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 sta 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.

Chapter 1

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 Tnecatre is n th

Figure 1.1: Payo 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

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

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 e□ects. It is represented by a random sample drawn from a normal distribution with mean zero. It adds contrisu strrddp212447(a)0.2-0.14703

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

$$dX = \varphi\sqrt{dt} \qquad (2.1.4)$$

φ

- 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

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 rΠdt 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 rΠdt then the arbitrager would make a risk less, no cost,

2.3 Black-Scholes For European Option

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

$$dS = \sigma S dX + (\mu - D_0)S dt \qquad (2.4.1)$$

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

$$-D_0 S \Delta dt \qquad (2.4.2)$$

Adding 2.4.2 to 2.2.4 we arrive at

$$d\Pi = dV - \Delta dS - D_0 S \Delta dt \qquad (2.4.3)$$

Following the same analysis as previously we obtain

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

The only change to the boundary con1.3 is m3 i2441Tf1005057(n)-0.312447(g)0.245057(e)-326.6

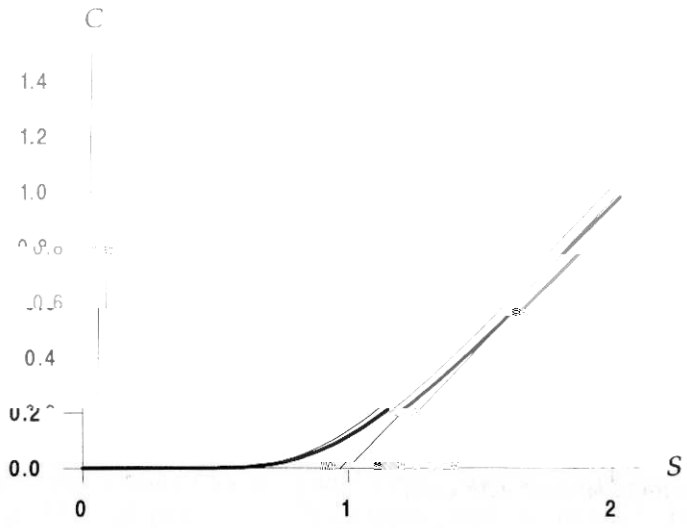


Figure 2.2: European Option Values with (lower) and without (lower) dividends

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

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 payo□ function
- the Black-Scholes equation is replaced by an inequality

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 coe□cient forward equation[1].

We now also subtract o□ the payo□ $S - E$ 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

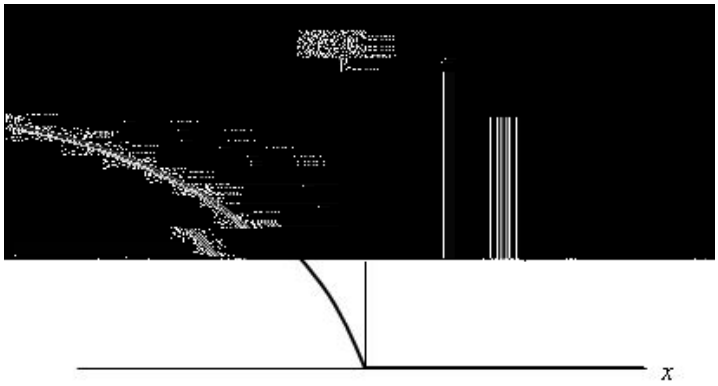


Figure 2.3: c(x,0)

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

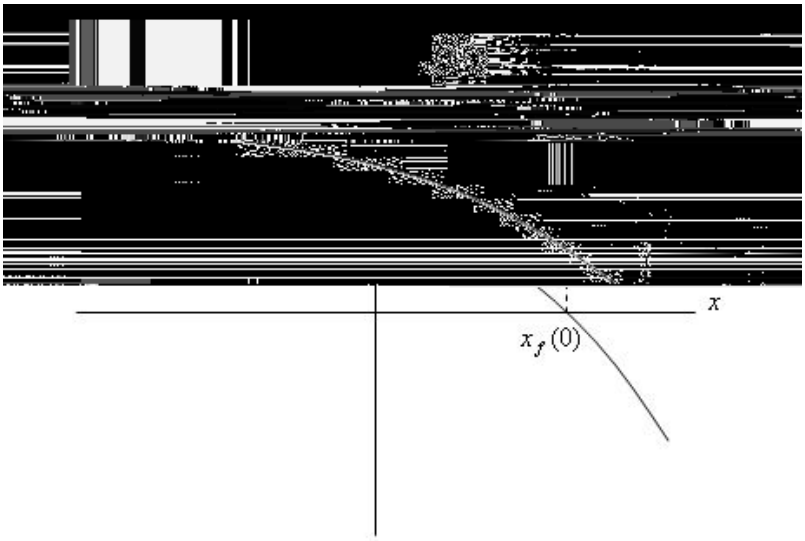
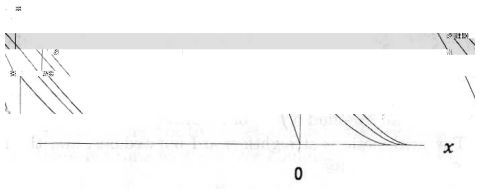


