Numerical Methods For American Options

S.C.Benbow

August 22, 2005

Abstract

The problem of solving the Black-Scholes equation for the valuation of American options is tackled using a Crank-Nicolson finite difference method formulated in a Lagrangian frame.

We firstly introduce a transformation to convert the Black-Scholes equation into a dimensionless constant coefficient forward equation. We then formulate the equation is the moving Lagrangian reference frame. The problem of solving the Black-Scholes equation for American options is treated as a free boundary problem, where we must determine both the value of the option, and also when the option should be exercised. We introduce a new method for locating the moving boundary. The equation is then solved using a finite difference Crank-Nicolson method.

A monitor function is introduced to increase the resolution of the method close to the exercise boundary, and the method is modified to accommodate this. We then attempt to solve the problem using a finite element method and compare the accuracy of the two approaches.

Acknowledgements

I would like to thank Prof. M.J.Baines for his time and invaluable help with this thesis. I have thoroughly enjoyed working with him, this dissertation has given me an insight into the true nature of research and provided me with motivation to continue with my studies. I would also like to acknowledge my colleagues on the Msc course and the other members of the academic staff in the Department of mathematics at the University of Reading.

This year has been both enjoyable and challenging. I would like to thank my family, especially me wife, for all of their support and hard work in the last twelve months in allowing me to study at Reading and take full advantage of this wonderful opportunity.

Contents

Chapter 1

Introduction

1.1 The Problem

The simplest financial option, a *European call option*, is a contract with the following conditions

- At a prescribed time in the future, known as the *expiry date* the holder of the option may do the following;
- purchase a prescribed asset, known as the *underlying* for a
- prescribed amount, known as the *exercise price* or *strike price*

For the holder of the option, the contract is a *right* but not an *obligation*. The other party to the contract, the individual who is known as the *writer* does have a potential obligation, he must sell the asset if the holder chooses to buy it.

The option confers to its holder a right without an obligation, and therefore has intrinsic value.

The main concerns in the valuation of options are:

- How much would one pay for this right, i.e. what is the value of an option?
- How can the writer minimize the risk associated with his obligation?

Options have become extremely popular recently, primarily because they are attractive to investors, both for speculation and for hedging, and because there is now a systematic way to determine how much they are worth.

We let E denote the exercise price, i.e the cost of purchasing the option, and $S(T)$ denote the underlying asset price at the expiry date. At expiry if $S(T) > E$ then the holder of the call option may buy that asset for E and then immediately sell it in the market for $S(T)$, gaining an amount $S(T) - E$. Conversely, if $E \geq S(T)$ then the holder gains nothing. The value of the call option may therefore be expressed as

$$C = max(S(T) - E, 0) \tag{1.1.1}$$

Plotting $S(T)$ on the x-axis and C on the y-axis gives a *payoff* diagram as shown in 1.1.

American options have the additional feature that exercise is permitted at any time during the life of the option. This is in contrast to a European option which may only be exercised *at* expiry. Since the American option gives its holder greater rights than the European option, via the right of early

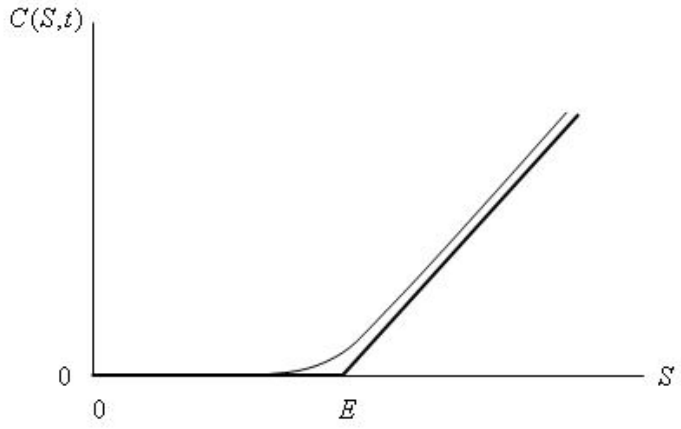

Figure 1.1: Payoff Diagram for a European Call

exercise, potentially it has a higher value. This report will focus on the valuation of American options, which are mathematically more interesting than their European counterparts since they can be interpreted as free boundary problems.

The American option valuation problem can be shown to be uniquely specified by a series of constraints, which are similar to those of an 'obstacle' problem. Since we do not know the location of the free boundary $S_f(t)$ a *priori* we are lacking one piece of information compared with the European options. Not only must a *value* be assigned to the option but, we must also determine *when it is best to exercise the option.*

1.2 Aims

The aims of this dissertation are to produce an accurate method for the valuation of American call options. The optimal exercise boundary is modeled

by a moving boundary. The time dependent boundary point is physically interpreted as the division between two regions, one where we should hold the option, and the other where we should exercise. This point is known as the **optimal exercise price**. We seek a method which will determine accurately the location of this free boundary, and furthermore provide a corresponding valuation for the option at discrete time steps up until the expiry date.

The conventional approach is to transform the Black-Scholes equation into a dimensionless parabolic equation and then discretise the problem using numerical methods to find a solution[1]. We propose a new method of solution, in which the equations are discretised on a moving grid. Since the optimal exercise boundary is time dependent, and found to increase in time for the case of a call, we also propose a new technique for expanding the domain.

Finally, we intend to solve same the problem using a finite element approach, and compare the accuracy between the two schemes.

[1]see K.N PANAZOPOULOS ET AL

Chapter 2

Background

2.1 Model for Asset Prices

In order to value an option we must develop a mathematical description of how the underlying asset behaves. The price of an asset is a measure of investors confidence. Although an oversimplification, it is reasonable to assume that the market responds instantaneously to external influences. Furthermore asset prices must move randomly because of the **efficient market hypothesis**. This can be stated as

- The past history is fully reflected in the present price, which does not hold any further information
- Markets respond immediately to any new information about an asset

With these two assumptions, the asset price is said to follow a **Markov process**

Definition 1. *A Markov process is a particular type of stochastic process where only the present value of a variable is relevant in predicting the future. The past history of the variable, and the way in which the present has emerged from the past are irrelevant*

Suppose that at time t the asset price is S. Consider a small subsequent time interval dt, during which S changes to $S + dS$. We decompose this return into two parts, one component that is deterministic, comparable to the return on a risk-free investment. This gives contribution to the return dS/S

$$\mu dt \tag{2.1.1}$$

μ is a measure of the average rate of growth of the asset price, also known as drift.

The second contribution to dS/S models the random change in the asset price in response to external effects. It is represented by a random sample drawn from a normal distribution with mean zero. It adds contribution

$$\sigma dX \tag{2.1.2}$$

σ is known as the *volitility*, which measures the standard deviation of the returns.

Putting the two together yields the **Stochastic Differential Equation**

$$\frac{dS}{S} = \sigma dX + \mu dt \tag{2.1.3}$$

This is the basic mathematical representation for generating asset prices. The term dX, which contains the random element which is a feature of asset

prices, is known as a **Wiener process**, it has the following properties:

- dX is a random variable, drawn from a normal distribution
- dX has a mean of zero and a variance dt

This may be expressed as

$$dX = \phi\sqrt{dt} \tag{2.1.4}$$

ϕ is a random variable drawn from a standardized normal distribution, with zero mean and variance of one. The probability distribution function is given by

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\phi^2} \tag{2.1.5}$$

A generalized Wiener process with drift μ and variance σ is shown in figure 2.1

2.2 Black-Scholes Model

The key question concerning the valuation of Options is: what is an option worth at time t=0? The problem is to systematically determine a *fair* value for an option, at the time at which the contract is entered into. Before we proceed with any analysis, there are several assumptions that we must first make

- The asset price follows a log normal random walk
- The risk-free interest rate r and the asset volatility σ are known functions of time over the life of the option.

- There are no associated transaction costs
- The underlying asset pays no dividends during the life of the option.
- There are no arbitrage possibilities
- Trading of the underlying asset can take place continuously

We now look for a function $V(S,t)$ that gives the option value for any asset price $S \geq 0$ and at any time $0 \leq t \leq T$. In this setting, $V(S_0, 0)$ is the required time-zero option value. We further assume that such a function exists and is smooth in both variables. Ito's Lemma provides us with a derivative chain rule for stochastic functions; i.e. if $f = f(W,t)$ where W is some stochastic function.

$$df = \frac{df}{dS}(\sigma S dX + \mu S dt) + \frac{1}{2}\sigma^2 S^2 \frac{d^2 f}{dS^2} dt \tag{2.2.1}$$

Using 2.2.1 we can write

$$dV = \sigma S \frac{dV}{dS} dX + \left(\mu S \frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + \frac{\partial V}{\partial t} \right) dt \tag{2.2.2}$$

This gives the random walk followed by V.

We now construct a portfolio consisting of one option and a proportion $-\Delta$ of the underlying asset. The value of the portfolio is

$$\Pi = V - \Delta S \tag{2.2.3}$$

The change in the value of this portfolio in one time-step is

$$d\Pi = dV - \Delta dS \tag{2.2.4}$$

Combining 2.1.3, 2.2.2 and 2.2.3 we find that Π follows the random walk

$$d\Pi = \sigma S\left(\frac{\partial V}{\partial S} - \Delta\right) dX + \left(\mu S\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2}{\partial S^2} + \frac{\partial V}{\partial t} - \mu\Delta S\right) dt \quad (2.2.5)$$

We can eliminate the random component by choosing $\Delta = \frac{\partial V}{\partial S}$. This results in a portfolio whose increment is wholly deterministic

$$d\Pi = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}\right) dt \quad (2.2.6)$$

The return on on an amount Π invested in a risk less asset would see a growth of $r\Pi dt$ in a time dt. If the right hand side of 2.2.6 were greater than this amount, an arbitrager could make a guaranteed risk less profit by borrowing an amount Π to invest in the portfolio. Conversely, if the right-hand side of 2.2.6 were less than $r\Pi dt$ then the arbitrager would make a risk less, no cost, instantaneous profit.

Thus we have

$$r\Pi dt = \left(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2}\right) dt \quad (2.2.7)$$

Substituting 2.2.3 and $\Delta = \frac{\partial V}{\partial S}$ into 2.2.7 and dividing by dt we arrive at the *Black-Scholes partial differential equation*

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \quad (2.2.8)$$

Any derivative security whose price depends *only* on the current value of S and on t, which is paid for up-front, must satisfy the Black-Scholes equation.

2.3 Black-Scholes For European Option

The Black-Scholes equation 2.2.8 is a backward parabolic equation. We must therefore impose boundary conditions to ensure a unique solution. We must impose two conditions on S, and one on t. For example we may specify that $V(S,t) = V_a(t)$ on $S = a$ and also that $V(S,t) = V_b(t)$ on $S = b$.

Since the equation is backward in time we must also impose a final condition such as

$$V(S,t) \quad = \quad V_T(S) \qquad on \qquad t = T \tag{2.3.1}$$

Where V_T is a known function of time.

In the case of European options, in particular the European call, we denote the call value by $C(S,t)$, with exercise price E and expiry date T.

At time $t = T$, the value of the call is known for certainty to be

$$C(S,T) = max(S - E, 0) \tag{2.3.2}$$

This is the final condition.
If $S = 0$ at expiry then the payoff is zero, the call option is therefore worthless, even if there is still a period of time until expiry.

$$C(0,t) = 0 \tag{2.3.3}$$

As the asset price increases, it will become more likely that the option will be exercised and the actual magnitude of the exercise price becomes less

important. We can therefore write that as $S \to \infty$ the value of the option becomes that of the asset

$$C(S,t)\ S \quad as \quad S \to \infty \tag{2.3.4}$$

For the European options, without the possibility of early exercise 2.2.8 can be solved exactly to give the Black-Scholes value of the call option.

Assuming that the interest rate and the volatility are constant, the explicit solution for the European call is

$$C(S,t) = SN(d_1) - Ee^{-r(T-t)}N(d_2) \tag{2.3.5}$$

N is a cumulative distribution function for a standardized normal random variable, given by

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy \tag{2.3.6}$$

$$d_1 = \frac{log(S/E) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{2.3.7}$$

$$d_2 = \frac{log(S/E) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{2.3.8}$$

Figure 2.3 below shows a European call value $C(S,t)$ as a function of S for several values of time to expiry, with $r = 0.1$ and $\sigma = 0.2$

2.4 Modification to the model

In this dissertation we attempt to model the price of American options based on dividend paying assets. The model introduced in the previous section makes the simplification that no dividends are paid. We now consider the

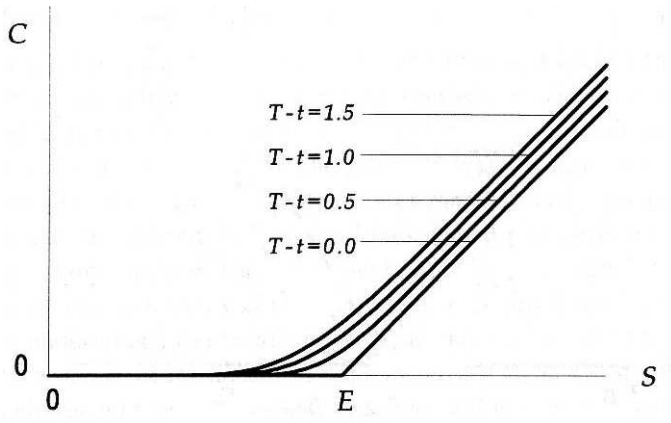

Figure 2.1: American Call Value Prior to expiry

effect on the options price when dividend payment is incorporated into the model.

When assets pay out dividends, the price of an option on a underlying asset is affected by the payments. A modification must be made to the black-scholes equation.

In modeling dividends we must ask two questions:

- When and how often are dividend payments made?
- How large are the dividend payments?

The amounts paid as dividends may be modeled as either deterministic or stochastic. In this dissertation we consider only those equities with dividends whose amount and timing is known at the start of the options life.

Suppose that in time dt the underlying asset pays out a dividend D_0Sdt where D_0 is a constant. The payment is independent of time, but dependent

on the stock price S. The *Dividend Yield* is defined as the proportion of the asset price that is paid out per unit time in this way.

Arbitrage considerations show that in each time-step dt, the asset price must fall by the amount of the dividend payment, in addition to the usual fluctuations. The random walk of the asset price 2.1.3 is modified to become

$$dS = \sigma S dX + (\mu - D_0) S dt \tag{2.4.1}$$

Considering the effect of the dividend payments on our hedged portfolio, we receive and amount $D_0 S dt$ for every asset held, and since we hold $-\Delta$ of the underlying, the portfolio changes by an amount

$$-D_0 S \Delta dt \tag{2.4.2}$$

Adding 2.4.2 to 2.2.4 we arrive at

$$d\Pi = dV - \Delta dS - D_0 S \Delta dt \tag{2.4.3}$$

Following the same analysis as previously we obtain

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + (r - D_0) S \frac{\partial V}{\partial S} - rV = 0 \tag{2.4.4}$$

The only change to the boundary conditions is that

$$C(S,t) \; Se^{-D_0(T-t)} \quad as \quad S \to \infty \tag{2.4.5}$$

The value of a European call with dividends can be shown to be

$$C(S,t) = e^{-D_0(T-t)} S N(d_{10}) - E e^{-r(T-t)} N(d_{20}) \tag{2.4.6}$$

Where

$$d_{10} = \frac{log(S/E) + (r - D_0 + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \tag{2.4.7}$$

$$d_{20} = d_1 - \sigma\sqrt{T-t} \tag{2.4.8}$$

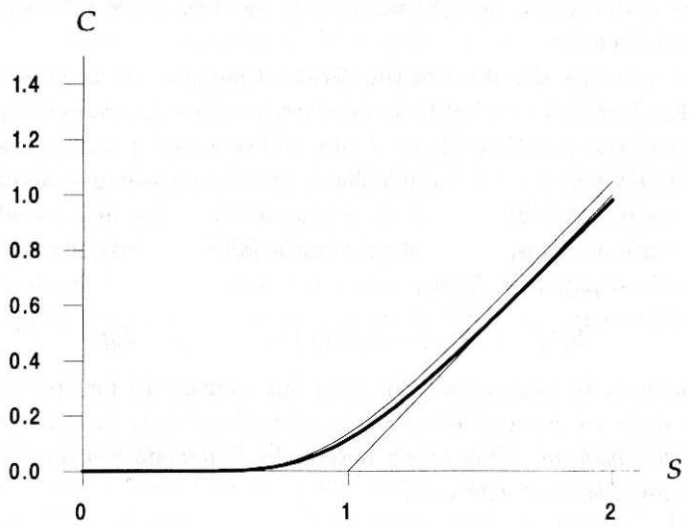

Figure 2.2: European Option Values with (lower) and without (lower) dividends

2.5 American Option

American Options have the important additional feature that early exercise is permitted at any time during the life of the option.

Definition 2. *An American Call Option gives its holder the right, but not the obligation, to purchase from the writer a prescribed asset for a prescribed price at any time between the start date and a prescribed expiry date in the future.*

The formulae in section 2.3 and 2.4 do not necessarily agree with the value of American options. The ability to exercise the option at any time extends to the owner additional rights, and thus the American option has potentially a higher value.

If S lies in this range so that $P(S,t) < max(E-S,0)$ and we exercise the option, there is an obvious arbitrage opportunity. We could buy the asset in the market immediately for S and at the same time buy the option for P; if we then exercised the option by selling the asset for E we make a risk free profit of $E-P-S$.

This opportunity would not last long before the value of the option was pushed up by the demand of the arbitragers. We must therefore conclude that when early exercise is permitted we must impose the constraint

$$V(S,t) \geq max(S-E,0) \tag{2.5.1}$$

American and European options **must** therefore have different values.

In the case of American options there are some values of S for which it is optimal from the holders point of view to exercise the American option. If this were not the case the option would have the same value as the European option, the Black-Scholes equation would hold for all S.

The valuation of an American option is therefore more complicated than its European counterpart since we have to determine not only the option value but also, for each value of S, whether or not it should be exercised.

This is what is known as a *free boundary* problem. At each time t there is a particular value of S which marks the boundary between two regions: to one side one should hold the option and to the other side one should exercise it.

We denote this value, which varies with time, by $S_f(t)$, and refer to it as the *optimal exercise price*.

As we have already observed, since we do not know S_f a *priori* unlike the corresponding European problem, we do not know where to apply the boundary conditions, and for this reason, the problem is called a *free boundary problem*.

An American option valuation can be shown to be uniquely specified by a set of constraints

- the option value must be greater than or equal to the payoff function
- the Black-Scholes equation is replaced by an inequality

- the option value must be a continuous function of S
- the option delta (slope) must be continuous

2.6 American Call with Dividends

From section 2.2 the value $C(S,t)$ of an American option satisfies

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + (r - D_0)S\frac{\partial C}{\partial S} - rC = 0 \tag{2.6.1}$$

This holds as long as exercise is not optimal. The payoff condition is

$$C(S,T) = max(S - E, 0) \tag{2.6.2}$$

Also, since the option may be exercised at any time, we have that

$$C(S,t) \geq max(S - E, 0) \tag{2.6.3}$$

Along the optimal exercise boundary $S = S_f(t)$

$$C(S_f(t),t) = S_f(t) - E\frac{\partial C}{\partial S}(S_f(t),t) = 1 \tag{2.6.4}$$

If the optimal exercise boundary exists then 2.6.1 is valid only while $C(S,t) > max(S-E,0)$ since $max(S-E,0)$ is not a solution of the Black-Scholes equation.

2.6.1 can be replaced by an inequality

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + (r - D_0)S\frac{\partial C}{\partial S} - rC \leq 0 \tag{2.6.5}$$

The inequality holds only if $C(S,t) > max(S - E, 0)$. If early exercise if optimal, it is because the option would be less valuable than if it were exercised immediately and the funds deposited in an interest paying bank account.

2.7 General Analysis of Call with Dividends

We can simplify the Black-Scholes equation with dividend payments by assuming that the interest rate and the dividend payments satisfy $r > D_0 > 0$. We can then make equations 2.6.1,2.6.2 and 2.6.4 dimensionless and reduce 2.6.1 to a constant coefficient forward equation[1].

