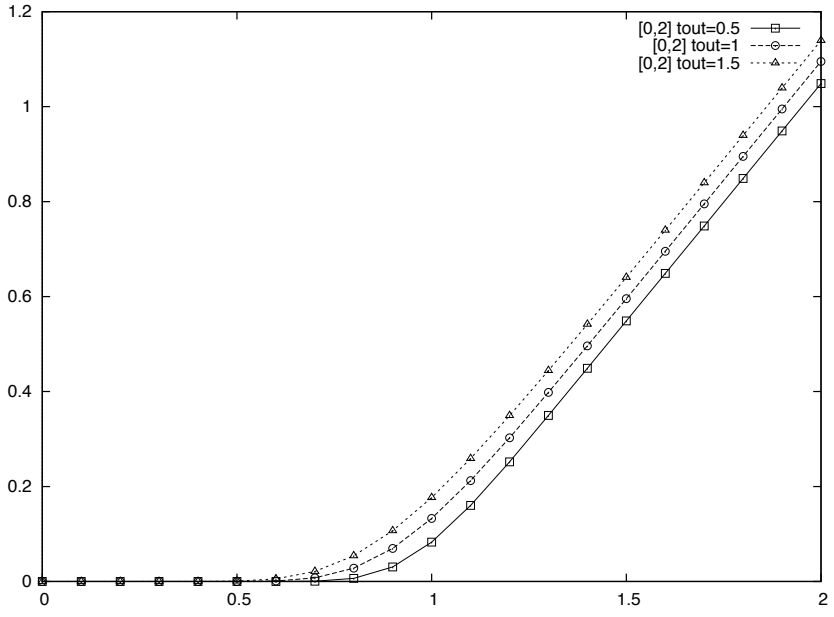

Figure 9: Graph of the solution to the Black-Scholes problem where $r = 0.1$, $\sigma = 0.2$, $E = 1$ and $x_R = 10.0$ at time TOUT $= 0.5$, 1.0 and 1.5

In addition, EPDCOL and the driver program can handle problems where parameters other than TOUT are also changed. An explanation of how compute the data that generates Figure 9 can be found in the appendix.

4.9 European Put Options

In addition to being able to solve different instances of the European call option, EPDCOL can also solve the European put option. As we have previously discussed, the boundary and initial conditions for a put option are different than for a call option. To deal with this, we have to change only the subroutines BNDRY and UINIT to correspond to the boundary and initial conditions for the put option. The section of code that was changed can be found in the appendix.

The following are graphs for the put option for the same instance of the problem we have been considering throughout the thesis. We have $r = 0.1$, $\sigma = 0.2$, $E = 1.0$ and $x_R = 10.0$, EPS $= 1.0 \times 10^{-5}$, KORD $= 4$ and NINT $= 100$ for a uniform and a non-uniform mesh. A more detailed explanation of how to do this can be found in the appendix.

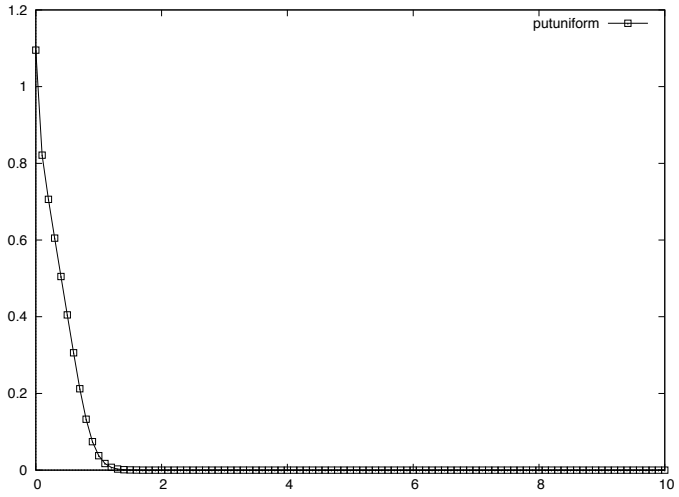

Figure 10: Graph of the solution to the Black-Scholes put option problem on a uniform mesh where $r = 0.1$, $\sigma = 0.2$, $E = 1$ and $x_R = 10.0$ at time TOUT = 1.0.