Figure 2.4: f(x)

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 di□usion setting, and the original $C(S, t)$ at times prior to expiry.


x
0

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

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

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

$$(0, B\tilde{(t))} \times [0, T$$

Where

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

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

$$u(x,0) = \max(1 - e^x, 0), x \in (-\infty, B(0)) \qquad (3.2.8)$$

$$B(0) = \max(0, \log\frac{r}{D}) \qquad (3.2.9)$$

$$\lim_{n\to-\infty} u(x,\tau) = e^{k\ \tau}(1 - e^{(x-(k\ -1)\tau)}) \qquad (3.2.10)$$

$$u(B(\tau),\tau) = 0 \qquad (3.2.11)$$

$$u_x(B(\tau),\tau) = 0 \qquad (3.2.12)$$

Chapter 4

Numerical Methods

the moving coordinates x at time t

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

We have

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

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

Applying the chain rule to 4.1.2 gives

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

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

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

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.

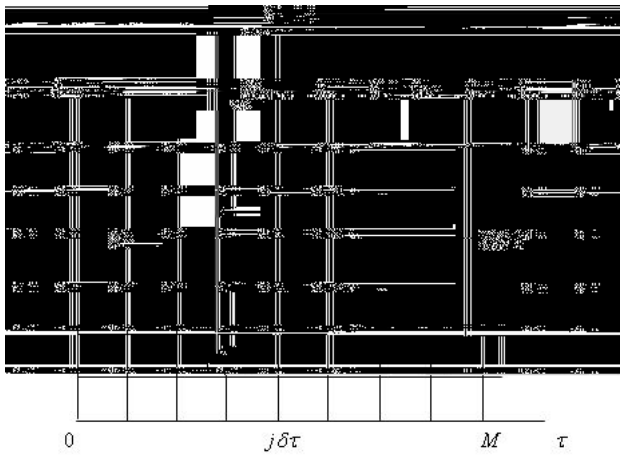


Figure 4.1: Mesh for the finite difference approximation

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

$$\frac{\partial U}{\partial S} \approx \partial U_i^j$$

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

Re-arranging equation 4.1.13 we obtain

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

Where

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

Rearranging 4.1.16 we are left with

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

where

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

The problem is then reduced to solving the system of equations

$$T\mathbf{U}^{j+1} + \beta X_N^{j+1} = B\mathbf{U}^j + \acute{e} \qquad (4.1.17)$$

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 $u_{N-} = u_N$. Writing this as a matrix equation with ??, we have the equations

$$T\bar{u}^{j+1} + \beta x_N = B\bar{u}^j + \acute{e} \qquad (4.1.18)$$

$$h^T\bar{u} = 0$$

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

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

To simplify the notation we absorb the known quantity Bu^j into the d vector.

Writing this matrix equation explicitly we now have

$$\begin{bmatrix} 2-2r & -r & 0 & \cdots & 0 & -\beta_1 \\ -r & 2-2r & -r & & \cdots & -\beta_2 \\ 0 & -r & \ddots & \ddots & 0 & \vdots \\ \vdots & & \ddots & \ddots & -r & \vdots \\ 0 & \cdots & 0 & -r & 2-2r & -\beta_{N-1} \\ 0 & 0 & \cdots & -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} u_1^j \\ u_2^j \\ \vdots \\ \vdots \\ u_{N-}^j \\ x_N^{j+1} \\ 0 \end{bmatrix} = \begin{bmatrix} d_1' \\ d_2' \\ \vdots \\ \vdots \\ d_{N-}' \\ 0 \\ 0 \end{bmatrix}$$

This may be written symbolically as

$$\begin{pmatrix} T & \beta \\ h^T & 0 \end{pmatrix} \begin{pmatrix} u \\ x_N^{j+1} \end{pmatrix} = \begin{pmatrix} d' \\ 0 \end{pmatrix} \qquad (4.1.23)$$

Equation 4.7.5 ca81]218509(5)-327.939(9(5)-327.939(c)0.0490113(a)0.24327.939(c)0.0490113(a)0.