We now also subtract off the payoff $S - E$ for the call value $C(S,t)$.

$$S = Ee^x, t = T - \frac{\tau}{\frac{1}{2}\sigma^2}, C(S,t) = S - E + Ec(x,\tau) \tag{2.7.1}$$

the result is

$$\frac{\partial c}{\partial \tau} = \frac{\partial^2 c}{\partial x^2} + (k' - 1)\frac{\partial c}{\partial x} - kc + f(x) \tag{2.7.2}$$

for $-\infty < x, \infty$ and $\tau > 0$. The function $c(x,0)$,the initial profile, is given by

$$c(x,0) = max(1 - e^x, 0) \tag{2.7.3}$$

A graph of $c(x,0)$ is shown in 2.7. The two parameters k and $\acute{k}$ are given by

$$k = \frac{r}{\frac{1}{2}\sigma^2}, \ \acute{k} = \frac{(r - D_0)}{\frac{1}{2}\sigma^2} \tag{2.7.4}$$

The function f is given by

$$f(x) = (\acute{k} - k)e^x + k \tag{2.7.5}$$

Assuming that the free boundary does exist, $x = x_f(t)$, at this boundary we have

$$c(x_f(\tau),\tau) = \frac{\partial c}{\partial x}(x_f(\tau),\tau) = 0 \tag{2.7.6}$$

[1]We follow the same analysis as Wilmott,Mathematics of Financial Derivatives,Chapter 7.7

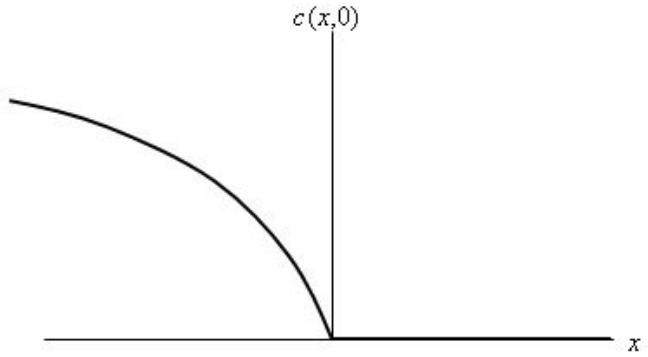

Figure 2.3: *c(x,0)*

We now have the constraint that $c \geq max(1 - e^x, 0)$. The behavior of $f(x)$, the consumption/replenishment term, is critical to the behavior of the free boundary. A graph of $f(x)$ is shown in figure 2.7. $f(x)$ is positive when $x < x_0$ where

$$x_0 = log(k/(k - \acute{k})) = log(r/D_0) > 0 \tag{2.7.7}$$

For $x \geq x_0$ the function is negative. If we suppose that no free boundary existed and consider the initial data $c(x, 0)$ for positive values of x. For $x > 0$

$$c(x, 0) = \frac{\partial c(x, 0)}{\partial x} = \frac{\partial^2 c(x, 0)}{\partial x^2} = 0 \tag{2.7.8}$$

From equation 2.7.2 at expiry we have

$$\frac{\partial c}{\partial \tau} = f(x) \tag{2.7.9}$$

For $0 < x < x_0$, $f(x) > 0$ and thus c is positive. If $x > x_0$ then $f(x) < 0$ and c will be negative. We have the constraint that $c > 0$ for $x > 0$ thus the latter does not satisfy this constraint.

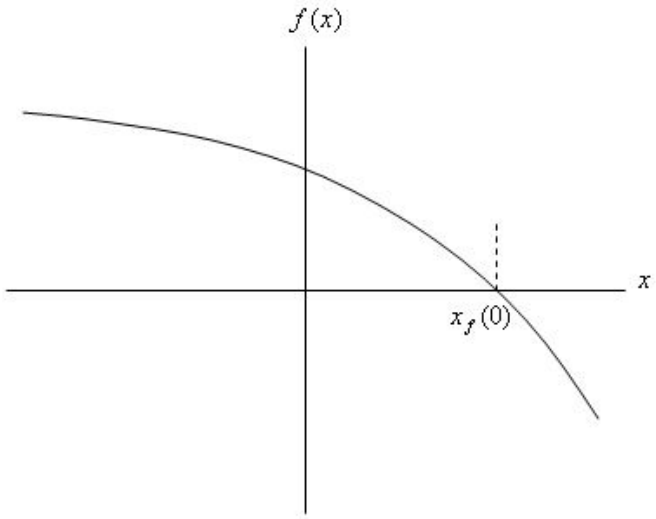

Figure 2.4: *f(x)*

If we hold the option in $x > x_0$ the option falls below its intrinsic value and the constraint is broken.

We must therefore take $x_f(0) = x_0$ since this is the only point consistent with $c(x_f(0), 0) = 0$

Figures 2.7 and 2.7 show the values of $c(x, \tau)$ in the dimensionless diffusion setting, and the original $C(S, t)$ at times prior to expiry.

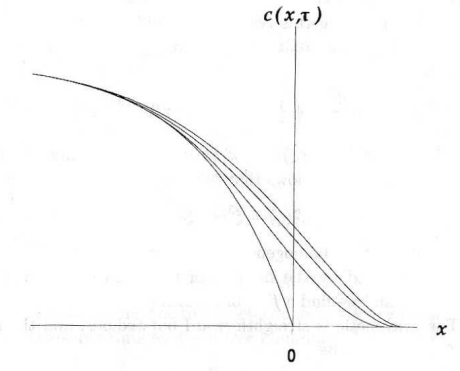

Figure 2.5: Local Solution $c(x, \tau)$

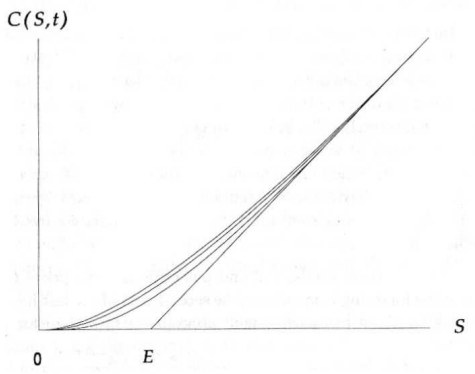

Figure 2.6: Option Value $C(S, t)$

Chapter 3

Transformation

3.1 American Options PDE

In this section we introduce a transformation for the valuation of American calls. The Black-Scholes equation modeling the price of a dividend paying asset V, under deterministic yield D, volatility σ and interest rate r may be written as

$$V_t + \frac{1}{2}\sigma^2 S^2 V_{ss} + (r - D)SV_s - rV = 0 \tag{3.1.1}$$

Here S denotes the underlying asset on which the call option is written. The *early exercise* feature of the American option results in an optimal exercise boundary problem, which in the PDE setting is treated as a free boundary problem.

We denoted the free boundary with $\tilde{B}(t)$. The domain of equation 3.1.1 is

$(0, \tilde{B(t)}) \times [0, T)$. The boundary conditions are given below

$$\begin{aligned} V(S,T) &= max(S-K,0),\ S \in (0, \tilde{B}(T)) && (3.1.2)\\ \tilde{B}(T) &= max(K, \frac{rK}{D}), && (3.1.3)\\ V(0,t) &= 0 && (3.1.4)\\ V(\tilde{B}(t),t) &= \tilde{B}(t) - K && (3.1.5)\\ V_s(\tilde{B}(t),t) &= 1 && (3.1.6) \end{aligned}$$

3.2 Transformation to Diffusion Equation

The transformation we apply, a detailed derivation can be found in Panzaopoulos,Houstis and Kortesis(1997), reduces the Black-Scholes equation to a diffusion equation. The benefit of doing this is that the diffusion equation is a far simpler and less cluttered equation than the Black-Scholes. It is then a simpler matter to find exact solutions to the diffusion equation and then convert back to financial variables. Letting k_1 and k_2 be defined as follows

$$k_1 = \frac{2r}{\sigma^2} \qquad k_2 = \frac{2(r-D)}{\sigma^2} \tag{3.2.1}$$

We introduce the following transformations

$$\begin{aligned} \tau &= \frac{1}{2}\sigma^2(T-t) && (3.2.2)\\ x &= log(S/K) + (K_2 - 1)\tau && (3.2.3)\\ B(\tau) &= log(\tilde{B(\tau)}/K) + (k_2 - 1)\tau && (3.2.4)\\ u(x,\tau) &= \frac{e^{k_1\tau}}{K}(V(S,t) - S + K) && (3.2.5) \end{aligned}$$

Equation 2.6.5 becomes

$$u_\tau = u_{xx} + g(x,\tau) \tag{3.2.6}$$

Where

$$g(x,\tau) = e^{k_1\tau}((k_2 - k_1)e^{x-(k_2-1)\tau} + k_1) \tag{3.2.7}$$

This is the principle equation which we will attempt to solve using numerical methods in the following section. The domain of equation 3.2.6 is $(-\infty, B(0)) \times (0, \frac{1}{2}\sigma^2 T)$. The boundary conditions for the American call become

$$\begin{aligned} u(x,0) &= max(1-e^x, 0), x \in (-\infty, B(0)) && (3.2.8)\\ B(0) &= max(0, log\frac{r}{D}) && (3.2.9)\\ lim_{n\to-\infty} u(x,\tau) &= e^{k_1\tau}(1 - e^{(x-(k_2-1)\tau}) && (3.2.10)\\ u(B(\tau),\tau) &= 0 && (3.2.11)\\ u_x(B(\tau),\tau) &= 0 && (3.2.12) \end{aligned}$$

Chapter 4

Numerical Methods

4.1 Finite Difference Based Front Tracking Method

In the PDE 3.2.6 $u = u(\mathbf{x}, t)$ is defined in a fixed frame of reference with coordinate $\mathbf{x}$ and time t. The differential operator L[1] involves space derivatives only.

Instead of working in the fixed(Eulerian) frame it is possible to take a Lagrangian viewpoint in which $\mathbf{x}$ is taken to be a moving coordinate $\mathbf{x}(t)$. We then have a time-dependent mapping from a fixed set of reference coordinates, e.g. $\mathbf{a} = \mathbf{x}(0)$.

If we now define an invertible mapping between the fixed coordinates $\mathbf{a}$ and

[1] $Lu = \dot{u} - \dot{x}u_x - u_{xx}$

the moving coordinates $\mathbf{x}$ at time t

$$\mathbf{x} = \hat{\mathbf{x}}(\mathbf{a}, t) \tag{4.1.1}$$

We have

$$u(\mathbf{x}, t) = u(\hat{\mathbf{x}}(\mathbf{a}, t), t) = \hat{u}(\mathbf{a}, t) \tag{4.1.2}$$

where $\hat{u}$ and $\hat{\mathbf{x}}$ are Eulerian.

Applying the chain rule to 4.1.2 gives

$$\frac{\partial \hat{u}}{\partial t} = \frac{\partial \hat{\mathbf{x}}}{\partial t} \cdot \frac{\partial u}{\partial \hat{\mathbf{x}}} + \frac{\partial u}{\partial t} \tag{4.1.3}$$

From equation 3.2.6 we have that $u_\tau = u_{xx} + g$. Substituting into equation 4.1.3 yields

$$\frac{\partial \hat{u}}{\partial t} = \frac{\partial \hat{\mathbf{x}}}{\partial t} \cdot \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + g(x, \tau) \tag{4.1.4}$$

This is the time dependent equation, the solution of which gives the price for the American call option.

We now discretise the problem using finite difference methods. We let N denote the number dividing the interval of S into equally spaced subintervals.

$$S_i = i\delta S,\ i = 0, \cdots, N \tag{4.1.5}$$

$$\delta S = \frac{B(\tau) - x^-}{N} \tag{4.1.6}$$

L denotes the number dividing the time interval such that

$$\tau_j = j\delta\tau,\ j = 0, \cdots, L \tag{4.1.7}$$

$$\delta\tau = \frac{1}{2}\sigma^2 T/L \tag{4.1.8}$$

The grid used for this numerical scheme is shown in Figure 4.1.

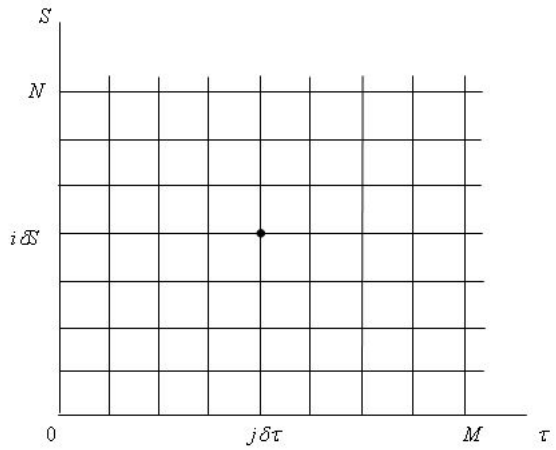

Figure 4.1: Mesh for the finite difference approximation

For an interior point (i, j) on the grid, $\frac{\partial U}{\partial S}$ is approximated by a central difference formula

$$\frac{\partial U}{\partial S} \approx \frac{\partial U_{i+1}^j - U_{i-1}^j}{2\delta S} \tag{4.1.9}$$

To approximate the time derivative $\frac{\partial U}{\partial \tau}$ we use a forward difference approximation

$$\frac{\partial U}{\partial \tau} \approx \frac{U_i^{j+1} - U_i^j}{\delta \tau} \tag{4.1.10}$$

The second spatial derivative,$\frac{\partial^2 U}{\partial S^2}$ is approximated by

$$\frac{\partial^2 U}{\partial S^2} \approx \frac{U_{i-1}^j - 2U_i^j + U_{i+1}^j}{(\delta S)^2} \tag{4.1.11}$$

Finally, we approximate the 'nodal velocity', $\dot{S}$ by

$$\frac{\partial S}{\partial \tau} \approx \frac{S_i^{j+1} - S_i^j}{\delta \tau} \tag{4.1.12}$$

We first discretised the PDE 4.1.4

$$\begin{aligned}\frac{U_i^{j+1}-U_i^j}{\delta\tau} &= \theta_1\left(\frac{U_{i-1}^{j+1}-2U_i^j+U_{i+1}^{j+1}}{\delta S^2}\right)+\theta_2\left(\frac{U_{i-1}^{j}-2U_i^j+U_{i+1}^{j}}{\delta S^2}\right)\\ &+ \left[\left(\frac{U_i^j-U_{i-1}^j}{\delta S}\frac{S_i^{j+1}-S_i^j}{\delta\tau}\right)\right]\\ &+ \left(\theta_3 G_i^{j+1}+\theta_4 G_i^j\right) \qquad (4.1.13)\end{aligned}$$

For $1 \leq j \leq J-1$ and $1 \leq n \leq N-1$. The parameters θ_i control the implicitness of the scheme.

For consistency we must have

$$\theta_1+\theta_2=\theta_3+\theta_4=1 \qquad (4.1.14)$$

As time increases the domain expands with $B(\tau)$. The grid is appropriately expanded by first determining the position of the free boundary then dividing the domain into equal linearly spaced grid points. i.e if we let x_N^{j+1} denote the position of the free boundary, $x_f(\tau)$, then the grid points at the $j+1$ time step are defined by $x_i^{j+1}=x^-+\frac{i}{N}\left(x_N^{j+1}-x^-\right)$ where $i=1,2,\ldots,N$. Differentiating gives the relation

$$\dot{x}_i=\frac{i}{N}\left(\dot{x}_N\right) \qquad (4.1.15)$$

We use this equation to determine the velocity of each nodal point.

θ-Weighted Finite Difference Discretization

For $\theta=0$ the discretization is explicit, for $\theta=\frac{1}{2}$ we have the Crank-Nicolson scheme, and for $\theta=1$ the method is implicit. In this dissertation we look only at the $\theta=\frac{1}{2}$ case.

Re-arranging equation 4.1.13 we obtain

$$\begin{aligned} U_i^{j+1} - U_i^j &= \alpha_i \left[\theta_1 \left(U_{i-1}^{j+1} - 2U_i^{j+1} + U_{i+1}^{j+1}\right) + \theta_2 \left(U_{i-1}^j - 2U_i^j + U_{i+1}^j\right)\right] \\ &+ \beta_i \left[\theta_3 G_i^{j+1} + \theta_4 G_i^j\right] + \gamma_i \left[\left(U_i^j - U_{i-1}^j\right)\left(X_N^{j+1} - X_N^j\right)\right] \end{aligned}$$

Where

$\alpha_i = \frac{\delta\tau}{(\delta S)^2} > 0,\ \beta_i = 2k > 0,\ \gamma_i = \frac{i}{N\delta S} > 0$

Rearranging 4.1.16 we are left with

$$\begin{aligned} c_i U_{i-1}^{j+1} + a_i U_i^{j+1} + b_i U_{i+1}^{j+1} + f_i (U_i^j - U_{i-1}^j) X_N^{j+1} &= \acute{c}_i U_{i-1}^j + \acute{a}_i U_i^j + \acute{b}_i U_{i+1}^j \\ &+ \acute{f}_i (U_i^j - U_{i-1}^j) X_N^j + e_i G_i^{j+1} + \acute{e}_i G_i^j \end{aligned}$$

where

$$\begin{aligned} c_i &= -\alpha_i\theta_1 & \acute{c}_i &= \alpha_i\theta_2 \\ a_i &= 1 + 2\alpha_i\theta_1 & \acute{a}_i &= 1 - 2\alpha_i\theta_2 \\ b_i &= -\alpha_i\theta_1 & \acute{b}_i &= \alpha_i\theta_2 \\ e_i &= 2\theta_1\beta_i & \acute{e}_i &= 2\theta_2\beta_i \\ f_i &= \theta_1\gamma_i & \acute{f}_i &= \theta_2\gamma_i \end{aligned} \tag{4.1.16}$$

The problem is then reduced to solving the system of equations

$$TU^{j+1} + \vec{\beta} X_N^{j+1} = BU^j + \vec{d} \tag{4.1.17}$$

In order to find the location of free boundary at each successive time step, we require one more piece of information. This is given by the derivative boundary conditions 4.1.17. The condition $\frac{\partial C(B(\tau),\tau)}{\partial x} = 0$ gives one extra equation, namely $u_{N-1} = u_N$. Writing this as a matrix equation with **??**, we have the equations

$$\begin{aligned} T\vec{u}^{j+1} + \vec{\beta} x_N &= B\vec{u}^j + \vec{d} \\ h^T \vec{u} &= 0 \end{aligned} \tag{4.1.18}$$

Where the components of T,B,d and β are given by

$$T = \begin{bmatrix} 2+2r & -r & 0 & \cdots & 0 \\ -r & 2+2r & -r & & \\ 0 & -r & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & -r \\ 0 & \cdots & 0 & -r & 2+2r \end{bmatrix} \tag{4.1.19}$$

$$B = \begin{bmatrix} 2-2r & r & 0 & \cdots & 0 \\ r & 2-2r & r & & \\ 0 & r & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & r \\ 0 & \cdots & 0 & r & 2-2r \end{bmatrix} \tag{4.1.20}$$

$$\begin{aligned} d_i &= \tfrac{1}{2}\Big(g(x^- + ih, j\Delta\tau) + g(x^- + ih, (j+1)\Delta\tau)\Big) - \left(\frac{u_i^j - u_{i-1}^j}{x_i - x_{i-1}}\right)\left(\frac{i}{N}\right)x_N^j \\ \beta_i &= -\frac{i}{N}\frac{u_i^j - u_{i-1}^j}{x_i^j - x_{i-1}^j} \end{aligned} \tag{4.1.21}$$

$$h^T = 0\ 0\ \cdots\ \cdots - 1\ 1 \tag{4.1.22}$$

To simplify the notation we absorb the known quantity Bu^j into the d vector. Writing this matrix equation explicitly we now have

$$\begin{bmatrix} 2-2r & -r & 0 & \cdots & 0 & -\beta_1 \\ -r & 2-2r & -r & & \cdots & -\beta_2 \\ 0 & -r & \ddots & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & -r & \vdots \\ 0 & \cdots & 0 & -r & 2-2r & -\beta_{N-1} \\ 0 & 0 & \cdots & -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ \vdots \\ u_{N-1}^j \\ x_N^{j+1} \\ 0 \end{bmatrix} = \begin{bmatrix} \acute{d}_1 \\ \acute{d}_2 \\ \vdots \\ \vdots \\ \acute{d}_{N-1} \\ 0 \\ 0 \end{bmatrix}$$

This may be written symbolically as

$$\begin{pmatrix} \mathbf{T} & \vec{\beta} \\ \vec{h}^T & 0 \end{pmatrix} \begin{pmatrix} \vec{u} \\ x_N^{j+1} \end{pmatrix} = \begin{pmatrix} \vec{\acute{d}} \\ 0 \end{pmatrix} \tag{4.1.23}$$

Equation 4.7.5 can now be re-arranged to solve for $x_{_N}$

$$\begin{aligned} T\vec{u} + \vec{\beta} x_N &= \vec{d} \\ \vec{h}^T \vec{u} &= 0 \\ \Rightarrow \vec{u} &= T^{-1}\left(\vec{\acute{d}} - \vec{\beta} x - N\right) \\ \Rightarrow \vec{h}^T \left(T^{-1}\vec{\acute{d}} - T^{-1}\vec{\beta} x_N\right) &= 0 \\ \Rightarrow x_{_N} &= \frac{h^T T^{-1} \tilde{\acute{d}}}{h^T T^{-1} \tilde{\beta}} \end{aligned} \tag{4.1.24}$$

We have therefore defined a method for locating the free boundary $x_f(\tau)$ at each successive time step of the algorithm. Once x_N^{j+1} has been calculated, we may determine the velocity at which the nodes move using the simple equation

$$\dot{x}_i = \frac{i}{N} \dot{x}_N \tag{4.1.25}$$

We may then substitute into equation 4.7.5 to obtain

$$T\vec{u} = \vec{d} - \vec{\beta}x_N^{j+1} \tag{4.1.26}$$

Solving this equation is straightforward, and is accomplished using a tridiagonal solver, where

$$\vec{u} = T^{-1}(\vec{d} - \vec{\beta}x_{_N}) \tag{4.1.27}$$

4.2 Invertbility

We must ensure that in the case of explicit schemes, e.g. Crank-Nicolson, that the matrix T is indeed invertible. To be able to solve equation 4.1.27 we must be able to find the inverse T^{-1}; we consider the following definition

Definition 3. *A tridiagonal matrix A is said to be strictly diagonally dominant (s.d.d) if and only if*

$$|a_i| > |c_i| + |b_i|$$

The $a_i's$ are the coefficients along the diagonal and the $c_i's$ and the $b_i's$ are the coefficients on the lower and upper diagonal respectively. Then the matrix A is non-singular.