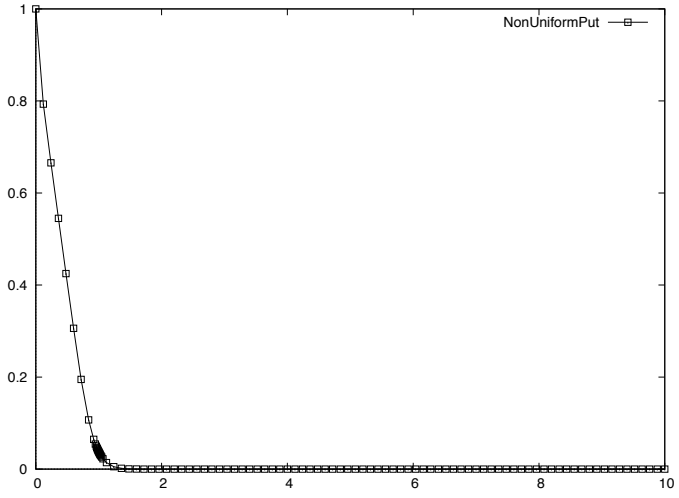

Figure 11: Graph of the solution to the Black-Scholes put option problem on a non-uniform mesh where $r = 0.1$, $\sigma = 0.2$, $E = 1$ and $x_R = 10.0$ at time TOUT = 1.0.

5 Conclusion

We have successfully used EPDCOL to solve the Black-Scholes equation which values European call and put options. Several parameter selection issues have been considered, addressed and chosen experimentally to give a solution with a good time versus accuracy trade-off. We were successfully able to deal with the issue of the non-smooth initial condition by placing a mesh point at the location of the discontinuity of the first derivative of the initial condition. We noted that there were two contributors to the error in the computed solution; the temporal and spatial error. The time tolerance was defined and passed to EPDCOL which adaptively produced a computed solution to within that indicated tolerance. We were able to compute an estimate of the spatial error and we considered both uniform and non-uniform spatial meshes. We also computed the "exact" spatial error for the two types of meshes. The results we received were contradictory to what we were expecting. We were hoping that using a non-uniform mesh would increase the accuracy of the solution but our analyses indicated that the non-uniform mesh gave solutions with a larger spatial error.

The contradictory results regarding the spatial error would be an interesting next step to pursue. More analyses would hopefully help to explain the results we are seeing. Another interesting next step would be to consider using a numerical software package such as BACOL which has both spatial and temporal error control. From here we could consider the American option version of the problem which still has the imposition of a non-smooth initial condition. Since BACOL does not let us specify the mesh ourselves we cannot use it directly on the American option problem. However, we could use EPDCOL to take the first few time steps and place a mesh point on the non-smooth point of the domain. Then we could pass the computation on to BACOL. Although the first part of the solution computed by EPDCOL would not have spatial error control we could get an estimate of the spatial error the same way we did for the European option in this thesis. Since the American option can be exercised at

any time before expiry it has a moving right boundary condition. This is an issue we would have to consider in order to use BACOL to solve this problem.

References

[1] *A Beginner's Guide To Hedging*. Retrieved from: http://www.investopedia.com /articles/basics/03/080103.asp. (Feb 19, 2010).

[2] P. Amodio, J.R. Cash, G. Roussos, R.W. Wright, G. Fairweather, I. Gladwell, G.L. Kraut and M.Paprzycki. Almost block diagonal linear systems: sequential and parallel solution techniques, and applications. *Numer. Linear Algebra Appl.*, 7:275-317, 2000.

[3] J.H. Cerutti and S.V. Parter. Collocation methods for parabolic partial differential equations in one space dimension. *Numer. Math.*, 26(3):227-254, 1976.

[4] Carl deBoor. On calculating with B-splines. *J. Approx. Theory*, 6:50-62, 1972.

[5] J. C. Diaz , G. Fairweather and P. Keast. FORTRAN packages for solving certain almost block diagonal linear systems by modified alternate row and column elimination. *ACM Trans. Math. Softw.*. 9(3): 358-375, 1983.

[6] J. Douglas, Jr. and T. Dupont. *Collocation Methods for Parabolic Equations in a Single Space Variable.* Lecture Notes in Mathematics, 385. Springer-Verlag, Berlin, 1974.

[7] Norbert Hilber, Oleg Riechmann, Christoph Schwab, Christoph Winter. *Computational Methods for Quantitative Finance*. Springer, 2013.

[8] P. Keast and P. Muir. Algorithm 688 EPDCOL: a more efficient PDECOL code. *ACM Trans. Math. Softw.*, 17:153-166, 1991.