893 8 Tf 6.741 j

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 di□erence

numerical scheme is given by

$$\mathbf{u}^{j}$$

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 $(-\infty, B(0)) \times (0, \frac{1}{2}\sigma^2 T$

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 we expand $f(x)$ by a Taylor series about x .

$$f(x) = f(x) + f'(x)(x - x) + O((x - x))^2 \qquad (4.4.1)$$

$$(x - x)f'(x) = -k(x - x)$$

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

$$\frac{\partial c}{\partial \tau} = \frac{\partial^2 c}{\partial x^2} f($$

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

$$u_{N-} = u_N \qquad (4.5.1)$$

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.

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

$$ay^2 + by = 0 \qquad (4.5.2)$$

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

$$2ay + b = 0 \qquad (4.5.3)$$

The derivaitve is known to be zero at y

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 $\mathbf{u^0}$, $\mathbf{B^0, x^0}$

Approximate the derivative at $\mathbf{x}_N$ *by 'parabola'*

For $\mathbf{j = 1}$ *DO*

- *Set the Velocity of* $\mathbf{x}_N$ *to* $\boldsymbol{\xi_0}\sqrt{\boldsymbol{\tau}}$
- *Rescale* $\mathbf{x}$ *grid points*

- *Solve equation 4 24 for* $\mathbf{x}_{\mathbf{N}}^{i}$

- *Rescale* $\mathbf{x}$ *grid points*

- *Approximate the derivative at* $\mathbf{x}_N$ *by 'parabola'*

- *Solve equation 4 26 for* $\mathbf{u}_i^j$

- *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 e□ect of which is to increase the number of grid points in the local region(s) where the curve is changing rapidly. Likewise, regions

This function gives the position of each nodal point in terms of the free

tive in equation gives

$$\frac{u_i^{j} - u_i^{j}}{\tau} = \theta\left\{ \frac{u_{i-}^{j}}{x_{i-}^{j} - x_i^{j}, x_{i-}^{j} - x_i^{j},} + \frac{u_i^{j}}{x_i^{j} - x_{i-}^{j}, x_i^{j} - x_i^{j},} + \frac{u_i^{j}}{x_i^{j} - x_{i-}^{j}, x_i^{j} - x_i^{j},} \right\}$$

$$+ (1-\theta)\left\{ \frac{u_{i-}^{j}}{x_{i-}^{j} - x_i^{j}, x_{i-}^{j} - x_i^{j},} + \frac{u_i^{j}}{x_i^{j} - x_{i-}^{j}, x_i^{j} - x_i^{j},} + \frac{u_i^{j}}{x_i^{j} - x_{i-}^{j}, x_i^{j} - x_i^{j},} \right\}$$

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

values, $\frac{k}{h}$, 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^{-} to obtain

$$D^{-} T\bar{u} = D^{-} \bar{d} - D^{-} \beta x_N^{j+1} \qquad (4.9.1)$$

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

$$u = \acute{T}^{-} \acute{f} \qquad (4.9.2)$$

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 26 by* D^{-1}
- *Solve equation 4 2 for* u^1

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

END DO

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

- *Solve equation 4 24 for* x

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

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

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

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

$$\acute{\varphi}_i(x) = \begin{cases} 0 & : \quad 0 \leq x \leq x_{i-1} \\ \frac{1}{} & \end{cases}$$

5.2 Weak Form

The weak form of the di□erential equation 3.2.6 is found as follows.

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

$$\int_{x_{i-}(\tau)}^{x_i(\tau)} \varphi_i u_t dx = \int_{x_{i-}(\tau)}^{x_i(\tau)} \varphi_i (u_{xx} + g)\, dx \tag{5.2.1}$$

$$\tag{5.2.2}$$

Since the grid is not fixed

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

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'.