If we consider the matrix T in the Crank-Nicolson, $\theta = \frac{1}{2}$ scheme, the coefficients are

$$c_i = -\alpha_i\theta_1$$

$$a_i = 1 + 2\alpha_i\theta_1$$

$$b_i = -\alpha_i\theta_1$$

Clearly

$$1 + 2\alpha_i\theta_1 > 2\alpha_i\theta_1 \Rightarrow |a_i| > c_i| + |b_i|$$

Therefore the matrix T is $s.d.d$ and invertible

4.3 Stability Analysis

In this section we analyse the problem of stability of the finite difference calculations that are used to solve equation 3.2.6.

Let Q^j and R^j be two solutions of the system of equations $AU^{j+1} = BU^j + \bar{d}$, that have the same inhomogeneous term $\bar{d}$ but with different initial data Q^0 and R^0. Their difference $U^j = Q^j - R^j$ satisfies the homogeneous system of equations and stability is achieved by establishing that

$$AU^{j+1} = BU^j \text{ and } \Rightarrow \left\|W^j\right\| \leq K \left\|W^0\right\|$$

If the constant K is such that $|K| \leq 1$ then the scheme is said to be stable. *Fourier* or *von Neumann's* method is the most precise and useful tool for studying stability in the l_2 norm. The Fourier method expresses the initial values at the mesh points along $t = 0$ in terms of a finite Fourier series, then considers the growth of a function that reduces to this series for $t = 0$ by a seperation of variables method.

Fourier stability analysis can be restrictive however, since it can only be applied to linear problems with constant coefficients and periodic boundary conditions. The problem we are considering, $\dot{u} - u_x\dot{x} = u_{xx} + g$, is not a linear equation. We can still proceed however, by 'linearising' the problem, and applying the stability condition locally at every interior point of the domain.

We begin by making the substitution $U_n^j = \lambda_n e^{ikj\delta S}$ and $X_n^j = \xi_n e^{ikj\delta S}$. The

numerical scheme is given by

$$\begin{aligned}\frac{u_i^{j+1}-u_i^j}{k} &= \theta\left\{\frac{u_{i-1}^{j+1}-2u_i^{j+1}+u_{i+1}^{j+1}}{h^2}+g(x^-+ih,(m+1)k)\right\}\\ &+ (1-\theta)\left\{\frac{u_{i-1}^{j}-2u_i^{j}+u_{i+1}^{j}}{h^2}+g(x^-+ih,mk)\right\}\\ &+ \frac{u_i^j-u_{i-1}^j}{h}\frac{i}{N}\frac{x_N^{j+1}-x_N^j}{k} \qquad (4.3.1)\end{aligned}$$

The term involving $\frac{u_i^j-u_{i-1}^j}{h}\frac{i}{N}\frac{x_N^{j+1}-x_N^j}{k}$ may be linearised by freezing the u_x term $\frac{u_i^j-u_{i-1}^j}{h}$. We also drop the known g function. We may then write

$$\begin{aligned}\left(\lambda_{n+1}\frac{e^{ikj\delta S}-\lambda_n e^{ikj\delta S}}{k}\right) &=\\ &= \frac{1}{2}\left(\frac{\lambda_{n+1}e^{ik(j-1)\delta S}-2\lambda_{n+1}e^{ikj\delta S}+\lambda_{n+1}e^{ik(j+1)\delta S}}{h^2}\right)\\ &+ \frac{1}{2}\left(\frac{\lambda_n e^{ik(j-1)\delta S}-2\lambda_n e^{ikj\delta S}+\lambda_n e^{ik(j+1)\delta S}}{h^2}\right)\\ &+ \beta_i\frac{\xi_{n+1}e^{ijk\delta S}-\xi_n e^{ijk\delta S}}{k} \qquad (4.3.2)\end{aligned}$$

multiplying through by k and $e^{ikj\delta S}$ gives

$$\lambda_{n+1}-\lambda_n = \frac{k}{2h^2}\lambda_{n+1}\left(e^{ik\delta S}-e^{-ik\delta S}-2\right)+\frac{k}{2h^2}\lambda_n\left(e^{ik\delta S}-e^{-ik\delta S}-2\right)+\beta_i\left(\xi_{n+1}-\xi_n\right)$$

Writing $e^{ik\delta S}$ as $2cos(k\delta S)$, using the identity $cos(k\delta S)=1-2sin^2(\frac{k\delta S}{2})$ and writing $\frac{k}{h^2}=\mu$ we obtain

$$\lambda_{n+1}(1-2\mu sin^2(\frac{ik\delta S}{2}))=\lambda_n(1-2\mu sin^2(\frac{ik\delta S}{2}))+(\xi_{n+1}-\xi_n) \qquad (4.3.3)$$

Which holds for $N-1$ interior equations.

We also have the boundary conditions that $\dot{u}_N-\dot{u}_{N-1}=0$ which provides one extra equation

$$\lambda_n - e^{ik\delta S}\lambda_{n-1}=0 \qquad (4.3.4)$$

We therefore have two equations in two unknowns allowing us to solve for amplification factors λ and ξ.

4.4 Local Analysis of the Free Boundary

A graph of the initial date profile is given in figure 2.7. The domain of equation 3.2.6 is $(-\infty, B(0)) \times (0, \frac{1}{2}\sigma^2 T)$. Furthermore, from section 2.7 we know where the free boundary, $x_f(0)$, must start. Clearly, the is a discrepancy between the points where the initial data falls to zero,i.e. x_N^0 and the position of $x_f(0)$.(see fig 4.4).

In moving the curve from $x = 0$ to $x = x_f(0)$ the finite difference algorithm

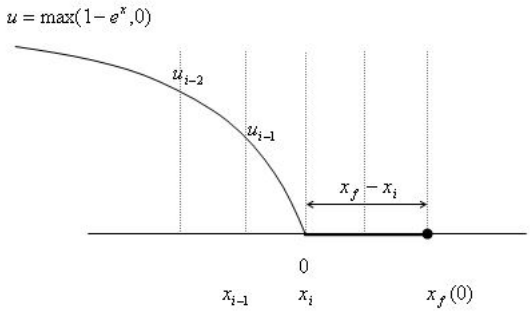

Figure 4.2: Initial Data Curve and Free Boundary

becomes unstable. We must find another way to advance the curve for the first time step of the algorithm. Once this has been achieved, the algorithm can then be used to progress the curve.

To see how the free boundary $x = x_f(\tau)$ initially moves away from $x_f(0)$

we find an asymptotic solution that is valid close to expiry[2] .

Restricting our analysis small values of τ and for x close to x_0 we expand $f(x)$ by a Taylor series about x_0.

$$f(x) = f(x_0) + \acute{f}(x_0)(x - x_0) + O((x - x_0))^2 \tag{4.4.1}$$

$$(x - x_0)\acute{f}(x_0) = -k(x - x_0)$$

We assume an approximate local solution $c(x, \tau)$that satisfies

$$\frac{\partial c}{\partial \tau} = \frac{\partial^2 c}{\partial x^2} - k(x - x_0) \tag{4.4.2}$$

with

$$c = \frac{\partial c}{\partial x} = 0 \tag{4.4.3}$$

Taken on

$$x = x_f(\tau) \tag{4.4.4}$$

$$x_f(0) = x_0 \tag{4.4.5}$$

This local problem can be solved exactly (see appendix). The similarity solution in terms of the variable

$$\xi = \frac{(x - x_0)}{\sqrt{\tau}} \tag{4.4.6}$$

is of the form

$$c = \tau^{3/2} c^*(\xi) \tag{4.4.7}$$

(note about cstar) We also try a free boundary of the form

$$x_f(\tau) = x_0 + \xi_0 \sqrt{\tau} \tag{4.4.8}$$

[2]See Wilmott, Mathematics of Financial Derivatives, section 7.7.2

Where ξ_0 is a constant taken to be 0.9034....

This is approximation to the free boundary motion that we use to expand the grid for the initial time step, i.e. for $i = 1$. The method described in section 4.1 is then used for successive time steps, up until expiry.

4.5 Derivative Boundary Conditions

Equation 4.1.17 gives the condition that the derivative at the moving boundary must be zero. In finite difference notation this can be written

$$u_{N-1} = u_N \tag{4.5.1}$$

Since we also have the boundary condition $u(B(\tau), \tau) = 0$ this implies that u_{N-1} is also zero. A graph of this data is shown in figure 4.5 below.

The existence of a discontinuity in the data, where the function falls to zero,

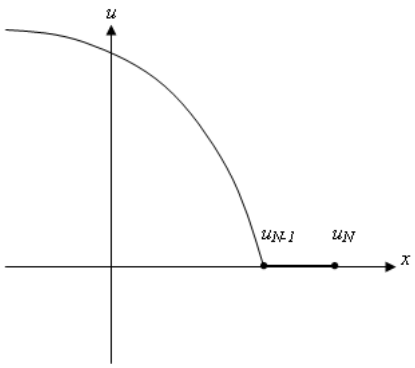

Figure 4.3: Derivative Conditions

causes instability close to the leading edge of the curve. This leads to poor

results in the region local to the moving boundary point.
In an effort to overcome this, we replace the zero derivative requirement by an approximation. We seek a function of the form

$$ay^2 + by = 0 \tag{4.5.2}$$

to approximate the curve close to x_f. Taking the derivative of this function gives

$$2ay + b = 0 \tag{4.5.3}$$

The derivaitve is known to be zero at $y = 0$ which implies that $b = 0$ and hence our curve is modeled by the function ay^2. We have the condition that

$$\begin{aligned} ah^2 &= u_{N-1} \\ a(2h)^2 &= u_{N-2} \end{aligned} \tag{4.5.4}$$

We therefore have a relation that

$$u_{N-2} = 4u_{N-1} \tag{4.5.5}$$

4.6 Description of Algorithm 1

The initial data profile presents two difficulties from a numerical perspective. Firstly, the algorithm is found to be unstable when applied to the function $max(1-exp(x), 0)$. This problem is overcome by utilizing the approximation to the free boundary derived in section 4.4. The second problem arises due to the discontinuous data curve that results from imposing the boundary

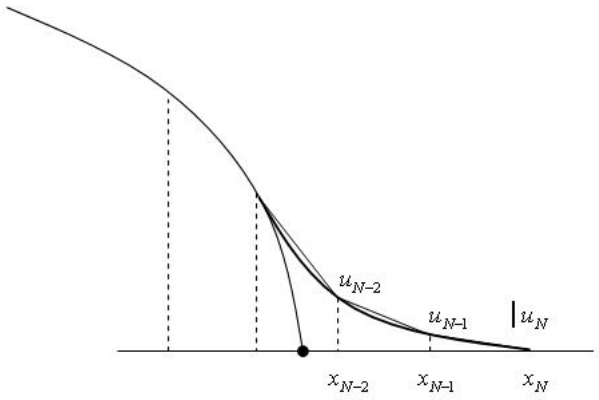

Figure 4.4: Approximation to the Derivative

conditions 4.1.17. At the moving boundary,x_N, the form of the solution is approximated by a parabola .

We now present a brief overview of the algorithm

Set Initial Conditions u^0, B^0, x^0

Approximate the derivative at x_N *by 'parabola'*

For $j = 1$ *DO*

- *Set the Velocity of* x_N *to* $\xi_0\sqrt{\tau}$
- *Rescale* x *grid points*
- *Solve equation 4.1.26 for* u^1
- *Set* $\tau = \tau + \Delta\tau$

END DO

For $j = 2$ *to* $N - 1$ *DO*

- *Solve equation 4.1.24 for* x_N^i

- *Rescale* x *grid points*

- *Approximate the derivative at* x_N *by 'parabola'*

- *Solve equation 4.1.26 for* u_i^j

- *Set* $\tau = \tau + \Delta\tau$

END DO

4.7 Algorithm 2

To improve the accuracy of the algorithm used in section 4.1 we now introduce a monitor function, the effect of which is to increase the number of grid points in the local region(s) where the curve is changing rapidly. Likewise, regions where the data is varying less rapidly will be assigned fewer grid points.
Our aim is to increase the resolution close to the moving boundary point $\beta(\tau)$. To understand the significance of this, we make a transformation back to financial variables. Here, the moving boundary represents a division between two points. Points to the left of the moving boundary represent asset prices for which it would be unprofitable to enter into the option, whereas points to the right are asset prices for which the option would be profitable. It is therefore essential to gain an accurate approximation for this point.
The monitor function is described by

$$x_i = x_f(\tau) - \left(\frac{(x_f(\tau) - x^-)}{N^2} (N - i)^2 \right) \tag{4.7.1}$$

This function gives the position of each nodal point in terms of the free boundary. The velocity of the nodes may be calculated from 4.7.1 as

$$\dot{x_i} = \left(1 - \frac{(N-i)^2}{N^2}\right) \dot{x_f}(\tau) \tag{4.7.2}$$

Algorithm 1 uses equal linearly spaced intervals. The introduction of the

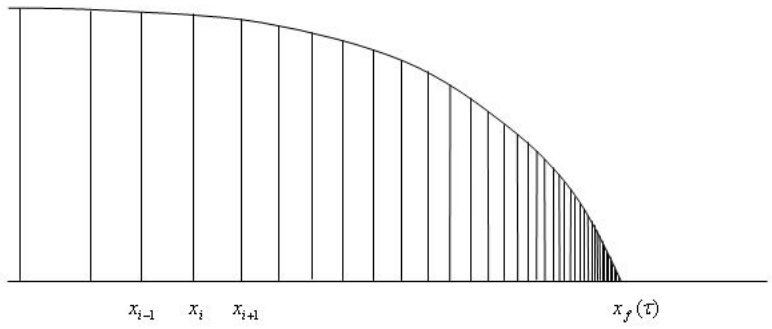

Figure 4.5: Grid Spacing

monitor function leads to a parabolic spacing of the grid points, see fig 4.7. Since the $x-$ spacing is no longer constant between grid points, the Crank-Nicolson method used in the previous algorithm is no longer valid. In order to apply a finite difference discretization we introduce a Lagrange polynomial to approximate the second derivative.

$$\frac{\partial^2 f}{\partial x^2} = \frac{2f(x_0)}{(x_0 - x_1)(x_0)(x_2)} + \frac{2f(x_1)}{(x_1 - x_0)(x_1 - x_2)} + \frac{2f(x_2)}{(x_2 - x_0)(x_2 - x_1)} \tag{4.7.3}$$

The lagrange polynomial is degree two on the support $\{x_0, x_1, x_2\}$ for the function $u(x)$. Replacing the difference approximation to the second deriva-

tive in equation gives

$$\frac{u_i^{j+1}-u_i^j}{\Delta\tau}=\theta\left\{\frac{2u_{i-1}^{j+1}}{(x_{i-1}^{j+1}-x_i^{j+1})(x_{i-1}^{j+1}-x_{i+1}^{j+1})}+\frac{2u_i^{j+1}}{(x_i^{j+1}-x_{i-1}^{j+1})(x_i^{j+1}-x_{i+1}^{j+1})}+\frac{2u_{i+1}^{j+1}}{(x_{i+1}^{j+1}-x_{i-1}^{j+1})(x_{i+1}^{j+1}-x_i^{j+1})}\right\}$$
$$+(1-\theta)\left\{\frac{2u_{i-1}^{j}}{(x_{i-1}^{j}-x_i^{j})(x_{i-1}^{j}-x_{i+1}^{j})}+\frac{2u_i^{j}}{(x_i^{j}-x_{i-1}^{j})(x_i^{j}-x_i^{j})}+\frac{2u_{i+1}^{j}}{(x_{i+1}^{j}-x_{i-1}^{j})(x_{i+1}^{j}-x_i^{j})}\right\}$$
$$+\theta\Big(g(x^-+ih,j\Delta\tau)\Big)+(1-\theta)\Big(g(x^-+ih,(j+1)\Delta\tau)\Big)+\frac{u_i^j-u_{i-1}^j}{x_i^j-x_{i-1}^j}\left(1-\frac{(N-i)^2}{N^2}\right)\frac{x_N^{j+1}-x_N^j}{\Delta\tau}$$

As before this may be written as a matrix equation

$$T\vec{u}+\vec{\beta}x_{_N}=\vec{\acute{d}} \tag{4.7.4}$$
$$h^T\vec{u}=0$$

Where Where the components of T, $\acute{d}$, β and h^T are given by

$$T = \begin{bmatrix} 1-\frac{k}{\acute{\alpha_2}} & -\frac{k}{\acute{\alpha_3}} & \cdots & & \cdots & 0 \\ -\frac{k}{\acute{\alpha_2}} & 1-\frac{k}{\acute{\alpha_3}} & -\frac{k}{\acute{\alpha_4}} & & & \\ 0 & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & & -\frac{k}{\alpha_{\acute{N}-2}} & 1-\frac{k}{\alpha_{\acute{N}-1}} \end{bmatrix} \tag{4.7.5}$$

$$\acute{d_i} = \tfrac{k}{\alpha_{i-1}}u_{i-1}^j+\Big(1+\tfrac{k}{\alpha_i}\Big)u_i^j+\tfrac{k}{\alpha_{i+1}}u_{i+1}^j\}$$
$$\beta_i = -\left(1-\frac{(N-i)^2}{N^2}\right)\frac{u_i^j-u_{i-1}^j}{x_i^j-x_{i-1}^j}$$
$$h^T = 0\ 0\ \cdots\ \cdots\ -1\ \ 1 \tag{4.7.6}$$

Where

$$\alpha_{i-1}=(x_{i-1}^j-x_i^j)(x_{i-1}^j-x_{i+1}^j)$$
$$\alpha_i=(x_i^j-x_{i-1}^j)(x_i^j-x_i^j)$$
$$\alpha_{i+1}=(x_{i+1}^j-x_{i-1}^j)(x_{i+1}^j-x_i^j) \tag{4.7.7}$$

Writing this matrix equation explicitly we have

$$\begin{bmatrix} 1-\frac{k}{\acute{\alpha_2}} & -\frac{k}{\acute{\alpha_3}} & 0 & \cdots & & & -\beta_1 \\ -\frac{k}{\acute{\alpha_1}} & 1-\frac{k}{\acute{\alpha_3}} & -\frac{k}{\acute{\alpha_4}} & \cdots & & & -\beta_2 \\ 0 & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \dots & \dots & & -\frac{k}{\acute{\alpha_{N-2}}} & 1-\frac{k}{\acute{\alpha_{N-1}}} & -\beta_{N-1} \\ 0 & 0 & \cdots & \cdots & -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ \vdots \\ u_{N-1}^j \\ x_N \end{bmatrix} = \begin{bmatrix} \acute{d_1} \\ \acute{d_2} \\ \vdots \\ \vdots \\ \acute{d}_{N-1} \\ 0 \end{bmatrix}$$

The solution method for x_N remains unchanged. We solve for x_N as before, and express the solution as $\vec{u} = T^{-1}(\vec{\acute{d}} - \vec{\beta} x_N)$.