[9] N.K. Madsen and R.F. Sicovec. ALGORITHM 540 PDECOL, general collocation software for partial differential equations. *ACM Trans. Math. Softw.*, 5:326-351, 1979.

[10] James T. McClave and Terry Sincich. *A First Course in Statistics(10th Edition)*. Pearson, 2009.

[11] R. Wang, P. Keast and P.H. Muir. BACOL: B-spline adaptive COL-location software for 1-D parabolic PDEs. *ACM Trans. Math. Software*, 30(4):454-470, 2004.

[12] Eric W. Weisstein. *Normal Distribution Function*. Retrieved from: http:// mathworld.wolfram.com/NormalDistributionFunction.html

[13] Eric W. Weisstein. *Gaussian Integral*. Retrieved from: http://mathworld. wolfram.com/GaussianIntegral.html

[14] Paul Wilmott, Sam Howison and Jeff Dewynne. *The Mathematics of Financial Derivatives: A Student Introduction*. Cambridge University Press, 1995.

A Appendix

A.1 Driver Program Code

A.1.1 Call Option Uniform Mesh

The source code for the driver program that gives solution approximations for a call option at uniformly spaced mesh points.

```
C
C     THE INPUT REQUIRED CONSISTS OF :
C                  NINT   :  THE NUMBER OF SUBINTERVALS TO BE USED IN THE
C                            VARIABLE.
```

```
C                KORD  :  THE ORDER OF THE B-SPLINES TO BE USED
C
      implicit none
      DOUBLE PRECISION XLEFT,DTUSED,XBKPT(1001),U(1,1001),SCTCH(1001),
     *                 WORK(110000),T0,TOUT,DT,EPS,DX,SIGMA,R,E

      INTEGER NQ,NSTEPS,NF,NJ,IWORK(5000),NPDE,NINT,NPTS,KORD,NCC,
     *        MF,I,K,INDEX


      COMMON /ENDPT/ XLEFT
      COMMON /GEAR0/ DTUSED,NQ,NSTEPS,NF,NJ
      COMMON /PROBPARAMS/ SIGMA,R,E
      NPDE = 1

      READ(5,9100,END=9000) NINT, KORD
      NPTS = NINT + 1
      NCC = 2
      T0 = 0.0
      TOUT = 1.0
      DT = 1.D-7
      EPS = 1.0D-5
      MF = 21
      SIGMA = 0.2
      R = 0.1
      E = 1.0

      WRITE(6,9200)NINT,KORD,EPS
      INDEX = 1
      IWORK(1) = 110000
      IWORK(2) = 5000
C     DEFINES THE UNIFORM MESH
      DO 10 I = 1,NPTS
        DX = 10.0/(DBLE(NPTS-1))
        XBKPT(I) = DBLE(I-1) * DX
   10 CONTINUE
      XLEFT = XBKPT(1)

      CALL PDECOL(T0,TOUT,DT,XBKPT,EPS,NINT,KORD,NCC,NPDE,MF,INDEX,
     *            WORK,IWORK)
      IF ( INDEX .NE. 0 ) GO TO 70
      WRITE(6,9150) TOUT,DTUSED,NSTEPS
      CALL VALUES(XBKPT,U,SCTCH,NPDE,NPTS,NPTS,0,WORK)
      DO 60 K = 1,NPDE
        WRITE(6,9300)K
```

```
        WRITE(6,9400)(XBKPT(I), U(K,I), I=1,NPTS)
   60 CONTINUE
   70 WRITE(6,9500) INDEX
 9000 STOP
 9100 FORMAT(2I4)
 9150 FORMAT(' T = ',E10.3,' DT = ',E10.3,' TOTAL STEPS = ',I6)
 9200 FORMAT('NO. OF SUBINTERVALS = ',I3,'  KORD = ',I2,
     *' EPS = ',D10.2)
 9300 FORMAT(10X,'PDE COMPONENT = ',I3)
 9400 FORMAT(10X,2F15.8)
 9500 FORMAT('  INDEX = ',I3)
      END
      SUBROUTINE F(T,X,U,UX,UXX,FVAL,NPDE)
C
C     THIS IS THE USER SUPPLIED SUBROUTINE TO SPECIFY THE DIFFERENTIAL
C     EQUATIONS.
C
      DOUBLE PRECISION U(NPDE),UX(NPDE),UXX(NPDE),FVAL(NPDE),T,X,SIGMA,R,E
      INTEGER NPDE

      COMMON /PROBPARAMS/ SIGMA,R,E

      FVAL(1) = 0.5*SIGMA**2*X**2*UXX(1) + R*X*UX(1)
     *          - R*U(1)

      RETURN
      END
      SUBROUTINE BNDRY(T,X,U,UX,DBDU,DBDUX,DZDT,NPDE)
C
C     THIS ROUTINE SPECIFIES THE DERIVATIVES OF THE BOUNDARY CONDITION EQUATIONS.
C
      INTEGER NPDE
      DOUBLE PRECISION T,X,U(NPDE),UX(NPDE),DZDT(NPDE),
     *                 DBDU(NPDE,NPDE), DBDUX(NPDE,NPDE),
     *                 XLEFT
      COMMON /ENDPT/ XLEFT
      IF ( X .NE. XLEFT ) GO TO 10
        DBDU(1,1) = 1.0
        DBDUX(1,1) = 0.0
        DZDT(1) = 0.0
      RETURN
   10 CONTINUE
        DBDU(1,1) = 1.0
        DBDUX(1,1) = 0.0
        DZDT(1) = 0.0
```

```
      RETURN
      END
      SUBROUTINE UINIT(X,U,NPDE)
C
C     UINIT GIVES THE INITIAL CONDITIONS AT T=T0.
C
      INTEGER NPDE
      DOUBLE PRECISION X,U(NPDE), SIGMA, R, E

      COMMON /PROBPARAMS/ SIGMA,R,E

      U(1) = max(X - E, 0.0)
      RETURN
      END
      SUBROUTINE DERIVF(T,X,U,UX,UXX,DFDU,DFDUX,DFDUXX,NPDE)
C
C     THIS IS THE OPTIONAL ROUTINE PROVIDED IF THE USER WISHES TO
C     SUPPLY AN ANALYTIC JACOBIAN.
C
      INTEGER NPDE
      DOUBLE PRECISION T,X,U(NPDE),UX(NPDE),UXX(NPDE),
     *       DFDU(NPDE,NPDE),DFDUX(NPDE,NPDE),DFDUXX(NPDE,NPDE),
     *       SIGMA,R,E
      COMMON /PROBPARAMS/ SIGMA,R,E

      DFDU(1,1) = -R
      DFDUX(1,1) = R*X
      DFDUXX(1,1) = (0.5)*SIGMA**2*X**2
      RETURN
      END
```