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

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, $1 \leq i \leq N-1$

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

At i = 0

$$-u_x|_{x=x^-} - \int_{x(t)}^{x(t)} \frac{d\varphi_1}{dx} u_x dx + \int_x^x \varphi_1 g dx + \varphi_i u\dot{x}\,|_{x_{i-}(t)}^{x_i(t)} - \int_{x(t)}^{x(t)} \frac{d\varphi_1}{dx} u\dot{x} dx = c_1\dot{\theta} \tag{5.2.5}$$

At i = N

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

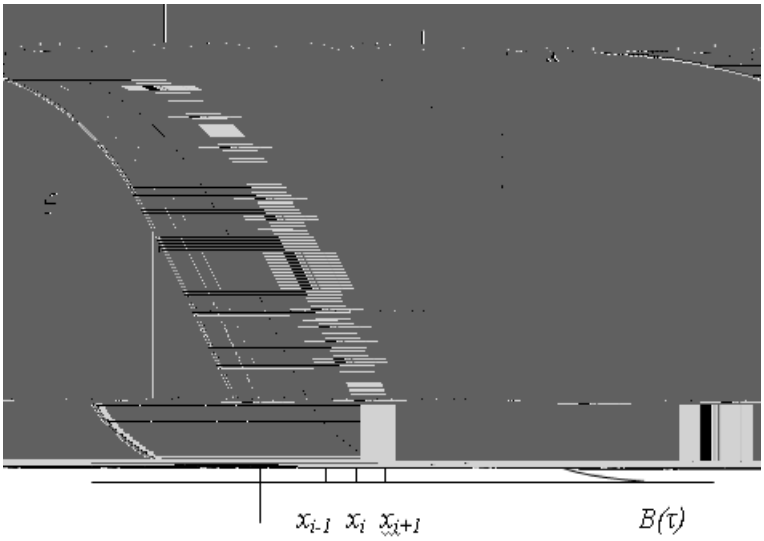


Figure 5.2: Monitor Function

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

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

The problem is then to solve

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

From 5.2.9 we have that

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

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 $\acute{K}_{ij}$ 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 $\mathbf{K}_{ij}$

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

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

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

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 l l lun

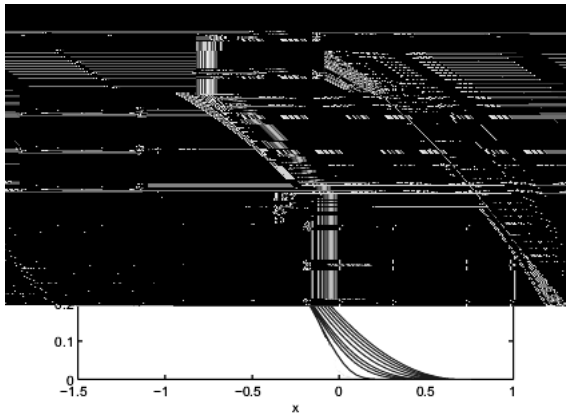


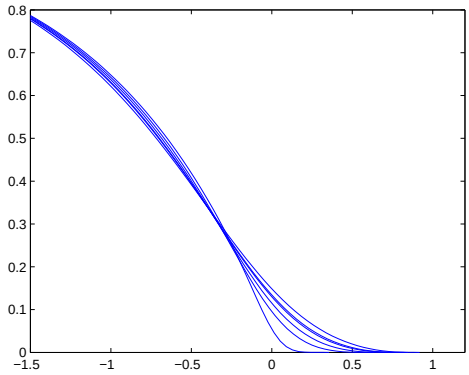
Figure 6.1: $\{T = 1, \sigma = 0.2, r = 0.03, D = 0.8r\}$