4.8 Invertbility-method two

We proceed as previously outlined in section 4.2

$$c_i = -\frac{k}{\acute{\alpha}_{i-1}} a_i = (1 - \frac{k}{\acute{\alpha}_i}) b_i = -\frac{k}{\acute{\alpha}_{i+1}}$$

Again, we need to ensure $|a_i| > |b_i| + |c_i|$

$$\Rightarrow (1 - \frac{k}{\acute{\alpha}_i}) > -\frac{k}{\acute{\alpha}_{i-1}} - \frac{k}{\acute{\alpha}_{i+1}}$$

This is restriction imposed on the grid spacing

4.9 Description of Algorithm 2

The introduction of the monitor function, reducing the spatial interval near the moving boundary, modifies the T matrix given in equation 4.7.6. The 'r'

values, $\frac{k}{h^2}$, are no longer a constant and must be evaluated at each pair of nodes within the domain. In our modified algorithm, the T matrix contains entries $\frac{k}{\alpha}$ where α is equivalent to h in our previous method when the domain was divided into linear elements. Clearly as the spacing becomes smaller, the magnitude of $\frac{k}{\alpha}$ becomes larger. When the ration $\frac{k}{\alpha}$ becomes larger than 1 the solution method is found to become unstable. To rectify this we precondition the matrices as follows;

Let D be the diagonal of T. Multiply both sides of equation 4.1.26 by D^{-1} to obtain

$$D^{-1}T\vec{u} = D^{-1}\vec{d} - D^{-1}\vec{\beta}x_N^{j+1} \tag{4.9.1}$$

To simplify the notation we let $TD^{-1} = \acute{T}$ and $D^{-1}\vec{d} - D^{-1}\vec{\beta}x_N^{j+1} = \acute{f}$. We then solve the equation

$$u = \acute{T}^{-1}\acute{f} \tag{4.9.2}$$

We now present a brief description of the algorithm

Set Initial Conditions u^0, B^0, x^0

Approximate the derivative at x_N *by 'parabola'*

For $j = 1$ *DO*

- *Set the Velocity of* x_N *to* $\xi_0\sqrt{\tau}$
- *Rescale* x *grid points using the monitor function*
- *Calculate* β_i *and construct* T *matrix*
- *Multiply equation 4.1.26 by* D^{-1}
- *Solve equation 4.9.2 for* u^1

- *Set* $\tau = \tau + \Delta\tau$

END DO

For $j = 2$ *to* $N - 1$ *DO*

- *Solve equation 4.1.24 for* x_N^i

- *Rescale* x *grid points using the monitor function*

- *Approximate the derivative at* x_N *by 'parabola'*

- *Calculate* β_i *and construct* T *matrix*

- *Multiply equation 4.1.26 by* D^{-1}

- *Solve equation 4.9.2 for* u_i^j

- *Set* $\tau = \tau + \Delta\tau$

END DO

Chapter 5

Finite Element Method

5.1 Introduction

In this section we attempt to find a solution to the American call problem using a finite element method. The previous approach has been to replace the continuous operation of differentiation with the discrete operation of finite differences. We then reformulate the equations in terms of finite differences. The fundamental idea of the finite element method is the replacement of continuous functions by piecewise approximations. The most elementary choice of basis functions is the piecewise linear polynomials, this is the approximation we shall use in this report.

We divide the interval $(-\infty, B(0))$ into linearly spaced elements. The partition is defined by

$$\left[x^{-} = x_0, x_1, \cdots, x_{j-1}, x_j, x_{j+1}, \cdots, x_{N+1} = x_f\right] \tag{5.1.1}$$

Define the basis functions ('hat' functions) as follows

$$\phi_i(x) = \begin{cases} 0 & : \quad 0 \leq x \leq x_{i-1} \\ \frac{x-x_{i-1}}{x_i-x_{i-1}} & : \quad x_i < x \leq x_{i+1} \\ \frac{x_{i+1}-x}{x_{i+1}-x_i} & : \quad x_i < x \leq x_{i+1} \\ 0 & : \quad x_{i+1} < x \leq 1 \end{cases}$$

The functions ϕ_i are piecewise linear, the derivatives $\acute{\phi}_i$ are constant on (x_i, x_{i+1}) for each $i = 0, 1, \cdots, n$.

$$\acute{\phi}_i(x) = \begin{cases} 0 & : \quad 0 \leq x \leq x_{i-1} \\ \frac{1}{h_{i-1}} & : \quad x_i < x \leq x_{i+1} \\ \frac{-1}{h_i} & : \quad x_i < x \leq x_{i+1} \\ 0 & : \quad x_{i+1} < x \leq 1 \end{cases}$$

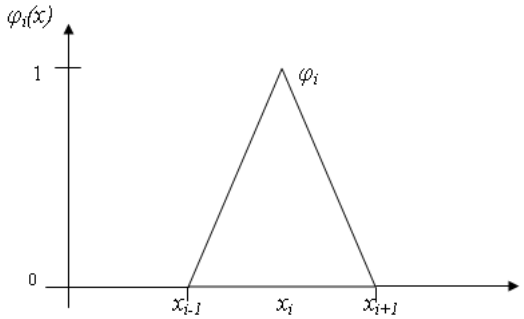

Figure 5.1: Hat Function

Finally let

$$U(x) = \sum_{i=0}^{N+1} U_i \phi_i(x) \tag{5.1.2}$$

be a trial function, once differentiable between each x_i. This is the piecewise linear *Finite Dimensional Representation.*

5.2 Weak Form

The weak form of the differential equation 3.2.6 is found as follows.

Let $\phi_i(x)$ be a set of test functions where $\phi_i(x) \in C^1$. Multiply 3.2.6 by ϕ_i and integrate.

$$\int_{x_{i-1}(\tau)}^{x_{i+1}(\tau)} \phi_i u_t dx \;\; = \;\; \int_{x_{i-1}(\tau)}^{x_{i+1}(\tau)} \phi_i \left(u_{xx} + g\right) dx \tag{5.2.1}$$

$$\tag{5.2.2}$$

Since the grid is not fixed

$$\frac{d}{dt}\int_{x_{i-1}(\tau)}^{x^{i+1}(\tau)} \phi_i u dx = \int_{x_{i-1}(\tau)}^{x^{i+1}} \phi_i u_t + \left[\phi_i u \frac{dx}{dt}\right]_{x_{i-1}(\tau)}^{x_{i+1}(\tau)} \text{Let } \theta = \int_A^{B(\tau)} u dx$$

Now θ, the area under the curve, is not constant in time, since we have a source term $g(x,\tau)$ that adds and subtracts 'mass'.

$$\text{Let } \int_{x_{i-1}(t)}^{x_{i+1}(t)} \phi_i u dx = c_i \theta \tag{5.2.3}$$

be our new monitor function, where c_i is a fraction such that $\sum_i c_i\theta = 1$. See **??** below We now define $\dot{x}$ to be eqaul to $\frac{d\psi}{dx}$, the 'velocity' potential.

At interior points, $1 \leq i \leq N-1$

$$-\int_{x_{i-1}(t)}^{x_{i+1}} (t) \frac{d\phi_i}{dx} u_x dx + \int_{x_{i-1}(t)}^{x_{i+1}(t)} \phi_i g dx - \int_{x_{i-1}(t)}^{x_{i+1}(t)} u \frac{d\phi_i}{dx}\frac{d\psi}{dx} = c_i \dot{\theta} \tag{5.2.4}$$

At $i = 0$

$$-u_x|_{x=x^-} - \int_{x_0(t)}^{x_2(t)} \frac{d\phi_1}{dx} u_x dx + \int_{x_0}^{x_2} \phi_1 g dx + \phi_i u\dot{x} \;|_{x_{i-1}(t)}^{x_{i+1}(t)} - \int_{x_0(t)}^{x_2(t)} \frac{d\phi_1}{dx} u\dot{x} dx = c_1\dot{\theta} \tag{5.2.5}$$

At $i = N$

$$-\int_{x_{N-1}(t)}^{x_N(t)} \frac{d\phi_{N-1}}{dx} u_x dx + \int_{x_{N-2}(t)}^{x_N(t)} \phi_{N-1} g dx \tag{5.2.6}$$

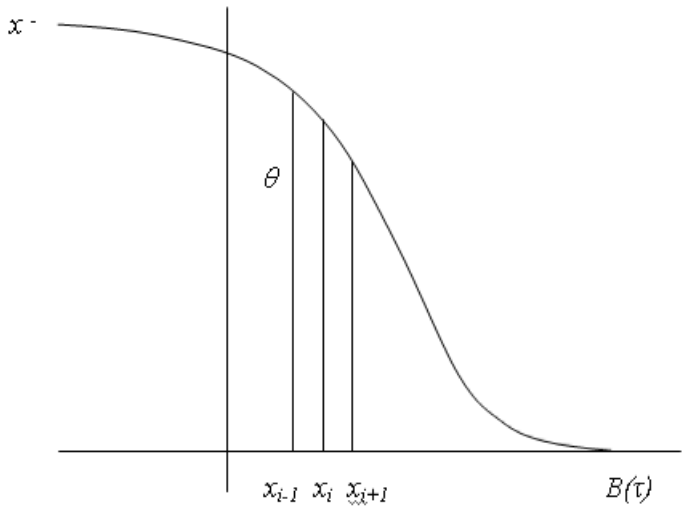

Figure 5.2: Monitor Function

Equation 5.2.4 may be written as a system of equations for ψ and $\dot{\theta}$

$$\acute{K}\psi + \dot{\theta}c = g - Ku \tag{5.2.7}$$

Where

$$\begin{aligned}
\int_{x_{i-1}}^{x_{i+1}} \frac{d\phi_i}{dx}\frac{d\phi_j}{dx}dx &= \acute{\mathbf{K}}_{ij} \\
\int_{x_{i-1}}^{x_{i+1}} \frac{d\phi_i}{dx}u\frac{d\phi_j}{dx}dx &= \mathbf{K}_{ij} \\
\int_{x_{i-1}}^{x_{i+1}} \phi_i g dx &= G_i \\
\int_{x_{i-1}}^{x_{i+1}} \phi_i\phi_j dx &= \mathbf{M}_{ij}
\end{aligned} \tag{5.2.8}$$

the tridiagonal matrix $\mathbf{K}$ is known as the stiffness matrix, while $\mathbf{M}$ is known as the mass matrix.

In order to solve 5.2.9 we need one extra piece of information. This is provided

by the boundary condition $\dot{x} = 0$ at $x = x^-$. This implies that $\frac{d\psi}{dx} = 0$ at $x = x^-$, i.e. $\psi_0 = \psi_1$. Writing this as a matrix equation we have

$$\begin{pmatrix} \acute{\mathbf{K}} & \vec{c} \\ \vec{h}^T & 0 \end{pmatrix} \begin{pmatrix} \vec{\psi} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \vec{f} \\ 0 \end{pmatrix} \tag{5.2.9}$$

The problem is then to solve

$$\vec{\psi} = \acute{K}^{-1}(\vec{g} - K\vec{u} - \dot{\vec{\theta}}) \tag{5.2.10}$$

From 5.2.9 we have that

$$\begin{aligned} \vec{h}^T\vec{\psi} &= 0 \\ \Rightarrow \vec{h}^T\vec{\psi} &= \vec{h}^T\acute{K}^{-1}\vec{g} + \vec{h}^T\acute{K}^{-1}K\vec{u} - \dot{\vec{\theta}}\vec{h}^T\acute{K}^{-1}\vec{c} \\ \Rightarrow \dot{\vec{\theta}} &= \frac{\vec{h}^T\acute{K}^{-1}\vec{g} - \vec{h}^T\acute{K}^{-1}K\vec{u}}{\vec{h}^T\acute{K}^{-1}\vec{c}} \end{aligned} \tag{5.2.11}$$

5.3 Finite Elements Algorithm

In this section we provide an outline of the steps involved in calculating the Finite Element solution of the American Call problem , 4.1.4. In order to evaluate the stiffness matrix $\acute{\mathbf{K}}_{ij}$ we approximate the function u by a linear interpolant between each node i.e.

$$U = u_i\left(\frac{x_{i+1} - x}{x_{i+1} - x_i}\right) + u_{i+1}\left(\frac{x - x_i}{x_{i+1} - x_i}\right) \tag{5.3.1}$$

The steps of the algoritm are outlined below *Set Initial Conditions* u^0, B^0, x^0

For $j = 1$*to* $N - 1$ *DO*

For $i = 1$ *to* $N - 1$ *DO*

- Calculate g_i

- Calculate K_{ij}

- Calculate $\acute{K}_{ij}$ using linear interpolant

- Set $\dot{\vec{\theta}} = \frac{\vec{h}^T \acute{K}^{-1} \vec{g} - \vec{h}^T \acute{K}^{-1} K \vec{u}}{\vec{h}^T \acute{K}^{-1} \vec{c}}$

- Solve $\acute{K} \vec{\psi} = \vec{g} - K\vec{u} - \dot{\vec{\theta}} \vec{c}$

- Calculate nodal velocities $\dot{x}$ by setting $\dot{x} = \frac{d\psi}{dx}$

- Integrate $\dot{x}$ to find new node positions x_i^{j+1}

- Evaluate **M** using new node positions

- Solve $M\vec{u} = \vec{c}$

END DO

END DO

The $\dot{x}$ values are integrated using a Runge-Kutta order four method, $u_{i+1} = u_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + h_4)$.

Elements of the K matrix are calculated exactly whilst the entries in g vector and $\acute{K}$ matrix are evaluated using a numerical quadrature. ($n = 2$: Simpsons Rule). Both the K and $\acute{K}$ matrices are tridiagonal, equation 5.2.10 is found using a tridiagonal solver. (Thomas Algorithm)

Chapter 6

Results

6.1 Finite Differences

In this section we present the numerical results of the finite difference method. We assume the following parameters

$$\begin{aligned} K &= 10 \\ T &= \{0.25, 0.5, 1.0\} \\ r &= \{0.03, 0.06, 0.1\} \\ \sigma &= \{0.2, 0.4, 0.6\} \\ D &= \{0.8r, 1.0r, 1.2r\} \end{aligned} \tag{6.1.1}$$

The smaller values of σ represent lower volatility of the underlying assets, while values of T represent short, medium and long term call options.
We first make the transformation back to financial variables, taking u_i to be our numerical solution to the diffusion problem, we find the solution to our

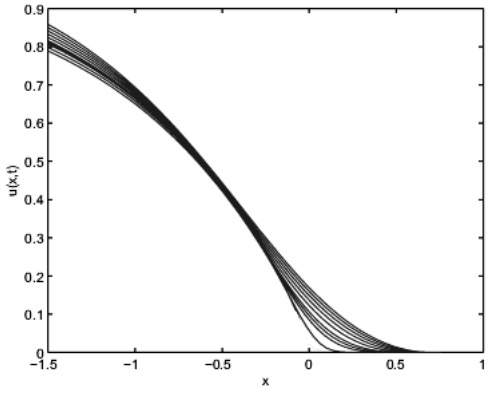

Figure 6.1: $\{T = 1, \sigma = 0.2, r = 0.03, D = 0.8r\}$

option value as follows

$$\begin{aligned} S &= exp^{x-(k_2-1)\tau}K \\ t &= -((2\tau)\sigma^2 - T) \\ V(S,t) &= \frac{u(x,\tau)K}{exp^{k_1\tau}} + S - K \end{aligned} \tag{6.1.2}$$

We apply the numerical scheme for the three combinations of the parameters given in 6.1.1, time to expiry is taken as 1.

Figures 6.1,6.1 and 6.1 show the data curves plotted in the dimensionless (Diffusion) setting. A time step of $\Delta\tau = \frac{\sigma^2}{2}/100$ is taken,the spatial increment,Δh, is 0.001. It is clear from figures that as we vary the parameters the is a recognisable change in the profile of the curve. The contour of the curve is determined predominantly by the source term g which itself is a function of the interest and dividend parameters.

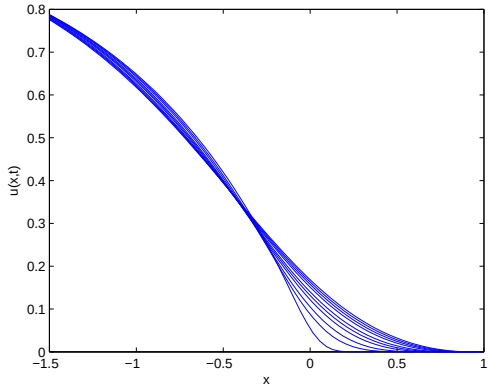

Figure 6.2: $\{T = 1, \sigma = 0.4, r = 0.06, D = 1.0r\}$

From the figures we see that choosing a larger interest rate and dividend parameter causes the free boundary to move further along the $x-$axis, as would be expected. Shown below is a plot of the 'source' term g in the diffusion equation 3.2.6

This function represents a consumption term for $x < 0$ and a replenishment term for $x > 0$. Looking more closely at the function $u(x, \tau)$ in the dimensionless setting (see fig 6.1) we can see for $x > 0$ this function has the effect of decreasing $u(x, \tau)$, drawing out the moving boundary, whilst for $x < 0$ the function adds mass to the equation, causing the curve to increase with time.

Figures 6.1,6.1 and 6.1 show the data once it has been transformed back into financial variables. The small movement of the free boundary in the dimensionless field is now emphasized. The moving boundary, which now represents the point at which we should exercise the option, can be seen

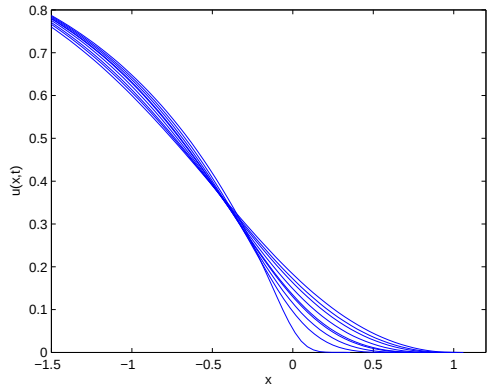

Figure 6.3: $\{T = 1, \sigma = 0.6, r = 0.1, D = 1.2r\}$

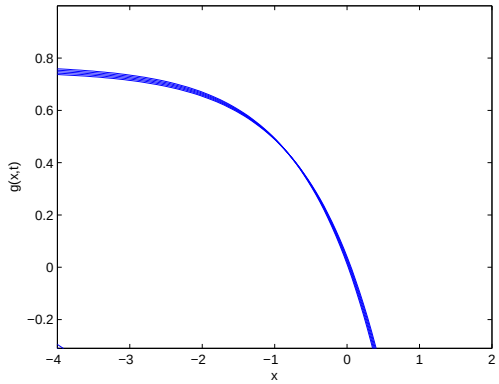

Figure 6.4: $G - function$ $\{T = 1, \sigma = 0.6, r = 0.1, D = 1.2r\}$

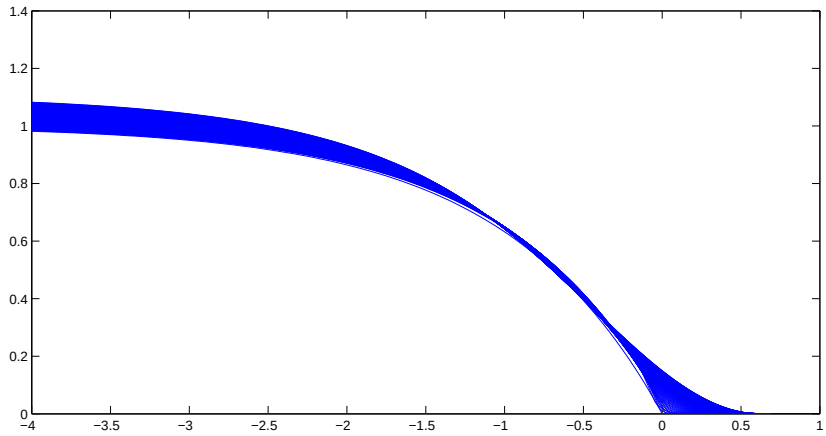

Figure 6.5: $U(x, \tau)$

moving to the right. This is as we would expect. As the time to expiry decreases, there is less time for the price of the underlying asset to move, and thus less potential for the option to become profitable. Consequently the boundary which determines when we should 'hold' or 'sell' the option shifts to the right.

Table 6.1 gives values a comparison of values for a call option for a number of values of the ratio S/K. We take 100 time steps and the parameters: $K = 10, \sigma = 0.3, D = 0.02, r = 0.04, T = 0.25$.

The Benchmark solution is the solution given by the binomial method for 2500 time steps. The money less ration, S/K, is taken in the range $0.7 - 1.3$. The columns IFT,MLII and LC are included for comparison. They represent other PDE methods that have been proposed for the pricing of American options. We also show the relative error between the option value given by BENCH and our approximation.