A.1.2 Put Option Uniform Mesh

The subroutines from the driver program that have been modified to solve the put option on a uniform mesh.

```
      SUBROUTINE BNDRY(T,X,U,UX,DBDU,DBDUX,DZDT,NPDE)
C
C     THIS ROUTINE SPECIFIES THE DERIVATIVES OF THE BOUNDARY CONDITION EQUATIONS
C
      INTEGER NPDE
      DOUBLE PRECISION T,X,U(NPDE),UX(NPDE),DZDT(NPDE),
```

```
     *                  DBDU(NPDE,NPDE), DBDUX(NPDE,NPDE),
     *                  XLEFT,TOUT, SIGMA, R, E
      COMMON /ENDPT/ XLEFT
      COMMON /EXPIRTYTIME/ TOUT
      COMMON /PROBPARAMS/ SIGMA,R,E

      IF ( X .NE. XLEFT ) GO TO 10
        DBDU(1,1) = 1.0
        DBDUX(1,1) = 0.0
        DZDT(1) = E*R*exp(R*T - R*TOUT)
      RETURN
   10 CONTINUE
        DBDU(1,1) = 1.0
        DBDUX(1,1) = 0.0
        DZDT(1) = 0.0
      RETURN
      END
      SUBROUTINE UINIT(X,U,NPDE)
C
C     UINIT GIVES THE INITIAL CONDITIONS AT T=T0.
C
      INTEGER NPDE
      DOUBLE PRECISION X,U(NPDE)
      REAL SIGMA, R, E
      COMMON /PROBPARAMS/ SIGMA,R,E

      U(1) = max(E - X, 0.0)
      RETURN
```