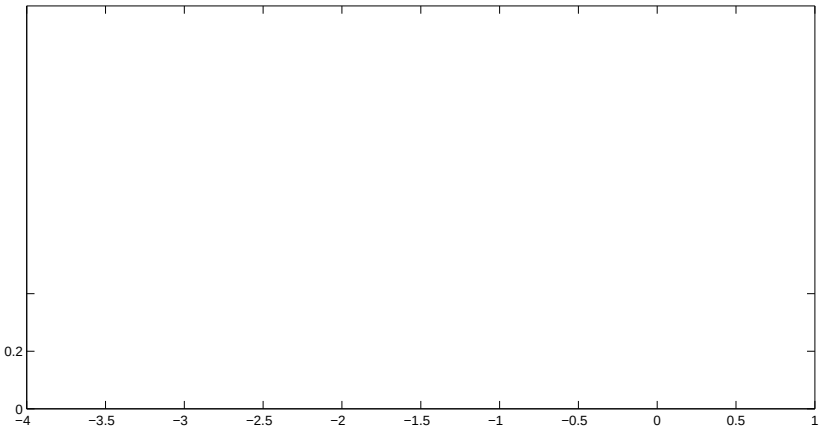
option value as follows

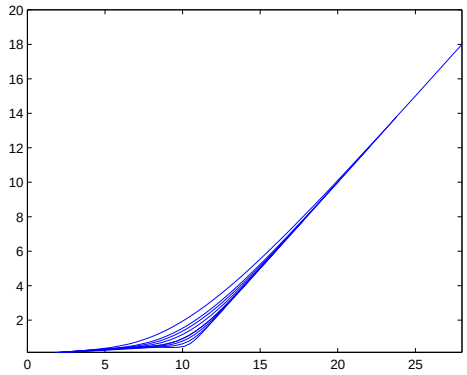
$$\begin{aligned} S &= \exp^{x-(k\ -1)\tau} K \\ t &= -((2\tau)\sigma^2 - T) \\ V(S,t) &= \frac{u(x,\tau)K}{\exp^{k\ \tau}} + S - K \end{aligned} \tag{6.1.2}$$

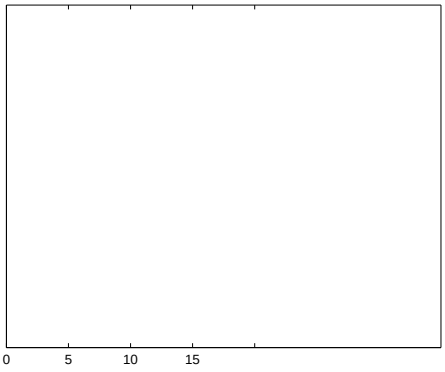
We apply the numerical scheme for the three combinations of the parameters given in 6.1.1, time to expiry is taken as 1.

Figures 6.1,6.1 and 6.1 show the data curves plotted in the dimensionless (Di□usion) setting. A time step of $\Delta\tau = \frac{\sigma}{2}/100$ 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.


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
−1.5
−1
−0.5
0
0.5
1

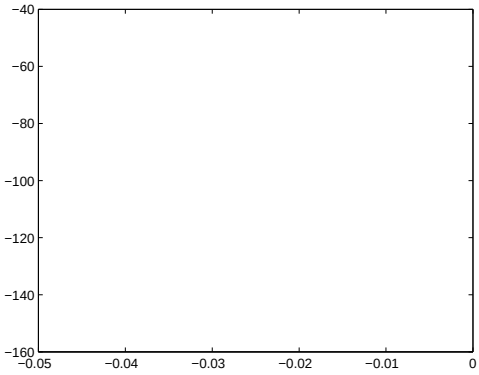

0.2
0
−4
−3.5
−3
−2.5
−2
−1.5
−1
−0.5
0
0.5
1


20
18
16
14
12
10
8
6
4
2
0
5
10
15
20
25


0
5
10
15

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 di□erence method are displayed in column 'FD'. The values generated by our method are approx54245057(n)-0.312447(d)-32kawnly 232


−40
−60
−80
−100
−120
−140
−160
−0.05
−0.04
−0.03
−0.02
−0.01
0

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

In an e□ort to overcome this we take the same approach as section ??, pre-multiplying by D^{-}

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 po-

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 nite element algorithm for the adaptive solution of time-dependent partial di erential 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* **,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(:),
```

```
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)
```

```
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)
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)
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)

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)


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'
      STOP
END IF

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

j=0
DO i=0,N

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

   X2D(j,i)=X(i)

END DO

DO i=0,N

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

END DO

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

    DO i=1,(N-
```

```
CALL CRANK_NICOLSON(N,k,U,RHS,B,X,t,h,DelTau,Tau)

   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

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

    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)
    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

    END DO

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

```
    DO i=0,N

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

        X2D(j,i)=X(i)


    END DO

    DO i=0,N

        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)

    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



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)
 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)
      CLOSE(UNIT=15)
      CLOSE(UNIT=16)
      CLOSE(UNIT=17)
      CLOSE(UNIT=18)
      CLOSE(UNIT=19)
      CLOSE(UNIT=20)
      CLOSE(UNIT=21)
      CLOSE(UNIT=22)
      CLOSE(UNIT=23)
      CLOSE(UNIT=24)
      CLOSE(UNIT=25)
      CLOSE(UNIT=26)
      CLOSE(UNIT=27)
      CLOSE(UNIT=28)
      CLOSE(UNIT=29)
      CLOSE(UNIT=30)

      CLOSE(UNIT=31)
      CLOSE(UNIT=32)
      CLOSE(UNIT=33)
      CLOSE(UNIT=34)
      CLOSE(UNIT=35)
      CLOSE(UNIT=36)
      CLOSE(UNIT=37)
      CLOSE(UNIT=38)
      CLOSE(UNIT=39)
      CLOSE(UNIT=40)
      CLOSE(UNIT=41)
      CLOSE(UNIT=42)
      CLOSE(UNIT=43)
      CLOSE(UNIT=44)
      CLOSE(UNIT=45)
      CLOSE(UNIT=46)
      CLOSE(UNIT=47)
```