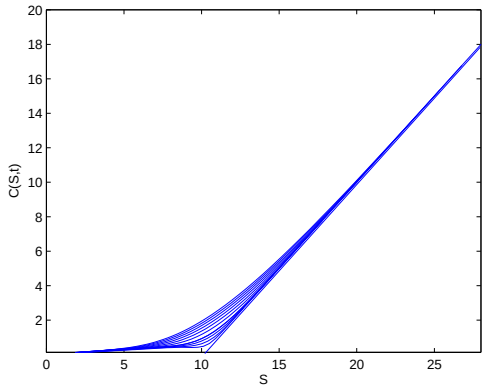

Figure 6.6: $\{T = 1, \sigma = 0.2, r = 0.03, D = 0.8r\}$

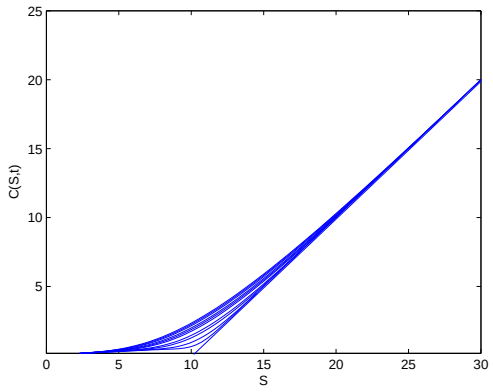

Figure 6.7: $\{T = 1, \sigma = 0.4, r = 0.06, D = 1.0r\}$

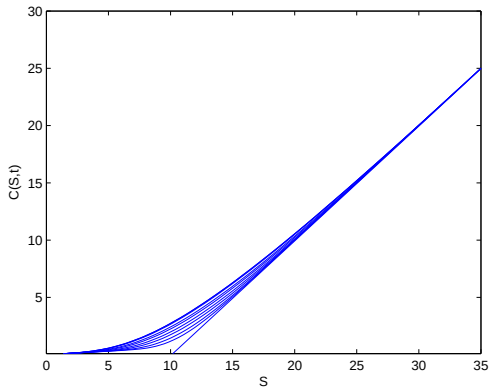

Figure 6.8: $\{T = 1, \sigma = 0.6, r = 0.1, D = 1.2r\}$

Table 6.1: Prices of Call options

S/K	FD	IFT	MLII	LC	BENCH	Error
Call Options						
0.70000	0.003261	0.004025	0.003977	0.004057	0.003988	0.182291
0.75454	0.013500	0.016526	0.016450	0.016577	0.016450	0.178977
0.80909	0.047274	0.051412	0.051360	0.051353	0.051353	0.079435
0.91818	0.117828	0.127973	0.128069	0.127895	0.127893	0.078698
0.97272	0.443699	0.481072	0.481230	0.480072	0.486390	0.087771
1.02727	0.714487	0.775478	0.775818	0.775283	0.775587	0.787790
1.08181	1.056226	1.142981	1.143191	1.142845	1.142953	0.075885
1.13636	1.414501	1.569793	1.569901	1.569956	1.569856	0.099026
1.19090	1.883253	2.040556	2.040556	2.040546	2.040508	0.770665
1.24554	2.364591	2.540893	2.540893	2.541063	2.540901	0.069388
1.30000	2.973134	3.060005	3.060005	3.059990	3.059931	0.029398

The ratio of S/K represents the value of the option, for a given value of S when the strike price is taken to be 10. The term 'moneyness' represents the fact that when the ratio $S/K < 1$ the value of the underlying stock is less than the price of the option, and when the fraction $S/K > 1$ the asset price has risen above the strike price. Referring to table 6.1 we see that as the vale of the underlying stock approaches the strike price the option value begins to increase, and once the stock price actually exceeds the strike, the value of option rises quickly.

The numerical results from the finite difference method are displayed in column 'FD'. The values generated by our method are approximately 25 smaller than the benchmark values. This suggest that our approximation to the free boundary in the dimensionless setting is poor.

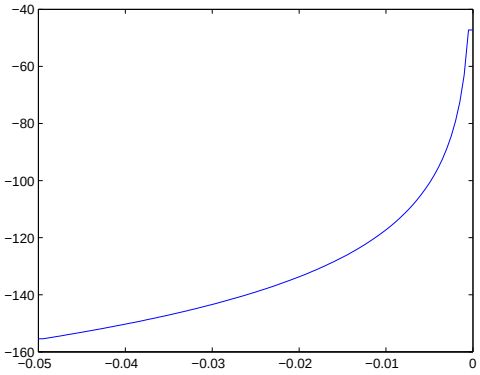

Figure 6.9: Velocity Potential

6.2 Finite Element Algoritm

The results provided by the finite elements algorithm proved to be very poor. The method was unsuccessful in accurately modeling the motion of the free boundary; after a very small number of time steps the velocity of the nodes in the region close to the moving boundary became too large by several orders of magnitude.

In the finite element code, we calculate a velocity potential Ψ, where $\Psi = \Sigma\psi\phi_i$, and then rescale the grid by calculating the nodal velocity as equal to the gradient of the velocity potential. ($\dot{x} = \frac{d\Psi_i}{dt}$. It is then a simple matter of integrating this velocity to obtain the new grid points. A graph of a typical values of Ψvsx is shown below. Ultimately, determining the new node points rests on solving the system of equations

$$\acute{K}\psi = \acute{f}$$

The $\acute{K}$ matrix is given by equation 5.2.8. Because of the nature of the u function, the diagonal entries of the $\acute{K}$ matrix ranged from approximately

40 in size, down to 0.01. This matrix was found to be ill-conditioned, which lead to poor results for ψ.

In an effort to overcome this we take the same approach as section **??**, pre-multiplying by D^{-1} where the D matrix is the sum of the diagonal matrices. This did not solve the problem of the steep gradient close to the free boundary however, and due to time constraints we were unable to obtain any accurate results from this method.

Chapter 7

conclusion

The finite difference method applied to the American Call problem produced valuations to within -30% of the BENCH value. The undervaluing of the option is a result of the algorithms inability to accurately resolve the true position of the moving boundary at each time step. The algorithm was found to be unstable when applying the derivative boundary conditions directly, hence we saught to approximate the derivative by the form of a parabola. Although this improved stability, we believe that this reduces accuracy of the method in determing the moving boundary, the velocity is consistently over approximated.

The introduction of the monitor function improved results, further work is required in analysing the effect of approximation to the boundary conditions, there are also several further monitor functions that can be implemented. A comparison of these methods would be the next step in our research.

We have devised a new method for the valuation of American options as free boundary problems, transforming the equation to a Lagrangian frame, and are also the first to introduce the Shur method for locating the free bound-

ary. (To our Knowledge) Due to time limitations we were unsuccessful in applying the finite element algorithm to the problem. We were unable to resolve the problem caused by the ill-conditioned matrix when evaluating the velocity potentials.

Concluding, further work is required in formulating the finite element approach to this problem and finding an alternative method of calculating the velocity potential . A comparison could then be made between the accuracy of the finite difference method we have presented, and the finite element method that was originally proposed.

Chapter 8

Reference

1 M.J.Baines *A moving mesh finite element algorithm for the adaptive solution of time-dependent partial differential equations with moving boundaries*, Applied Numerical Mathematics 54(2005)450-469

2 F.Black & M.Scholes, *The pricing and Corportate Liabilities*, Journal of Political Economy, Vol. 81, 1973

3 R.L.Burden *Numerical Analysis*,Brooks/Cole 1997

4 L.Clewlow & C.Strickland,*Implementing Derivatives Models*,John Wiley & Sons,1998

5 P.Garrcia-Navarro & A.Priestley, *A Conservative and Shape-Preserving Semi-Lagrangian method for the solution of Shallow Water Equations.* Numerical Analysis Report 6/93, University of Reading, 1993

6 D.J.Higham *An Introduction to Financial Option Valuation.* Cambridge, 2004.

7 J.Hull *Options,Futures, and Other Derivatives*. Prentice-Hall International, 2003

8 A. Priestley, A *Quasi-Conservative Version of the Semi-Lagrangian Advection Scheme*. Numerical Analysis Report 2/92, University of Reading, 1992.

9 K.Singh, *A Comparison of Numerical Schemes for Pricing Bond Options*. Msc Thesis, University of Reading, 1998

10 S.Smith, Submitted PhD thesis, University of Reading, 1996

11 G.D.Smith,*Numerical Solution of Partial Differential Equations*. Oxford, 1985

12 I.M.Smith,*Programming in Fortran 90*,John Wiley & Sons, 1995

13 P.Wilmott *The Mathematics of Finacial Derivatives*. Cambridge 2002.

.1 Program 1 - pt1.f90

```
      PROGRAM AMERICAN_OPTIONS707
            DOUBLE PRECISION::K,NEGINFINITY,XN,Delt,Tfinal,t,B,h,A,C,DelTau,
              Domain,Tau,SOUT
          DOUBLE PRECISION,ALLOCATABLE::U(:),UTEMP(:),RHS(:),Z(:),Y(:),HT(:),V(:),
            X(:),U2D(:,:),X2D(:,:),S2D(:,:),V2D(:,:)
          INTEGER::IOS,j,N,L,i
          DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.5D0,D=0.8D0*Ir,KK=10.0D0
          DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
            sigma**2.0D0)

 8        PRINT*,'*******************************************'
          PRINT*,'*                                         *'
          PRINT*,'*          CRANK_NICOLSON SOLULTION       *'
          PRINT*,'*                                         *'
          PRINT*,'*            DIFFUSION EQUATION           *'
          PRINT*,'*                                         *'
          PRINT*,'*                                         *'
          PRINT*,'*******************************************'
16        PRINT*

          NEGINFINITY=-30.0D0
          Tfinal=((Sigma**2.0D0)/2.0)*1.0D0



          Domain=NEGINFINITY
          N=800
24        L=NINT((Tfinal/(Domain/N)**2))
          print*,L

          k=(Tfinal/L)

          DelTau=(Tfinal)/L

          B=0.0D0
32        XN=B
```

```
      ALLOCATE(U(0:N),UTEMP(0:N),RHS(1:N-1),V(1:N-1),Z(1:N-1),Y(1:N-1),HT(1:N
         -1)&
      ,X(0:N),X2D(0:L,0:N),U2D(0:L,0:N),&
      S2D(0:L,0:N),V2D(0:L,0:N))

      U=0.0D0
      UTEMP=0.0D0
      RHS=0.0D0
      V=0.0D0
      Z=0.0D0
      Y=0.0D0
      HT=0.0D0
      t=0.0D0
      S2D=0.0
      V2D=0.0

      OPEN(UNIT=13,FILE="x.dat",IOSTAT=IOS)
      OPEN(UNIT=14,FILE="U.dat",IOSTAT=IOS)
      OPEN(UNIT=15,FILE="x1.dat",IOSTAT=IOS)
      OPEN(UNIT=16,FILE="U1.dat",IOSTAT=IOS)
      OPEN(UNIT=17,FILE="x2.dat",IOSTAT=IOS)
      OPEN(UNIT=18,FILE="U2.dat",IOSTAT=IOS)
      OPEN(UNIT=19,FILE="x3.dat",IOSTAT=IOS)
      OPEN(UNIT=20,FILE="U3.dat",IOSTAT=IOS)
      OPEN(UNIT=21,FILE="x4.dat",IOSTAT=IOS)
      OPEN(UNIT=22,FILE="U4.dat",IOSTAT=IOS)
      OPEN(UNIT=23,FILE="x5.dat",IOSTAT=IOS)
      OPEN(UNIT=24,FILE="U5.dat",IOSTAT=IOS)
      OPEN(UNIT=25,FILE="x6.dat",IOSTAT=IOS)
      OPEN(UNIT=26,FILE="U6.dat",IOSTAT=IOS)
      OPEN(UNIT=27,FILE="x7.dat",IOSTAT=IOS)
      OPEN(UNIT=28,FILE="U7.dat",IOSTAT=IOS)
      OPEN(UNIT=29,FILE="x8.dat",IOSTAT=IOS)
      OPEN(UNIT=30,FILE="U8.dat",IOSTAT=IOS)
      OPEN(UNIT=31,FILE="x9.dat",IOSTAT=IOS)
      OPEN(UNIT=32,FILE="U9.dat",IOSTAT=IOS)
      OPEN(UNIT=33,FILE="x10.dat",IOSTAT=IOS)
      OPEN(UNIT=34,FILE="u10.dat",IOSTAT=IOS)
```

```
72      OPEN(UNIT=35,FILE="x0.dat",IOSTAT=IOS)
        OPEN(UNIT=36,FILE="u0.dat",IOSTAT=IOS)

        OPEN(UNIT=37,FILE="S2.dat",IOSTAT=IOS)
        OPEN(UNIT=38,FILE="V2.dat",IOSTAT=IOS)
        OPEN(UNIT=39,FILE="S3.dat",IOSTAT=IOS)
        OPEN(UNIT=40,FILE="V3.dat",IOSTAT=IOS)
        OPEN(UNIT=41,FILE="S4.dat",IOSTAT=IOS)
80      OPEN(UNIT=42,FILE="V4.dat",IOSTAT=IOS)
        OPEN(UNIT=43,FILE="S5.dat",IOSTAT=IOS)
        OPEN(UNIT=44,FILE="V5.dat",IOSTAT=IOS)
        OPEN(UNIT=45,FILE="S6.dat",IOSTAT=IOS)
        OPEN(UNIT=46,FILE="V6.dat",IOSTAT=IOS)
        OPEN(UNIT=47,FILE="S7.dat",IOSTAT=IOS)
        OPEN(UNIT=48,FILE="V7.dat",IOSTAT=IOS)
        OPEN(UNIT=49,FILE="S8.dat",IOSTAT=IOS)
88      OPEN(UNIT=50,FILE="V8.dat",IOSTAT=IOS)
        OPEN(UNIT=51,FILE="S9.dat",IOSTAT=IOS)
        OPEN(UNIT=52,FILE="V9.dat",IOSTAT=IOS)
        OPEN(UNIT=53,FILE="S10.dat",IOSTAT=IOS)
        OPEN(UNIT=54,FILE="V10.dat",IOSTAT=IOS)

        OPEN(UNIT=60,FILE="x2D.dat",IOSTAT=IOS)
        OPEN(UNIT=61,FILE="u2D.dat",IOSTAT=IOS)
96
        OPEN(UNIT=62,FILE="S.dat",IOSTAT=IOS)
        OPEN(UNIT=63,FILE="V.dat",IOSTAT=IOS)

        OPEN(UNIT=64,FILE="S1.dat",IOSTAT=IOS)
        OPEN(UNIT=65,FILE="V1.dat",IOSTAT=IOS)


104     OPEN(UNIT=70,FILE="SOUT.dat",IOSTAT=IOS)
        OPEN(UNIT=71,FILE="STIME.dat",IOSTAT=IOS)



        IF(IOS/=0) THEN
```

```
          PRINT*,'Error Occured in Opening The Output File'
112       STOP
      END IF

      !*****************************************************
      !*                    MAIN PROGRAM                   *
      !*                                                   *
      !*****************************************************

120   j=0
      DO i=0,N

        X(i)=NEGINFINITY+(REAL(i)/REAL(N))*(B-NEGINFINITY)

        X2D(j,i)=X(i)

      END DO
128
      DO i=0,N

        WRITE(UNIT=13,FMT='(E12.6)')X(i)
        WRITE(UNIT=60,FMT='(4251E21.6)')X2D(j,i)

      END DO

136   CALL BOUNDARY_CONDITIONS(N,U,X,U2D,j,L)

        DO i=1,(N-4)
          HT(i)=0.0D0
      END DO

      HT(N-3)=9.0D0/4.0D0
      HT(N-2)=-1.0D0
144   HT(N-1)=4.0D0

      j=0
      Tau=(0.5D0*(Sigma**2.0D0))*(1.0D0)*(REAL(j)/REAL(L))
      h=(B-NEGINFINITY)/N
```

```
      CALL CRANK_NICOLSON(N,k,U,RHS,B,X,t,h,DelTau,Tau)
152
        XN=XN+0.9034*SQRT((1.0*(Sigma**2.0D0))*(1.0D0)*(1.0D0/REAL(L)))
        B=XN

      DO i=0,N
          X(i)=NEGINFINITY+(REAL(i)/REAL(N))*(B-NEGINFINITY)
      END DO

160   DO i=0,N
        WRITE(UNIT=13,FMT='(E12.6)')X(i)
      END DO

      CALL TRIDIAG_SOL(N,RHS,U,k,h,NEGINFINITY,X,U2D,j,L,S2D,V2D,tfinal)

      DO j=1,L

168     h=(B-NEGINFINITY)/N

        Tau=((Sigma**2.0D0)/2.0D0)*(1.0D0)*(REAL(j)/REAL(L))


        CALL CRANK_NICOLSON(N,k,U,RHS,B,X,t,h,DelTau,Tau)
        CALL VEC(N,U,V,h)
        CALL TRIDIAG_SOL1(N,RHS,Z,k,h)
176     CALL TRIDIAG_SOL2(N,V,Y,k,h)

        CALL XN_SOLVE(N,Y,Z,HT,XN)

        DO i=1,N-1

          RHS(i)=RHS(i)-V(i)*XN

184     END DO

        h=(XN-NEGINFINITY)/N
```

```
      DO i=0,N

        X(i)=NEGINFINITY+(REAL(i)/REAL(N))*(XN-NEGINFINITY)
192
        X2D(j,i)=X(i)


      END DO

      DO i=0,N

200     WRITE(UNIT=13,FMT='(E12.6)')X(i)
        WRITE(UNIT=60,FMT='(4251E21.6)')X2D(j,i)

      END DO


      CALL TRIDIAG_SOL(N,RHS,U,k,h,NEGINFINITY,X,U2D,j,L,S2D,V2D,tfinal)

208   B=XN
      t=Tfinal-(2.0*Tau/sigma**2)
       SOUT=(exp(XN+(1-K2)*tau))*KK
       WRITE(UNIT=70,FMT='(F12.6)')SOUT
       WRITE(UNIT=71,FMT='(F12.6)')Tau



216   END DO

      WRITE(UNIT=15,FMT='(F12.6)')((X2D(j,i),j=5,5),i=0,N)
      WRITE(UNIT=16,FMT='(F12.6)')((U2D(j,i),j=5,5),i=0,N)
      WRITE(UNIT=17,FMT='(F12.6)')((X2D(j,i),j=10,10),i=0,N)
      WRITE(UNIT=18,FMT='(F12.6)')((U2D(j,i),j=10,10),i=0,N)
      WRITE(UNIT=19,FMT='(F12.6)')((X2D(j,i),j=20,20),i=0,N)
      WRITE(UNIT=20,FMT='(F12.6)')((U2D(j,i),j=20,20),i=0,N)
224   WRITE(UNIT=21,FMT='(F12.6)')((X2D(j,i),j=30,30),i=0,N)
      WRITE(UNIT=22,FMT='(F12.6)')((U2D(j,i),j=30,30),i=0,N)
      WRITE(UNIT=23,FMT='(F12.6)')((X2D(j,i),j=40,40),i=0,N)
      WRITE(UNIT=24,FMT='(F12.6)')((U2D(j,i),j=40,40),i=0,N)
```

```
      WRITE(UNIT=25,FMT='(F12.6)')((X2D(j,i),j=50,50),i=0,N)
      WRITE(UNIT=26,FMT='(F12.6)')((U2D(j,i),j=50,50),i=0,N)
      WRITE(UNIT=27,FMT='(F12.6)')((X2D(j,i),j=60,60),i=0,N)
      WRITE(UNIT=28,FMT='(F12.6)')((U2D(j,i),j=60,60),i=0,N)
      WRITE(UNIT=29,FMT='(F12.6)')((X2D(j,i),j=70,70),i=0,N)
      WRITE(UNIT=30,FMT='(F12.6)')((U2D(j,i),j=70,70),i=0,N)
      WRITE(UNIT=31,FMT='(F12.6)')((X2D(j,i),j=80,80),i=0,N)
      WRITE(UNIT=32,FMT='(F12.6)')((U2D(j,i),j=80,80),i=0,N)
      WRITE(UNIT=33,FMT='(F12.6)')((X2D(j,i),j=89,89),i=0,N)
      WRITE(UNIT=34,FMT='(F12.6)')((U2D(j,i),j=89,89),i=0,N)
      WRITE(UNIT=35,FMT='(F12.6)')((X2D(j,i),j=89,89),i=0,N)
      WRITE(UNIT=36,FMT='(F12.6)')((U2D(j,i),j=89,89),i=0,N)

      WRITE(UNIT=37,FMT='(F12.6)')((S2D(j,i),j=5,5),i=0,N)
      WRITE(UNIT=38,FMT='(F12.6)')((V2D(j,i),j=5,5),i=0,N)
      WRITE(UNIT=39,FMT='(F12.6)')((S2D(j,i),j=10,10),i=0,N)
      WRITE(UNIT=40,FMT='(F12.6)')((V2D(j,i),j=10,10),i=0,N)
      WRITE(UNIT=41,FMT='(F12.6)')((S2D(j,i),j=15,15),i=0,N)
      WRITE(UNIT=42,FMT='(F12.6)')((V2D(j,i),j=15,15),i=0,N)
      WRITE(UNIT=43,FMT='(F12.6)')((S2D(j,i),j=20,20),i=0,N)
      WRITE(UNIT=44,FMT='(F12.6)')((V2D(j,i),j=20,20),i=0,N)
      WRITE(UNIT=45,FMT='(F12.6)')((S2D(j,i),j=30,30),i=0,N)
      WRITE(UNIT=46,FMT='(F12.6)')((V2D(j,i),j=30,30),i=0,N)
      WRITE(UNIT=47,FMT='(F12.6)')((S2D(j,i),j=40,40),i=0,N)
      WRITE(UNIT=48,FMT='(F12.6)')((V2D(j,i),j=40,40),i=0,N)
      WRITE(UNIT=49,FMT='(F12.6)')((S2D(j,i),j=50,50),i=0,N)
      WRITE(UNIT=50,FMT='(F12.6)')((V2D(j,i),j=50,50),i=0,N)
      WRITE(UNIT=51,FMT='(F12.6)')((S2D(j,i),j=60,60),i=0,N)
      WRITE(UNIT=52,FMT='(F12.6)')((V2D(j,i),j=60,60),i=0,N)
      WRITE(UNIT=53,FMT='(F12.6)')((S2D(j,i),j=70,70),i=0,N)
      WRITE(UNIT=54,FMT='(F12.6)')((V2D(j,i),j=70,70),i=0,N)

      WRITE(UNIT=64,FMT='(F12.6)')((S2D(j,i),j=80,80),i=0,N)
      WRITE(UNIT=65,FMT='(F12.6)')((V2D(j,i),j=80,80),i=0,N)
```

```
      CLOSE(UNIT=11)
      CLOSE(UNIT=12)
      CLOSE(UNIT=13)
      CLOSE(UNIT=14)
272   CLOSE(UNIT=15)
      CLOSE(UNIT=16)
      CLOSE(UNIT=17)
      CLOSE(UNIT=18)
      CLOSE(UNIT=19)
      CLOSE(UNIT=20)
      CLOSE(UNIT=21)
      CLOSE(UNIT=22)
280   CLOSE(UNIT=23)
      CLOSE(UNIT=24)
      CLOSE(UNIT=25)
      CLOSE(UNIT=26)
      CLOSE(UNIT=27)
      CLOSE(UNIT=28)
      CLOSE(UNIT=29)
      CLOSE(UNIT=30)
288
      CLOSE(UNIT=31)
      CLOSE(UNIT=32)
      CLOSE(UNIT=33)
      CLOSE(UNIT=34)
      CLOSE(UNIT=35)
      CLOSE(UNIT=36)
      CLOSE(UNIT=37)
296   CLOSE(UNIT=38)
      CLOSE(UNIT=39)
      CLOSE(UNIT=40)
      CLOSE(UNIT=41)
      CLOSE(UNIT=42)
      CLOSE(UNIT=43)
      CLOSE(UNIT=44)
      CLOSE(UNIT=45)
304   CLOSE(UNIT=46)
      CLOSE(UNIT=47)
```

```
CLOSE(UNIT=48)
CLOSE(UNIT=49)

CLOSE(UNIT=64)
CLOSE(UNIT=65)

312

END PROGRAM AMERICAN_OPTIONS707

!*****************************************************
!*                                                   *
!*          FUNCTIONS AND SUBROUTINES                *
!*                                                   *
320 !*****************************************************


SUBROUTINE BOUNDARY_CONDITIONS(N,U,X,U2D,j,L)
IMPLICIT NONE
INTEGER,INTENT(IN)::N
DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::X
DOUBLE PRECISION,DIMENSION(0:N),INTENT(OUT)::U
328 DOUBLE PRECISION,DIMENSION(0:L,0:N),INTENT(OUT)::U2D
INTEGER::i
INTEGER,INTENT(IN)::j,L

DO i=0,N

  U(i)=max(1.0D0-EXP(X(I)),0.0D0)

336

  U2D(j,i)=U(i)


END DO

DO i=0,N

344   WRITE(UNIT=14,FMT='(E12.6)')U(i)
```

```
      WRITE(UNIT=61,FMT='(4251E21.6)')U2D(j,i)

      END DO

      END SUBROUTINE BOUNDARY_CONDITIONS

      SUBROUTINE CRANK_NICOLSON(N,k,U,RHS,B,X,t,h,DelTau,Tau)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,INTENT(IN)::h,k,B,t,DelTau,Tau
      DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::U,X
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::RHS
      DOUBLE PRECISION,EXTERNAL::G
      DOUBLE PRECISION::r,TEST
      INTEGER::i

      r=k/(h**2.0D0)

      DO i=1,N-1

        RHS(i)=r*U(i-1)+(2.0D0-2.0D0*r)*U(i)+r*U(i+1)+1.0*k*(G(X(i),Tau)+G(X(
          i),Tau+DelTau))&
        +2.0D0*(REAL(i)/REAL(N))*((U(i)-U(i-1))/h)*(-B)

      END DO


      END SUBROUTINE CRANK_NICOLSON

      SUBROUTINE VEC(N,U,V,h)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::U
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::V
      DOUBLE PRECISION,INTENT(IN)::h

      INTEGER::i
```

```
      DO i=1,N-1
384
            V(i)=-2.0D0*(REAL(i)/REAL(N))*(U(i)-U(i-1))/h

      END DO

      END SUBROUTINE VEC

      DOUBLE PRECISION FUNCTION G(x,Tau)
392   IMPLICIT NONE
      DOUBLE PRECISION,INTENT(IN)::x,Tau
      DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.5D0,D=0.8D0*Ir,KK=10.0D0
      DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
         sigma**2.0D0)

      G=(EXP(K1*Tau))*(((K2-K1)*EXP(x-(K2-1.0D0)*Tau))+K1)

      END FUNCTION G
400

      SUBROUTINE TRIDIAG_SOL(N,RHS,U,k,h,NEGINFINITY,X,U2D,j,L,S2D,V2D,tfinal)
      IMPLICIT NONE
      DOUBLE PRECISION,INTENT(IN)::NEGINFINITY,h,k,tfinal
      INTEGER,INTENT(IN)::N,j,L
      DOUBLE PRECISION,DIMENSION(0:N)::Alpha,s,y
      DOUBLE PRECISION,DIMENSION(0:N),INTENT(OUT)::U
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::RHS
408   DOUBLE PRECISION,DIMENSION(0:L,0:N),INTENT(INOUT)::U2D
      DOUBLE PRECISION,DIMENSION(0:L,0:N),INTENT(INOUT)::S2D,V2D
      DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::X
      DOUBLE PRECISION,DIMENSION(0:N)::RES,SS
      DOUBLE PRECISION::a,b,c,Tau,r,Sv,t
      INTEGER::i,e

      DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.5D0,D=0.8D0*Ir,KK=10.0D0
416   DOUBLE PRECISION,PARAMETER::K1=(2.0D0*Ir)/(sigma**2.0D0),K2=(2.0D0*(Ir-D
         ))/(sigma**2.0D0)

      Alpha=0.0
      s=0.0
```

```
      y=0.0

      r=k/(h**2)

424   a=r
      b=(2.0D0+2.0D0*r)
      c=r
      Alpha(0)=b
      S(0)=RHS(1)

      Tau=((sigma**2.0D0)/2.0D0)*(1.0D0)*(REAL(j)/REAL(L))

432

      y(0)=exp(K1*Tau)*(1.0D0-exp(X(0)-(K2-1.0D0)*tau))

      DO i=1,(N-3)


         Alpha(i)=b-(a*c/Alpha(i-1))
         S(i)=RHS(i)+(a*S(i-1)/Alpha(i-1))
440
      END DO

      y(N-3)=S(N-3)/Alpha(N-3)
      y(N-2)=(4.0D0/9.0D0)*y(N-3)
      y(N-1)=(1.0D0/4.0D0)*y(N-2)
      y(N)=0.0

448   DO i=(N-4),1,-1
         y(i)=(s(i)+c*y(i+1))/Alpha(i)
      END DO

      Sv=7.0D0

      DO i=0,N
         t=Tfinal-(2.0*Tau/sigma**2)
456      U(i)=y(i)
```

```
      END DO
      DO  i=0,N

         SS(i)=(exp(x(i)+(1-K2)*tau))*KK

464


         S2D(j,i)=SS(i)
         RES(i)=((U(i)*KK)/(exp(K1*(tau))))+SS(i)-KK

         V2D(j,i)=RES(i)


472      U2D(j,i)=U(i)

      END DO

      DO  i=0,N

        WRITE(UNIT=62,FMT='(E12.6)')SS(i)

480     WRITE(UNIT=63,FMT='(E12.6)')RES(i)

        WRITE(UNIT=61,FMT='(4251E21.6)')U2D(j,i)

        WRITE(UNIT=14,FMT='(E12.6)')U(i)

      END DO

488

      END SUBROUTINE TRIDIAG_SOL

      SUBROUTINE TRIDIAG_SOL1(N,RHS,Z,k,h)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,DIMENSION(1:N-1)::Alpha,S,y
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::Z
496   DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::RHS
      DOUBLE PRECISION,INTENT(IN)::k,h
```

```
      DOUBLE PRECISION::a,b,c,r
      INTEGER::i


      s=0.0
      y=0.0
504
      r=k/(h**2.0D0)

      a=(r)
      b=(2.0D0+2.0D0*r)
      c=(r)
      Alpha(1)=b
      S(1)=RHS(1)
512

      DO i=2,(N-1)


         Alpha(i)=b-(a*c/Alpha(i-1))
         S(i)=RHS(i)+(a*S(i-1)/Alpha(i-1))

520   END DO

      y(N-1)=S(N-1)/Alpha(N-1)

      DO i=(N-2),1,-1

         y(i)=(S(i)+c*y(i+1))/Alpha(i)

528   END DO

      DO i=1,N-1

        Z(i)=y(i)

      END DO

536
```

```
      END SUBROUTINE TRIDIAG_SOL1

      SUBROUTINE TRIDIAG_SOL2(N,V,Y,k,h)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,INTENT(IN)::k,h
      DOUBLE PRECISION,DIMENSION(1:N-1)::Alpha,s,U
544   DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::Y
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::V
      DOUBLE PRECISION::a,b,c,r
      INTEGER::i



      s=0.0
552   Y=0.0
      U=0.0

      r=k/(h**2.0D0)

      a=(r)
      b=(2.0D0+2.0D0*r)
      c=(r)
560
      Alpha(1)=b
      S(1)=V(1)

      DO i=2,N-1

         Alpha(i)=b-(a*c/Alpha(i-1))
         S(i)=V(i)+(a*S(i-1)/Alpha(i-1))
568
      END DO

        U(N-1)=s(N-1)/Alpha(N-1)

       DO i=(N-2),1,-1

         U(i)=(S(i)+c*U(i+1))/Alpha(i)
```

```
576
        END DO

        DO i=1,N-1

            Y(i)=U(i)

        END DO
584

        END SUBROUTINE TRIDIAG_SOL2

        SUBROUTINE XN_SOLVE(N,Y,Z,HT,XN)
        IMPLICIT NONE
        INTEGER,INTENT(IN)::N
        DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::Y,Z,HT
592     DOUBLE PRECISION,INTENT(OUT)::XN
        DOUBLE PRECISION::A,B
        INTEGER::i

        A=0.0
        B=0.0
        DO i=1,N-1

600         A=A+Ht(i)*Z(i)
            B=B+Ht(i)*Y(i)

        END DO

        XN=A/B

    END SUBROUTINE XN_SOLVE
```