A.1.3 Call and Put Option Non-Uniform Mesh

The assignment statements for the nonuniform mesh used for both the call and put option problems

```
      DO I = 1,6
        XBKPT(I) = (I-1)*0.12
      END DO
        XBKPT(7) = .7199983371
        XBKPT(8) = .8393507169
        XBKPT(9) = .9214240563
        XBKPT(10) = .9471422875
        XBKPT(11) = .9601382971
```

```
      XBKPT(12) = .9690535379
      XBKPT(13) = .9760558198
      XBKPT(14) = .9819953133
      XBKPT(15) = .9872976591
      XBKPT(16) = .9922145724
      XBKPT(17) = .9969172220
      XBKPT(18) = 1.0
      XBKPT(19) = 1.006201160
      XBKPT(20) = 1.011030121
      XBKPT(21) = 1.016181222
      XBKPT(22) = 1.021872503
      XBKPT(23) = 1.028455400
      XBKPT(24) = 1.036590441
      XBKPT(25) = 1.047808847
      XBKPT(26) = 1.067185501
      XBKPT(27) = 1.124042134
      XBKPT(28) = 1.240012281
    DO 10 I = 29,101
      XBKPT(I) = (I-1)*0.12 - 2
10  CONTINUE
```

A.1.4 Code for Determining the Exact Error

First declare the array XOUT(10001) as double precision in the main program and then make the following change in the call to the subroutine VALUES

```
    DO K = 1,10001
      XOUT(K) = (K-1)*0.001
    END DO
    CALL VALUES(XOUT,U,SCTCH,1,10001,10001,0,WORK)
    DO 60 K = 1,10001
      WRITE(7,9400) XOUT(K), U(1,K)
 60 CONTINUE
```

A.2 Determining the Relevant Data for Chapter 4

A.2.1 Section 4 - Introduction

The data for Figures 7 and 8 in Section 4 showing a preliminary test can be obtained by running the driver program for a uniform mesh with $x_R = 20$, NINT = 40, EPS

$= 1.0 \times 10^{-6}$, $r = 0.1$, $\sigma = 0.2$ and $E = 1.0$.

A.2.2 Section 4.3

To determine the spatial error estimation for an approximate solution on a uniform mesh from Section 4.3 we run the uniform mesh driver program with EPS $= 1.0 \times 10^{-7}$, $r = 0.1$, $\sigma = 0.2$, $E = 1.0$, KORD $= 4$ and NINT $= 100$ and 200. Then we take the absolute value of the difference between corresponding solution values for NINT $=$ 100 and NINT $= 200$. The largest of these errors is the maximum spatial error and gives us a sense of how close our computed solution is to the exact solution.

A.2.3 Section 4.5

To determine the spatial error estimation in solutions on a non-uniform mesh from Section 4.5 we run the non-uniform mesh version of the driver for a call option with EPS $= 1.0 \times 10^{-7}$, $r = 0.1$, $\sigma = 0.2$, $E = 1.0$, NINT $= 100$ and KORD $= 4$ and 6. Then we take the absolute value of the difference between corresponding solution values for KORD $= 6$ and KORD $= 4$. The largest of these errors is the maximum spatial error giving us an estimate of how close our computed solution is to the exact solution.

A.2.4 Section 4.7

To generate the data used to compute the exact spatial error we began by computing the reference solution. To do this we ran the code with the changes indicated in A.1.4 made to the uniform mesh driver program with $r = 0.1$, $\sigma = 0.2$, $E = 1.0$, EPS $= 1.0 \times 10^{-8}$, KORD $= 10$, NINT $= 800$ and TOUT $= 1.0$. To generate the solution on the uniform mesh we repeated the last step but instead set EPS $= 1.0 \times 10^{-7}$, KORD $= 4$ and NINT $= 100$. To generate the solution on the non-uniform mesh we made the changes indicated in A.1.4 to the non-uniform mesh driver program with $r = 0.1$,

$\sigma = 0.2$, $E = 1.0$, EPS $= 1.0 \times 10^{-7}$, KORD $= 4$, NINT $= 100$ and TOUT $= 1.0$. We now have the reference solution and the approximate solutions on both the uniform and non-uniform meshes can now be compared.

A.2.5 Section 4.8

The data for Figure 9 in Section 4.7 showing three different solution approximations at different expiry times can be found by running the driver program for a call option with a uniform mesh where $r = 0.1$, $\sigma = 0.2$, $E = 1.0$, EPS $= 1.0 \times 10^{-5}$, KORD $=$ 4, NINT $= 100$ and TOUT $= 0.5$, 1.0 and 1.5.

A.2.6 Section 4.9

The generate the data for the put option graphs in Figure 10 and Figure 11 showing the approximate solutions on both a uniform and a non-uniform mesh, we ran the driver code with $r = 0.1$, $\sigma = 0.2$, $E = 1.0$, EPS $= 1.0 \times 10^{-5}$, KORD $= 4$, NINT $=$ 100 and TOUT $= 1.0$.