.2 Program 2 - pt2.f90

```
PROGRAM movingr
        DOUBLE PRECISION::K,NEGINFINITY,XN,Delt,Tfinal,t,h,A,C,KN,BN,left,
          domain
```

```
      DOUBLE PRECISION,ALLOCATABLE::U(:),UTEMP(:),RHS(:),Z(:),Y(:),HT(:),V(:),
     X(:),r1(:),r2(:),r3(:)
      INTEGER::IOS,j,N,L,i
      DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.2D0,D=0.8D0*Ir,KK=10.0D0
      DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
     sigma**2.0D0)

      PRINT*,'*****************************************'
      PRINT*,'*                                       *'
      PRINT*,'*          CRANK_NICOLSON SOLULTION     *'
      PRINT*,'*                                       *'
      PRINT*,'*            DIFFUSION EQUATION         *'
      PRINT*,'*                                       *'
      PRINT*,'*                                       *'
      PRINT*,'*****************************************'
      PRINT*


      NEGINFINITY=-30.0D0
      Tfinal=((Sigma**2.0D0)/2.0)*1.0


      Domain=NEGINFINITY
      N=300
      L=NINT((Tfinal/(Domain/REAL(N))**2))
      PRINT*,(Tfinal/(Domain/REAL(N))**2)


      k=Tfinal/L
      Left=-20.0D0
      N=100
      t=0.0D0
      Delt=1.0D0/L
      BN=0.0D0
      KN=((BN-LEFT)/N**2.0D0)
```

```
      ALLOCATE(U(0:N),UTEMP(0:N),RHS(1:N-1),V(1:N-1),Z(1:N-1),Y(1:N-1),HT(1:N
     -1),X(0:N),&
40    r1(1:N-1),r2(1:N-1),r3(1:N-1))

      U=0.0D0
      UTEMP=0.0D0
      RHS=0.0D0
      V=0.0D0
      Z=0.0D0
      Y=0.0D0
48    HT=0.0D0
      t=0.0D0



      OPEN(UNIT=11,FILE="x.dat",IOSTAT=IOS)
      OPEN(UNIT=12,FILE="U.dat",IOSTAT=IOS)

56    IF(IOS/=0) THEN
        PRINT*,'Error Occured in Opening The Output File'
        STOP
      END IF

      !************************************************
      !*                                              *
      !*                 MAIN PROGRAM                 *
64    !*                                              *
      !************************************************



      X(0)=LEFT
      WRITE(UNIT=11,FMT='(E12.6)')X(0)
        DO i=1,N-1
72        X(i)=BN-(KN*(N-i)**2.0D0)
          WRITE(UNIT=11,FMT='(E12.6)')X(i)
        END DO
      X(N)=BN
      WRITE(UNIT=11,FMT='(E12.6)')X(N)
```

```
      CALL BOUNDARY_CONDITIONS(N,U,X)

80          DO i=1,(N-4)
              HT(i)=0.0D0
      END DO

      HT(N-3)=9.0D0/4.0D0
      HT(N-2)=-1.0D0
      HT(N-1)=4.0D0

88    t=1.0D0
      XN=BN
      DO j=0,L

        t=t-delt

        tau=((sigma**2.0D0)/2.0D0)*(1.0D0-t)

96        DO i=1,(N-1)
            r1(i)=((x(i-1)-x(i))*(x(i-1)-x(i+1)))
            r2(i)=((x(i)-x(i-1))*(x(i)-x(i+1)))
            r3(i)=((x(i+1)-x(i-1))*(x(i+1)-x(i)))
        END DO


        CALL CRANK_NICOLSON(N,k,U,RHS,X,t,h,delt,r1,r2,r3,XN)
104     CALL VEC(N,U,V,h,X)
        CALL TRIDIAG_SOL1(N,RHS,Z,k,r1,r2,r3)
        CALL TRIDIAG_SOL2(N,V,Y,k,h,r1,r2,r3)

        CALL XN_SOLVE(N,Y,Z,HT,XN)

        BN=XN

112

        DO i=1,N-1
          RHS(i)=RHS(i)-V(i)*XN
        END DO
```

```
        KN=((BN-LEFT)/N**2.0D0)

        X(0)=LEFT
120     WRITE(UNIT=11,FMT='(E12.6)')X(0)
        DO i=1,N-1
            X(i)=BN-(KN*(N-i)**2.0D0)
            WRITE(UNIT=11,FMT='(E12.6)')X(i)
        END DO

        X(N)=BN
        WRITE(UNIT=11,FMT='(E12.6)')X(N)
128
      DO i=1,(N-1)
            r1(i)=((x(i-1)-x(i))*(x(i-1)-x(i+1)))
            r2(i)=((x(i)-x(i-1))*(x(i)-x(i+1)))
            r3(i)=((x(i+1)-x(i-1))*(x(i+1)-x(i)))

      END DO

136   CALL TRIDIAG_SOL(N,RHS,U,t,k,NEGINFINITY,r1,r2,r3,x)    ! Solve for
        new U


      END DO



    CLOSE(UNIT=11)
144 CLOSE(UNIT=12)

    END PROGRAM movingr

    !*****************************************************************
    !*                                                               *
    !*                    FUNCTIONS AND SUBROUTINES                  *
    !*                                                               *
152 !*****************************************************************
```

```
      SUBROUTINE BOUNDARY_CONDITIONS(N,U,X)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::X
      DOUBLE PRECISION,DIMENSION(0:N),INTENT(OUT)::U
160   INTEGER::i

      DO i=0,N
        U(i)=max(1.0D0-EXP(X(I)),0.0D0)
        WRITE(UNIT=12,FMT='(E12.6)')U(i)
      END DO

      END SUBROUTINE BOUNDARY_CONDITIONS
168
      SUBROUTINE CRANK_NICOLSON(N,k,U,RHS,X,t,h,delt,r1,r2,r3,XN)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,INTENT(IN)::h,k,t,delt,XN
      DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::U,X
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::r1,r2,r3
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::RHS
176   DOUBLE PRECISION,EXTERNAL::G
      DOUBLE PRECISION::TEST,tp
      INTEGER::i


      tp=t-delt

      DO i=1,N-1
184     RHS(i)=2.0*r1(i)*U(i-1)+(2.0D0+2.0*r2(i))*U(i)+2.0*r3(i)*U(i+1)+(k)*(
          G(X(i),t)+G(X(i),tp))&
        +(1.0D0-(((N-i)**2.0D0)/N**2.0D0))*((U(i)-U(i-1))/(x(i)-x(i-1)))*(-XN
          )*2.0
        !PRINT*,i,RHS(i)

      END DO

      END SUBROUTINE CRANK_NICOLSON
```

```
192     SUBROUTINE VEC(N,U,V,h,X)
        IMPLICIT NONE
        INTEGER,INTENT(IN)::N
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::U,X
        DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::V
        DOUBLE PRECISION,INTENT(IN)::h

        INTEGER::i
200

        DO i=1,N-1
            V(i)=-(2.0D0-(((N-i)**2.0D0)/N**2.0D0))*(U(i)-U(i-1))/(x(i)-x(i-1))
        END DO

        END SUBROUTINE VEC

208     DOUBLE PRECISION FUNCTION G(x,t)
        IMPLICIT NONE
        DOUBLE PRECISION,INTENT(IN)::x,t
        DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.2D0,D=0.8D0*Ir,KK=10.0D0
        DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
          sigma**2.0D0)
        DOUBLE PRECISION::Tau

        Tau=(0.5D0*(Sigma**2.0D0))*(1.0D0-t)
216     G=(EXP(K1*Tau))*(((K2-K1)*EXP(x-(K2-1.0D0)*Tau))+K1)

        END FUNCTION G

        SUBROUTINE TRIDIAG_SOL(N,RHS,U,t,k,NEGINFINITY,r1,r2,r3,x)
        IMPLICIT NONE
        DOUBLE PRECISION,INTENT(IN)::NEGINFINITY,k,t
        INTEGER,INTENT(IN)::N
224     DOUBLE PRECISION,DIMENSION(0:N)::Alpha,s,y
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(OUT)::U
        DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::RHS,r1,r2,r3
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::x
        DOUBLE PRECISION::a,b,c,tau,r
```

```
      INTEGER::i

      DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.2D0,D=0.8D0*Ir,KK=10.0D0
232   DOUBLE PRECISION,PARAMETER::K1=(2.0D0*Ir)/(sigma**2.0D0),K2=(2.0D0*(Ir-D
     ))/(sigma**2.0D0)

      Alpha=0.0D0
      s=0.0D0
      y=0.0D0


      a=2.0*r1(2)
240   b=(2.0D0-2.0*r2(2))
      c=2.0*r3(2)
      Alpha(1)=b
      S(1)=RHS(1)


      tau=((sigma**2.0D0)/2.0D0)*(1.0D0-t)

248

      DO i=2,(N-3)

         a=2.0*r1(i)
         b=(2.0D0-2.0*r2(i))
         c=2.0D0*r3(i)
         Alpha(i)=b-(a*c/Alpha(i-1))
         S(i)=RHS(i)+(a*S(i-1)/Alpha(i-1))
256
      END DO

      y(N-3)=S(N-3)/Alpha(N-3)
      y(N-2)=(4.0D0/9.0D0)*y(N-3)
      y(N-1)=(1.0D0/4.0D0)*y(N-2)
      y(N)=0.0D0

264

      DO i=(N-4),3,-1
         y(i)=(s(i)+c*y(i+1))/Alpha(i)
```

```
      END DO

      DO i=0,N
         U(i)=y(i)
272      WRITE(UNIT=12,FMT='(E12.6)')U(i)
         Print*,U(i)
      END DO


      END SUBROUTINE TRIDIAG_SOL

      SUBROUTINE TRIDIAG_SOL1(N,RHS,Z,k,r1,r2,r3)
280   IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,DIMENSION(1:N-1)::Alpha,S,Q
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::Z
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::RHS,r1,r2,r3
      DOUBLE PRECISION,INTENT(IN)::k
      DOUBLE PRECISION::a,b,c,r
      INTEGER::i
288


      s=0.0D0


      a=2.0*r1(1)
      b=(2.0D0-2.0*r2(1))
      c=2.0D0*r3(1)
296
      Alpha(1)=b
      S(1)=RHS(1)


      DO i=2,(N-1)

         a=2.0*r1(i)
304      b=(2.0D0-2.0*r2(i))
         c=2.0*r3(i)
```

```
        Alpha(i)=b-(a*c/Alpha(i-1))
        S(i)=RHS(i)+(a*S(i-1)/Alpha(i-1))

      END DO

312   Q(N-1)=S(N-1)/Alpha(N-1)

      DO i=(N-2),1,-1

        Q(i)=(S(i)+c*Q(i+1))/Alpha(i)

      END DO

320   DO i=1,N-1

        Z(i)=Q(i)

      END DO


      END SUBROUTINE TRIDIAG_SOL1

328

      SUBROUTINE TRIDIAG_SOL2(N,V,Y,k,h,r1,r2,r3)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,INTENT(IN)::k,h
      DOUBLE PRECISION,DIMENSION(1:N-1)::Alpha,s,D
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::Y
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::V,r1,r2,r3
336   DOUBLE PRECISION::a,b,c,r
      INTEGER::i



        s=0.0
        Y=0.0

344     a=2.0*r1(1)
```

```
      b=2.0-2.0*r2(1)
      c=2.0*r3(1)

      Alpha(1)=b
      S(1)=V(1)

      DO i=2,N-1
352
        a=2.0*r1(i)
        b=2.0D0-2.0*r2(i)
        c=2.0*r3(i)

        Alpha(i)=b-(a*c/Alpha(i-1))
        S(i)=V(i)+(a*S(i-1)/Alpha(i-1))

360   END DO

      D(N-1)=s(N-1)/Alpha(N-1)

      DO i=(N-2),1,-1

        D(i)=(S(i)+c*D(i+1))/Alpha(i)

368   END DO

      DO i=1,N-1

        Y(i)=D(i)

      END DO

376

      END SUBROUTINE TRIDIAG_SOL2

      SUBROUTINE XN_SOLVE(N,Y,Z,HT,XN)
      IMPLICIT NONE
      INTEGER,INTENT(IN)::N
      DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::Y,Z,HT
      DOUBLE PRECISION,INTENT(OUT)::XN
```

```
384     DOUBLE PRECISION::A,B
        INTEGER::i

        A=0.0D0
        B=0.0D0
        DO i=1,N-1

            A=A+Ht(i)*Z(i)
392         B=B+Ht(i)*Y(i)

        END DO

        XN=A/B

    END SUBROUTINE XN_SOLVE
```

.3 Program 3 - pt3.f90

```
    PROGRAM FINITE_ELEMENTS_V5n
            DOUBLE PRECISION::k,NEGINFINITY,Delt,Tfinal,t,h,SUMG1,SUMG2,Theta1,
              ThetaNew,tau,SumC,Domain,GINTERGRAL,DERIV,SUMG
        DOUBLE PRECISION,ALLOCATABLE::U(:),Z(:),Y(:),X(:),G(:),KM(:,:),KP(:,:),
          Phi(:),h_Vector(:),dPdX(:),Cvec(:),ThetaV(:),f(:),&
        W(:),HT(:),RHS(:),M(:,:),RHS1(:)
        INTEGER::IOS,j,N,L,i,q
        DOUBLE PRECISION,EXTERNAL::Uprime
        DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.2D0,D=0.8D0*Ir,KK=10.0D0
 8      DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
          sigma**2.0D0)

        PRINT*,'*******************************************'
        PRINT*,'*                                         *'
        PRINT*,'*            FINITE ELEMENT SOLULTION     *'
        PRINT*,'*                                         *'
        PRINT*,'*              DIFFUSION EQUATION         *'
        PRINT*,'*                                         *'
16      PRINT*,'*                                         *'
        PRINT*,'*******************************************'
```

```
      PRINT*

        NEGINFINITY=-0.05D0
        Tfinal=(Sigma**2.0)/2.0D0

        Domain=NEGINFINITY
        N=100
        L=NINT((Tfinal/(Domain/N)**2))



        k=(Tfinal/L)

        Delt=(Tfinal)/L



        t=0.0D0
        GINTERGRAL=0.0
        Theta1=0.0D0
        B=0.0D0
        f=0.0D0
        RHS=0.0D0

      ALLOCATE(U(0:N+1),G(0:N),Z(1:N-1),Y(1:N),X(0:N+1),KM(1:N,0:N+1),KP(1:N
         -1,0:N),&
     Phi(0:N),h_vector(0:N),dPdX(0:N),Cvec(1:N-1)&
     ,ThetaV(0:N),f(1:N-1),W(0:N),HT(1:N),RHS(1:N-1),M(1:N-1,0:N),RHS1(1:N
         -1))

        SumC=0.0D0
        U=0.0D0
        Y=0.0D0
        Z=0.0D0
        Y=0.0D0
        KM=0.0D0
        KP=0.0D0
        t=0.0D0
        f=0.0D0
```

```
      M=0.0D0
56
      OPEN(UNIT=11,FILE="x.dat",IOSTAT=IOS)
      OPEN(UNIT=12,FILE="U.dat",IOSTAT=IOS)
      OPEN(UNIT=13,FILE="K.dat",IOSTAT=IOS)
      OPEN(UNIT=14,FILE="KP.dat",IOSTAT=IOS)
      OPEN(UNIT=15,FILE="G.dat",IOSTAT=IOS)
      OPEN(UNIT=16,FILE="Phi.dat",IOSTAT=IOS)
64    OPEN(UNIT=17,FILE="M.dat",IOSTAT=IOS)
      OPEN(UNIT=18,FILE="C.dat",IOSTAT=IOS)
      OPEN(UNIT=19,FILE="SumTheta.dat",IOSTAT=IOS)
      OPEN(UNIT=20,FILE="Y.dat",IOSTAT=IOS)
      OPEN(UNIT=21,FILE="dPdX.dat",IOSTAT=IOS)
      OPEN(UNIT=22,FILE="NewTheta.dat",IOSTAT=IOS)
      OPEN(UNIT=23,FILE="Z.dat",IOSTAT=IOS)
      OPEN(UNIT=24,FILE="W.dat",IOSTAT=IOS)
72    OPEN(UNIT=25,FILE="G1.dat",IOSTAT=IOS)
      OPEN(UNIT=26,FILE="G2.dat",IOSTAT=IOS)
      OPEN(UNIT=27,FILE="Y1.dat",IOSTAT=IOS)
      OPEN(UNIT=28,FILE="kk.dat",IOSTAT=IOS)
      OPEN(UNIT=29,FILE="Stheta.dat",IOSTAT=IOS)
      OPEN(UNIT=30,FILE="Yold.dat",IOSTAT=IOS)
      OPEN(UNIT=31,FILE="MassMul.dat",IOSTAT=IOS)
      IF(IOS/=0) THEN
80        PRINT*,'Error Occured in Opening The Output File'
          STOP
      END IF

      !*****************************************************************
      !*                                                               *
      !*                          MAIN PROGRAM                         *
      !*                                                               *
88    !*****************************************************************

      WRITE(UNIT=28,FMT='(E12.6)')k

      DO i=0,N
```

```
      X(i)=NEGINFINITY+(REAL(i)/REAL(N))*(B-NEGINFINITY)

96    WRITE(UNIT=11,FMT='(E12.6)')X(i)

      END DO



      CALL INITIAL_DATA(N,U,X)

104

      HT(1)=1.0D0
      HT(2)=-1.0D0

      DO i=3,N

        HT(i)=0.0D0
112
      END DO


      t=1.0D0

       DO i=0,N-1

120       CALL Theta(X,ThetaV,N,i,t,U)

          Theta1=Theta1+ThetaV(i)

          WRITE(UNIT=19,FMT='(E12.6)')Theta1

        END DO
          WRITE(UNIT=29,FMT='(E12.6)')Theta1
128
      DO q=1,1

        GINTEGRAL=0.0
```

```
      Tau=((sigma**2.0D0)/2.0D0)*(1.0D0)*(REAL(q)/REAL(L))


136   DO i=1,N-1

        CALL Cvector(X,Cvec,N,i,t,U,Theta1)

        WRITE(UNIT=18,FMT='(E12.6)')Cvec(i)

        SumC=SumC+Cvec(i)

144   END DO


      DO i=1,N-1

        CALL SIMPSONS1(X,SUMG1,N,i,q,L)

        CALL SIMPSONS2(X,SUMG2,N,i,q,L)

152

            G(i)=SUMG1+SUMG2

      WRITE(UNIT=15,FMT='(E12.6)')G(i)

      END DO

      G(N)=G(N-1)

160

      DO i=1,N-1

        CALL MASS_MATRIX(i,N,X,KM)

        CALL MASS_MATRIX_PRIME(i,N,X,KP,U)

      END DO
168   KP(1,1)=KP(1,1)/2.0D0


      RHS1(1)=U(1)*KM(1,1)+U(2)*KM(1,2)
```

```
      DO j=2,N-2

176                RHS1(j)=U(j-1)*KM(j,j-1)+U(j)*KM(j,j)+U(j+1)*KM(j,j+1)
          WRITE(UNIT=31,FMT='(E14.8)')RHS1(j)

      END DO

      RHS1(N-1)=U(N-2)*KM(N-1,N-2)+U(N-1)*KM(N-1,N-1)

      DO j=1,N-1
184
        Y(j)=G(j)-RHS1(j)

        WRITE(UNIT=27,FMT='(E14.8)')Y(j)
      END DO



192   DO j=1,N-1

        f(j)=Y(j)

      END DO



200   DO i=1,N

        h_vector(i)=X(i)-X(i-1)

      END DO



208   CALL TRIDIAG_SOL1(N,f,Z,k,KP)

      CALL TRIDIAG_SOL2(N,Cvec,W,k,KP)
```

```
      CALL THETA_SOLVE(N,W,Z,HT,ThetaNew)
      WRITE(UNIT=22,FMT='(E14.8)')ThetaNew

216   DO j=1,N-1

        WRITE(UNIT=30,FMT='(E14.8)')Y(j)

      END DO

      Do i=1,N-1

224

      CALL IntegrateG(X,SUMG,N,i,q,L)

      GINTERGRAL=GINTERGRAL+SUMG

      END DO

      DERIV=-UPrime(NEGINFINITY,Tau)+GINTERGRAL
232


      PRINT*,THETANEW,'New Theta'


      DO j=1,N-1

240     Y(j)=Y(j)-(THETANEW)*Cvec(j)

        WRITE(UNIT=20,FMT='(E14.8)')Y(j)


      END DO

      CALL TRIDIAG_SOL(N,Y,KP,Phi)

248

       DO i=1,N
```

```
      IF (i/=N) THEN

        dPdX(i)=(((X(i)-X(i-1))*(Phi(i)-Phi(i-1))+(X(i+1)-X(i))*(Phi(
          i+1)-&
        Phi(i))))/((X(i)-X(i-1))+(X(i+1)-X(i)))

256

      ELSE

        dPdX(i)=(Phi(i)-Phi(i-1))/(X(i)-X(i-1))
      END IF

    END DO
264       dPdX(0)=0.0D0

    DO i=0,N

      CALL EULERS(N,X,h_vector,k,dPdX)

    END DO

272 DO i=0,N


    END DO

    DO i=1,N-1

      CALL M_MATRIX(i,N,X,M)
280
    END DO



    DO i=1,N-1

        RHS(i)=ThetaNew*Cvec(i)
```

```
288
            END DO

            CALL TRIDIAG_SOL3(N,M,U,RHS,tau,NEGINFINITY)

            DO i=0,N

                  !PRINT*,i,U(i)
296             !WRITE(UNIT=12,FMT='(E12.6) ')U(i)
            END DO
            !Theta1=ThetaNew
        END DO

        WRITE(UNIT=13,FMT='(101F14.8) ')(((KM(j,i)),i=0,N),j=1,N-1)
        WRITE(UNIT=14,FMT='(101F14.8) ')(((KP(j,i)),i=0,N),j=1,N-1)
        WRITE(UNIT=21,FMT='(E12.6) ')((dPdX(i)),i=0,N)
304     WRITE(UNIT=17,FMT='(101F14.8) ')(((M(j,i)),i=0,N),j=1,N-1)

        CLOSE(UNIT=11)
        CLOSE(UNIT=12)
        CLOSE(UNIT=13)
        CLOSE(UNIT=14)
        CLOSE(UNIT=15)
        CLOSE(UNIT=16)
312     CLOSE(UNIT=17)
        CLOSE(UNIT=18)
        CLOSE(UNIT=19)
        CLOSE(UNIT=20)
        CLOSE(UNIT=21)
        CLOSE(UNIT=22)
        CLOSE(UNIT=23)
        CLOSE(UNIT=24)
320     CLOSE(UNIT=25)
        CLOSE(UNIT=26)
        CLOSE(UNIT=27)
        CLOSE(UNIT=28)
        CLOSE(UNIT=29)
        CLOSE(UNIT=30)
        CLOSE(UNIT=31)
```

```
328       END PROGRAM FINITE_ELEMENTS_V5n

    SUBROUTINE INITIAL_DATA(N,U,X)
        IMPLICIT NONE
        INTEGER,INTENT(IN)::N
        DOUBLE PRECISION,DIMENSION(0:N+1),INTENT(IN)::X
        DOUBLE PRECISION,DIMENSION(0:N+1),INTENT(OUT)::U
        INTEGER::i
336
        DO i=0,N

            U(i)=max(1.0D0-EXP(X(I)),0.0D0)

            WRITE(UNIT=12,FMT='(E12.6)')U(i)

        END DO
344
        END SUBROUTINE INITIAL_DATA

    SUBROUTINE IntegrateG(X,SUMG,N,i,q,L)
            IMPLICIT NONE
            INTEGER,INTENT(IN)::N,i,q,L
            DOUBLE PRECISION,DIMENSION(0:N+1),INTENT(IN)::X
            DOUBLE PRECISION,INTENT(OUT)::SUMG
352         DOUBLE PRECISION::h,XI0,XI2,XI1,a,b,XI,SUM,Tau
            DOUBLE PRECISION,EXTERNAL::GF
            DOUBLE PRECISION,PARAMETER::Sigma=0.4D0
            INTEGER::k,M

            Tau=((Sigma**2.0D0)/2.0D0)*(0.5D0)*(REAL(q)/REAL(L))

            SUMG=0.0
360
            M=INT(N*200)
            a=X(i-1)
            b=X(i)

            h=ABS((X(i)-X(i-1)))/M
```

```
            XI0=GF(b,Tau)+GF(a,Tau)
368         XI1=0.0D0
            XI2=0.0D0

            DO k=1,M-1

                XI=a+k*h

                IF (MOD(k,int(2))/=0) THEN
376
                    XI2=XI2+GF(XI,Tau)
                ELSE
                    XI1=XI1+GF(XI,Tau)

                END IF
            END DO

384         SUMG=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0

        END SUBROUTINE IntegrateG



    SUBROUTINE SIMPSONS1(X,SUMG1,N,i,q,L)
392         IMPLICIT NONE
            INTEGER,INTENT(IN)::N,i,q,L
            DOUBLE PRECISION,DIMENSION(0:N+1),INTENT(IN)::X
            DOUBLE PRECISION,INTENT(OUT)::SUMG1
            DOUBLE PRECISION::h,XI0,XI2,XI1,a,b,XI,SUM,Tau
            DOUBLE PRECISION,EXTERNAL::GF
            DOUBLE PRECISION,PARAMETER::Sigma=0.4D0
            INTEGER::k,M
400
            Tau=((Sigma**2.0D0)/2.0D0)*(0.5D0)*(REAL(q)/REAL(L))

            M=N*200
            a=X(i-1)
```

```
        b=X(i)

        h=ABS((X(i)-X(i-1)))/M
408
        XI0=(b-X(i-1))*GF(b,Tau)+(a-X(i-1))*GF(a,Tau)
        XI1=0.0D0
        XI2=0.0D0

        DO k=1,M-1

            XI=a+k*h
416
            IF (MOD(k,int(2))/=0) THEN

                XI2=XI2+(XI-X(i-1))*GF(XI,Tau)
            ELSE
                XI1=XI1+(XI-X(i-1))*GF(XI,Tau)

            END IF
424     END DO

        SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
        SUMG1=SUM*(1.0D0/(X(i)-X(i-1)))



        WRITE(UNIT=25,FMT='(E12.6)')SUMG1

432 END SUBROUTINE SIMPSONS1

    SUBROUTINE SIMPSONS2(X,SUMG2,N,i,q,L)
        IMPLICIT NONE
        INTEGER,INTENT(IN)::N,i,q,L
        DOUBLE PRECISION,DIMENSION(0:N+1),INTENT(IN)::X
        DOUBLE PRECISION,INTENT(OUT)::SUMG2
        DOUBLE PRECISION::h,XI0,XI2,XI1,a,XI,b,SUM,Tau
440     DOUBLE PRECISION,PARAMETER::Sigma=0.4
        DOUBLE PRECISION,EXTERNAL::GF
        INTEGER::k,M
```

```
      Tau=((Sigma**2.0D0)/2.0D0)*(0.5D0)*(REAL(q)/REAL(L))

      a=X(i)
      b=X(i+1)
448   M=N*200
      h=ABS((X(i+1)-X(i)))/M

      XI0=(X(i+1)-a)*GF(a,Tau)+(X(i+1)-b)*GF(b,Tau)
      XI1=0.0D0
      XI2=0.0D0

      DO k=1,M-1
456
         XI=a+k*h

         IF (MOD(k,int(2))/=0) THEN

            XI2=XI2+(X(i+1)-XI)*GF(XI,Tau)
         ELSE
            XI1=XI1+(X(i+1)-XI)*GF(XI,Tau)
464
         END IF
      END DO

      SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
      SUMG2=SUM*(1.0D0/(X(i+1)-X(i)))

      WRITE(UNIT=26,FMT='(E12.6)')SUMG2
472

    END SUBROUTINE SIMPSONS2




    DOUBLE PRECISION FUNCTION GF(x,tau)
480 IMPLICIT NONE
    DOUBLE PRECISION,INTENT(IN)::x,tau
    DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.2D0,D=0.8D0*Ir,KK=10.0D0
```

```
      DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
         sigma**2.0D0)




488         GF=(EXP(K1*Tau))*(((K2-K1)*EXP(x-(K2-1.0D0)*Tau))+K1)

      END FUNCTION GF



      DOUBLE PRECISION FUNCTION UPrime(x,Tau)
      IMPLICIT NONE
496   DOUBLE PRECISION,INTENT(IN)::x,Tau
      DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.2D0,D=0.8D0*Ir,KK=10.0D0
      DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
         sigma**2.0D0)




         UPrime=-exp(K1*Tau)*(exp(x-(K2-1)*Tau))
504
      END FUNCTION Uprime



      SUBROUTINE MASS_MATRIX(i,N,X,KM)
          IMPLICIT NONE
512       INTEGER,INTENT(IN)::N,i
          DOUBLE PRECISION,DIMENSION(0:N+1),INTENT(IN)::X
          DOUBLE PRECISION,DIMENSION(1:N,0:N+1),INTENT(INOUT)::KM
          DOUBLE PRECISION::SUM1,SUM2,SUM3,SUM4,SUM
          DOUBLE PRECISION,EXTERNAL::func
          DOUBLE PRECISION::a,b,h,XI,XI1,XI2,XI0
          INTEGER::k,M
```

```
520         a=X(i-1)
            b=X(i)
           M=INT(N*200)
           h=ABS((X(i)-X(i-1)))/M

              XI0=func(a)+func(b)

              XI1=0.0D0
528           XI2=0.0D0
              XI=0.0D0

           DO k=1,M-1

                 XI=a+k*h

                 IF (MOD(k,int(2))/=0) THEN
536
                          XI2=XI2+func(XI)
                 ELSE
                       XI1=XI1+func(XI)

                 END IF
           END DO

544        SUM=h*((XI0+2.0D0*XI2+4.0*XI1)/3.0D0)

           SUM1=((1.0D0/(X(i)-X(i-1))))


           a=X(i)
           b=X(i+1)

552        h=ABS((X(i+1)-X(i)))/M

           XI0=func(a)+func(b)
           XI1=0.0D0
           XI2=0.0D0
           XI=0.0
```

```
      DO k=1,M-1
560
        XI=a+k*h

        IF (MOD(k,int(2))/=0) THEN

          XI2=XI2+func(XI)
        ELSE
          XI1=XI1+func(XI)
568
        END IF

      END DO

      SUM=(h)*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
      SUM2=(1.0D0/(X(i+1)-X(i)))

576   KM(i,i)=SUM1+SUM2

      a=X(i)
      b=X(i+1)

      h=ABS((X(i+1)-X(i)))/M

      XI0=func(a)+func(b)
584   XI1=0.0D0
      XI2=0.0D0
      XI=0.0

      DO k=1,M-1

        XI=a+k*h

592     IF (MOD(k,int(2))/=0) THEN

          XI2=XI2+func(XI)
        ELSE
          XI1=XI1+func(XI)
```

```
            END IF

600         END DO

            SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
            SUM3=-1.0*((1.0D0/(X(i+1)-X(i))**2))*SUM
            KM(i,i+1)=-1.0D0*(1.0D0/(X(i+1)-X(i)))

            a=X(i-1)
            b=X(i)
608
            h=ABS((X(i)-X(i-1)))/M

            XI0=func(a)+func(b)
            XI1=0.0D0
            XI2=0.0D0
            XI=0.0D0

616         DO k=1,M-1

                XI=a+k*h

                IF (MOD(k,int(2))/=0) THEN

                        XI2=XI2+func(XI)
                ELSE
624                     XI1=XI1+func(XI)

                END IF

            END DO

            SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
            SUM4=-1.0D0*((1.0D0/(X(i)-X(i-1)))**2)*SUM
632         KM(i,i-1)=-1.0D0*(1.0D0/(X(i)-X(i-1)))


        END SUBROUTINE MASS_MATRIX
```

```
      DOUBLE PRECISION FUNCTION func(x)
        IMPLICIT NONE
640     DOUBLE PRECISION,INTENT(IN)::x

          func=1.0D0

      END FUNCTION func

      DOUBLE PRECISION FUNCTION UINTER1(UN,UM,UP,Xin,i,a,b)
          IMPLICIT NONE
648       INTEGER,INTENT(IN)::i
          DOUBLE PRECISION,INTENT(IN)::UN,UM,UP
          DOUBLE PRECISION,INTENT(IN)::a,b
          DOUBLE PRECISION,INTENT(IN)::Xin

                UINTER1=UM*(Xin-a)/(b-a)+UN*((b-Xin)/(b-a))

      END FUNCTION UINTER1
656
       DOUBLE PRECISION FUNCTION UINTER2(UN,UM,UP,Xin,i,a,b)
          IMPLICIT NONE
          INTEGER,INTENT(IN)::i
          DOUBLE PRECISION,INTENT(IN)::UN,UM,UP
          DOUBLE PRECISION,INTENT(IN)::a,b
          DOUBLE PRECISION,INTENT(IN)::Xin

664             UINTER2=UM*(b-Xin)/(b-a)+UP*(Xin-a)/(b-a)

      END FUNCTION UINTER2

          DOUBLE PRECISION FUNCTION UINTER(UP,UM,Xin,i,a,b)
          IMPLICIT NONE
          INTEGER,INTENT(IN)::i
          DOUBLE PRECISION,INTENT(IN)::UP,UM
672       DOUBLE PRECISION,INTENT(IN)::a,b
          DOUBLE PRECISION,INTENT(IN)::Xin

                UINTER=UM*(b-Xin)/(b-a)+UP*((Xin-a)/(b-a))
```

```
      END FUNCTION UINTER

      SUBROUTINE MASS_MATRIX_PRIME(i,N,X,KP,U)
680       IMPLICIT NONE
          INTEGER,INTENT(IN)::N,i
          DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::X
          DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::U
          DOUBLE PRECISION,DIMENSION(1:N-1,0:N),INTENT(INOUT)::KP
          DOUBLE PRECISION::SUM1,SUM2,SUM3,SUM4,e,f,g

          DOUBLE PRECISION::a,b,h,XI,XI1,XI2,XI0,UN,UM,UP,TEST,SUM
688       DOUBLE PRECISION,EXTERNAL::UINTER1,UINTER2
          INTEGER::k,M

          UN=U(i-1)
          UM=U(i)
          UP=U(i+1)

          a=X(i-1)
696       b=X(i)
          M=N*200
          h=ABS((X(i)-X(i-1)))/M

            XI0=UINTER1(UN,UM,UP,a,i,a,b)+UINTER1(UN,UM,UP,b,i,a,b)
            XI1=0.0D0
            XI2=0.0D0
            XI=0.0
704
          DO k=1,M-1

              XI=a+k*h

              IF (MOD(k,int(2))/=0) THEN

                    XI2=XI2+UINTER1(UN,UM,UP,XI,i,a,b)
712           ELSE
                  XI1=XI1+UINTER1(UN,UM,UP,XI,i,a,b)
```

```
            END IF
        END DO

        SUM=(h)*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
        SUM1=SUM*((1.0D0/(X(i)-X(i-1)))**2.0D0)
720
        b=X(i+1)
        a=X(i)

        h=(X(i+1)-X(i))/M
        XI0=UINTER2(UN,UM,UP,a,i,a,b)+UINTER2(UN,UM,UP,b,i,a,b)
        XI1=0.0D0
        XI2=0.0D0
728     XI=0.0D0

        DO k=1,M-1

            XI=a+k*h

            IF (MOD(k,int(2))/=0) THEN

736                 XI2=XI2+UINTER2(UN,UM,UP,XI,i,a,b)
            ELSE
                XI1=XI1+UINTER2(UN,UM,UP,XI,i,a,b)

            END IF

        END DO

744     SUM=(h)*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
        SUM2=SUM*((1.0D0/(X(i+1)-X(i)))**2.0D0)

        KP(i,i)=SUM1+SUM2
        f=KP(i,i)

        b=X(i+1)
        a=X(i)
752
        h=ABS((X(i+1)-X(i)))/M
```

```
      XI0=UINTER2(UN,UM,UP,a,i,a,b)+UINTER2(UN,UM,UP,b,i,a,b)
      XI1=0.0D0
      XI2=0.0D0
      XI=0.0D0

760   DO k=1,M-1

        XI=a+k*h

        IF (MOD(k,int(2))/=0) THEN

          XI2=XI2+UINTER2(UN,UM,UP,XI,i,a,b)
        ELSE
768       XI1=XI1+UINTER2(UN,UM,UP,XI,i,a,b)

        END IF

      END DO

      SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
      SUM3=SUM*(-1.0D0)*((1.0D0/(X(i+1)-X(i)))**2.0D0)
776   KP(i,i+1)=SUM3
      g=KP(i,i+1)

      b=X(i)
      a=X(i-1)

      h=ABS((X(i)-X(i-1)))/M

784   XI0=UINTER1(UN,UM,UP,a,i,a,b)+UINTER1(UN,UM,UP,b,i,a,b)
      XI1=0.0D0
      XI2=0.0D0
      XI=0.0D0

      DO k=1,M-1

        XI=a+k*h

792
```

```
                IF (MOD(k,int(2))/=0) THEN

                        XI2=XI2+UINTER1(UN,UM,UP,XI,i,a,b)
                    TEST=UINTER1(UN,UM,UP,XI,i,a,b)

                ELSE
                    XI1=XI1+UINTER1(UN,UM,UP,XI,i,a,b)
800
                END IF

            END DO

            SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
            SUM4=SUM*((-1.0D0)*((1.0D0/(X(i)-X(i-1)))**2.0D0))
            KP(i,i-1)=SUM4
808         e=KP(i,i-1)



        END SUBROUTINE MASS_MATRIX_PRIME

        SUBROUTINE TRIDIAG_SOL(N,Y,KP,Phi)
        IMPLICIT NONE
816     INTEGER,INTENT(IN)::N
        DOUBLE PRECISION,DIMENSION(1:N)::Alpha,S,C
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(INOUT)::Phi
        DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::Y
        DOUBLE PRECISION,DIMENSION(0:N)::Z
        DOUBLE PRECISION,DIMENSION(1:N-1,0:N),INTENT(IN)::KP
        DOUBLE PRECISION::a,b
        INTEGER::i
824


        S=0.0D0
        a=-KP(1,0)
        b=KP(1,1)
        c(1)=-KP(1,2)

        Alpha(1)=b
```

```
832       S(1)=Y(1)


          DO i=2,(N-1)

             a=-KP(i,i-1)
             b=KP(i,i)
             c(i)=-KP(i,i+1)
840

             Alpha(i)=b-(a*c(i-1)/Alpha(i-1))
             S(i)=Y(i)+(a*S(i-1)/Alpha(i-1))

          END DO

          Z(N-1)=S(N-1)/Alpha(N-1)

848       DO i=(N-2),1,-1

             Z(i)=(S(i)+c(i)*Z(i+1))/Alpha(i)

          END DO
          Z(N)=Z(N-1)
          Z(0)=Z(1)

856       DO i=0,N

             Phi(i)=Z(i)

             WRITE(UNIT=16,FMT='(E12.6)')Phi(i)



864

          END DO


          END SUBROUTINE TRIDIAG_SOL

          SUBROUTINE EULERS(N,X,h_vector,k,dPdX)
```

```
        IMPLICIT NONE
872     INTEGER,INTENT(IN)::N
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(INOUT)::X
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::dPdX
        DOUBLE PRECISION,INTENT(IN)::k
        DOUBLE PRECISION,DIMENSION(1,N-1),INTENT(IN)::h_vector
        INTEGER::i

        DO i=0,N
880
            X(i)=X(i)+k*dPdX(i)

        END DO



    END SUBROUTINE EULERS
888

    SUBROUTINE Cvector(X,Cvec,N,i,t,U,Theta1)
        IMPLICIT NONE
        INTEGER,INTENT(IN)::N,i
        DOUBLE PRECISION,INTENT(IN)::t
        DOUBLE PRECISION,INTENT(IN)::Theta1
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::X
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::U
896     DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(INOUT)::Cvec
        DOUBLE PRECISION,EXTERNAL::UINTER1,UINTER2
        DOUBLE PRECISION::h,XI0,XI2,XI1,a,XI,SUMC1,SUMC2,UN,UM,UP,b,SUM,TEST
        INTEGER::k,M

        M=N*300
        a=X(i-1)
        b=X(i)
904     h=ABS((X(i)-X(i-1)))/M

        UN=U(i-1)
        UM=U(i)
        UP=U(i+1)
```

```
      XI0=(a-X(i-1))*UINTER1(UN,UM,UP,a,i,a,b)+(b-X(i-1))*UINTER1(UN,UM,UP
     ,b,i,a,b)
912   XI1=0.0D0
      XI2=0.0D0

      DO k=1,M-1

         XI=a+k*h

         IF (MOD(k,int(2))/=0) THEN
920
            XI2=XI2+(XI-X(i-1))*UINTER1(UN,UM,UP,XI,i,a,b)

         ELSE
            XI1=XI1+(XI-X(i-1))*UINTER1(UN,UM,UP,XI,i,a,b)

         END IF
      END DO
928
      SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
      SUMC1=(1.0D0/(X(i)-X(i-1)))*SUM




      b=X(i+1)
      a=X(i)
936


      h=ABS((X(i+1)-X(i)))/M


      XI0=(X(i+1)-a)*UINTER2(UN,UM,UP,a,i,a,b)+(X(i+1)-b)*UINTER2(UN,UM,UP
     ,b,i,a,b)
      XI1=0.0D0
      XI2=0.0D0
944
      DO k=1,M-1
```

```
                XI=a+k*h

                IF (MOD(k,int(2))/=0) THEN

                    XI2=XI2+(X(i+1)-XI)*UINTER2(UN,UM,UP,XI,i,a,b)
952             ELSE
                    XI1=XI1+(X(i+1)-XI)*UINTER2(UN,UM,UP,XI,i,a,b)

                    TEST=UINTER2(UN,UM,UP,XI,i,a,b)

                END IF
            END DO

960         SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
            SUMC2=(1.0D0/(X(i+1)-X(i)))*SUM

            Cvec(i)=(SUMC1+SUMC2)/Theta1


        END SUBROUTINE Cvector

968


        SUBROUTINE Theta(X,ThetaV,N,i,t,U)
            IMPLICIT NONE
            INTEGER,INTENT(IN)::N,i
            DOUBLE PRECISION,INTENT(IN)::t
            DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::X
            DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::U
976         DOUBLE PRECISION,DIMENSION(0:N),INTENT(INOUT)::ThetaV
            DOUBLE PRECISION,EXTERNAL::UINTER
            DOUBLE PRECISION::h,XI0,XI2,XI1,a,XI,SUMC1,UP,UM,b,test
            INTEGER::k,M

            M=(N*3000)
            a=X(i)
            b=X(i+1)
984         h=ABS((X(i+1)-X(i)))/M
```

```
      UP=U(i+1)
      UM=U(i)



      XI0=UINTER(UP,UM,a,i,a,b)+UINTER(UP,UM,b,i,a,b)
992   XI1=0.0D0
      XI2=0.0D0

      DO k=1,M-1

         XI=a+k*h

         IF (MOD(k,int(2))/=0) THEN
1000
            XI2=XI2+UINTER(UP,UM,XI,i,a,b)

         ELSE
            XI1=XI1+UINTER(UP,UM,XI,i,a,b)

         END IF
      END DO
1008
      SUMC1=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0

      ThetaV(i)=SUMC1

    END SUBROUTINE Theta

    SUBROUTINE TRIDIAG_SOL1(N,f,Z,k,KP)
1016  IMPLICIT NONE
    INTEGER,INTENT(IN)::N
    DOUBLE PRECISION,DIMENSION(1:N-1)::Alpha,S,y,C
    DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(OUT)::Z
    DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::f
    DOUBLE PRECISION,DIMENSION(1:N-1,0:N),INTENT(IN)::KP
    DOUBLE PRECISION,INTENT(IN)::k
    DOUBLE PRECISION::a,b,r
1024  INTEGER::i
```

```
      s=0.0D0
      y=0.0D0



1032  a=-KP(1,0)
      b=KP(1,1)
      c(1)=-KP(1,2)

      Alpha(1)=b
      S(1)=f(1)


1040  DO i=2,(N-1)

        a=-KP(i,i-1)
        b=KP(i,i)
        c(i)=-KP(i,i+1)


        Alpha(i)=b-(a*c(i-1)/Alpha(i-1))
1048    S(i)=f(i)+(a*S(i-1)/Alpha(i-1))

      END DO

      y(N-1)=S(N-1)/Alpha(N-1)

      DO i=(N-2),1,-1

1056    y(i)=(S(i)+c(i)*y(i+1))/Alpha(i)

      END DO

      DO i=1,N-1

        Z(i)=y(i)
        WRITE(UNIT=23,FMT='(E12.6)')Z(i)
```

```
1064        END DO


            END SUBROUTINE TRIDIAG_SOL1

            SUBROUTINE TRIDIAG_SOL2(N, Cvec ,W, k ,KP)
            IMPLICIT NONE
            INTEGER,INTENT(IN) : : N
1072        DOUBLE PRECISION,INTENT(IN) : : k
            DOUBLE PRECISION,DIMENSION(1:N-1) : : Alpha , s ,U,C
            DOUBLE PRECISION,DIMENSION(1:N-1) ,INTENT(OUT) : :W
            DOUBLE PRECISION,DIMENSION(1:N-1,0:N) ,INTENT(IN) : : KP
            DOUBLE PRECISION,DIMENSION(1:N-1) ,INTENT(IN) : : Cvec
            DOUBLE PRECISION : : a , b , r
            INTEGER : : i

1080

            s=0.0D0
            W=0.0D0
            U=0.0D0



1088        a=-KP(1 ,0)
            b=KP(1 ,1)
            c(1)=-KP(1 ,2)

            Alpha(1)=b
            S(1)=(Cvec(1)/k)

            DO i=2,N-1
1096

                a=-KP(i , i-1)
                b=KP(i , i)
                c(i)=-KP(i , i+1)

                Alpha(i)=b-(a*c(i-1)/Alpha(i-1))
                S(i)=(Cvec(i)/k)+(a*S(i-1)/Alpha(i-1))
```

```
1104        END DO

            U(N-1)=s(N-1)/Alpha(N-1)

            DO i=(N-2),1,-1

              U(i)=(S(i)+C(i)*U(i+1))/Alpha(i)

1112        END DO

            DO i=1,N-1

              W(i)=U(i)
              WRITE(UNIT=24,FMT='(E12.6)')W(i)
            END DO

1120

            END SUBROUTINE TRIDIAG_SOL2

            SUBROUTINE THETA_SOLVE(N,W,Z,HT,ThetaNew)
            IMPLICIT NONE
            INTEGER,INTENT(IN)::N
            DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::W,Z,HT
            DOUBLE PRECISION,INTENT(OUT)::ThetaNew
1128        DOUBLE PRECISION::A,B
            INTEGER::i

            A=0.0D0
            B=0.0D0
            DO i=1,N-1

              A=A+Ht(i)*Z(i)
1136          B=B+Ht(i)*W(i)

            END DO

            ThetaNew=A/B
```

```
    END SUBROUTINE THETA_SOLVE

1144 SUBROUTINE TRIDIAG_SOL3(N,M,U,RHS,tau,NEGINFINITY)
        IMPLICIT NONE
        INTEGER,INTENT(IN)::N
        DOUBLE PRECISION,DIMENSION(1:N)::Alpha,S,C
        DOUBLE PRECISION,DIMENSION(0:N),INTENT(INOUT)::U
        DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::RHS
        DOUBLE PRECISION,DIMENSION(1:N-1)::Z
        DOUBLE PRECISION,INTENT(IN)::tau,NEGINFINITY
1152    DOUBLE PRECISION,DIMENSION(1:N-1,0:N),INTENT(IN)::M
        DOUBLE PRECISION,PARAMETER::Ir=0.03D0,Sigma=0.2D0,D=0.8D0*Ir,KK=10.0D0
        DOUBLE PRECISION,PARAMETER::K1=2.0D0*Ir/(Sigma**2.0D0),K2=2.0D0*(Ir-D)/(
           sigma**2.0D0)

        DOUBLE PRECISION::a,b
        INTEGER::i


1160    S=0.0D0
        a=M(1,0)
        b=M(1,1)
        c(1)=M(1,2)
        Alpha=0.0D0

        Alpha(1)=b
        S(1)=RHS(1)
1168

        DO i=2,(N-1)

           a=M(i,i-1)
           b=M(i,i)
           c(i)=M(i,i+1)

1176       Alpha(i)=b-(a*c(i-1)/Alpha(i-1))
           S(i)=RHS(i)+(a*S(i-1)/Alpha(i-1))

        END DO
```

```
Z(N-1)=S(N-1)/Alpha(N-1)

DO i=(N-2),1,-1
1184

    Z(i)=(S(i)+c(i)*Z(i+1))/Alpha(i)

END DO

DO i=1,N-1

   U(i)=Z(i)
1192



END DO
U(0)=exp(K1*tau)*(1.0D0-exp(NEGINFINITY-(K2-1.0D0)*tau))
U(N)=0.0

END SUBROUTINE TRIDIAG_SOL3
1200

SUBROUTINE M_MATRIX(i,N,X,M)
    IMPLICIT NONE
    INTEGER,INTENT(IN)::N,i
    DOUBLE PRECISION,DIMENSION(0:N),INTENT(IN)::X
    DOUBLE PRECISION,DIMENSION(1:N-1,0:N),INTENT(INOUT)::M
    DOUBLE PRECISION::SUM1,SUM2,SUM3,SUM4,SUM
1208    DOUBLE PRECISION,EXTERNAL::func
    DOUBLE PRECISION::a,b,h,XI,XI1,XI2,XI0
    INTEGER::k,MI

    a=X(i-1)
    b=X(i)
    MI=int(N*200)
    h=ABS((X(i)-X(i-1)))/MI
1216

       XI0=(b-a)**2+(a-a)**2
       XI1=0.0D0
```

```
      XI2=0.0D0
      XI=0.0

      DO k=1,MI-1

1224      XI=a+k*h

          IF (MOD(k,int(2))/=0) THEN

              XI2=XI2+((XI-a)**2)
          ELSE
              XI1=XI1+((XI-a)**2)

1232      END IF
      END DO

      SUM=(h)*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
      SUM1=(((b**3-a**3)/3)+b*(a**2)-a*(b**2))/((X(i)-X(i-1))**2)

      a=X(i)
      b=X(i+1)
1240
      h=ABS((X(i+1)-X(i)))/MI

      XI0=(b-b)**2+(b-a)**2
      XI1=0.0D0
      XI2=0.0D0
      XI=0.0

1248  DO k=1,MI-1

          XI=a+k*h

          IF (MOD(k,int(2))/=0) THEN

              XI2=XI2+((b-XI)**2)
          ELSE
1256          XI1=XI1+((b-XI)**2)
```

```
            END IF

        END DO

        SUM=(h)*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
        SUM2=(((b**3-a**3)/3.0D0)+((a**2)*b-a*(b**2)))/((X(i+1)-X(i))**2)
1264

        M(i,i)=SUM1+SUM2


        a=X(i)
        b=X(i+1)

1272    h=ABS((X(i+1)-X(i)))/MI

        XI0=(b-b)*(b-a)+(b-a)*(a-a)
        XI1=0.0D0
        XI2=0.0D0
        XI=0.0

        DO k=1,MI-1
1280
            XI=a+k*h

            IF (MOD(k,int(2))/=0) THEN

                XI2=XI2+((b-XI)*(XI-a))
            ELSE
                XI1=XI1+((b-XI)*(XI-a))
1288
            END IF

        END DO

        SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
        SUM3=SUM*((1.0D0/(X(i+1)-X(i)))**2)
        M(i,i+1)=-1.0D0*(((b**3-a**3)/6.0D0)+((a**2)*b-a*(b**2))/2.0D0)/((X(
           i+1)-X(i))**2)
```

```
1296
                a=X(i-1)
                b=X(i)

                h=ABS((X(i)-X(i-1)))/MI

                XI0=(b-a)*(a-a)+(b-b)*(b-a)
1304            XI1=0.0D0
                XI2=0.0D0
                XI=0.0

                DO k=1,MI-1

                        XI=a+k*h

1312                    IF (MOD(k,int(2))/=0) THEN

                                XI2=XI2+((b-XI)*(XI-a))
                        ELSE
                                XI1=XI1+((b-XI)*(XI-a))

                        END IF

1320            END DO

                SUM=h*(XI0+2.0D0*XI2+4.0*XI1)/3.0D0
                SUM4=((1.0D0/(X(i)-X(i-1)))**2)*SUM
                M(i,i-1)=-1.0D0*(((b**3-a**3)/6.0D0)+((a**2)*b-a*(b**2))/2.0D0)/((X(
                   i+1)-X(i))**2)


        END SUBROUTINE M_MATRIX
```