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

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



END PROGRAM AMERICAN_OPTIONS707

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


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
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)


   U2D(j,i)=U(i)


END DO

DO i=0,N

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

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

```
DO i=1,N-1

      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)
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

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
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
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)

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))


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))

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

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
```

```
END DO
DO i=0,N

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



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

    V2D(j,i)=RES(i)


    U2D(j,i)=U(i)

END DO

DO i=0,N

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

   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


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
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

r=k/(h**2.0D0)

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


DO i=2,(N-1)


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

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

END DO

DO i=1,N-1

    Z(i)=y(i)

END DO
```

```
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
IN) :                ,DIMENSION(1:N-1)N:)Alp
```

```
      END DO

      DO i=1,N-1

         Y(i)=U(i)

      END DO


      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
      DOUBLE PRECISION::A,B
      INTEGER::i

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

         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),&
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
HT=0.0D0
t=0.0D0



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

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

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



X(0)=LEFT
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)
```

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

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

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

      X(0)=LEFT
```

```
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
INTEGER::i

DO i=0,N
```

```
INTEGER::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)

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


a=2.0*r1(2)
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)


DO i=2,(N-3)

    a=2.0*r1(i)
    b=(2.0D0-2.0*r2(i))3
```

```
END DO

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


END SUBROUTINE TRIDIAG_SOL

SUBROUTINE TRIDIAG_
```

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

END DO

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

DO i=1,N-1

    Z(i)=Q(i)

END DO


END SUBROUTINE TRIDIAG_SOL1

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



s=0.0
Y=0.0

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

    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))

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)

 END DO

 DO i = 1, N-1

    Y(i) = D(i)

END DO


END SUBROUTINE TRIDIAG_SOL2

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

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

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

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

      END DO

      XN=A/B

END SUBROUTINE XN_SOLVE
```

.3 Program 3 - pt3.f90

```
PROGRAM 2914698I
                  291469821.49.I
```

```
M=0.0D0


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

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

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

DO i=0,N
```

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

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

    END DO


    CALL INITIAL_DATA(N,U,X)


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

    DO i=3,N

      HT(i) = 0.0D0

    END DO


    t = 1.0D0

     DO i=0,N-1

          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

    DO q=1,1

      GINTEGRAL=0.0
```

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


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)
END DO


DO i=1,N-1
      CALL SIMPSONS1(X,SUMG1,N,i,q,L)
      CALL SIMPSONS2(X,SUMG2,N,i,q,L)
            G(i)=SUMG1+SUMG2
WRITE(UNIT=15,FMT='(E12.6)')G(i)
END DO
G(N)=G(N-1)
DO i=1,N-1
     CALL MASS_MATRIX(i,N,X,KM)
     CALL MASS_MATRIX_PRIME(i,N,X,KP,U)
END DO
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

      RHS1(j)=U(j-1)*KM(j,j-1)+U(j)*KM(j,j)+U(j+1)*KM(j,j+1)
WRITE(UNIT=31,FMTRT=11)U
```

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

DO j=1,N-1

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

END DO

Do i=1,N-1


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

GINTERGRAL=GINTERGRAL+SUMG

END DO

DERIV=-UPrime(NEGINFINITY,Tau)+GINTERGRAL



PRINT*,THETANEW,'New Theta'


DO j=1,N-1

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

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


END DO

CALL TRIDIAG_SOL(N,Y,KP,Phi)

 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)-&
```

5230Td[(X)32.9876(()225.859(i)282.997())208.95(=)-(4 [illegible] 73581(55]TJ 4603(65]TJ 35817.97011Tf3

```
    END DO

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

    DO i=0,N

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

WRITE(UNIT=13,FMT
```

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

      DO k=1,M-1

            XI=a+k*h

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

                  XI2=XI2+GF(XI,Tau)
            ELSE
                  XI1=XI1+GF(XI,Tau)

            END IF
      END DO

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

   END SUBROUTINE IntegrateG




SUBROUTINE SIMPSONS1(X,SUMG1,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)::SUMG1
      DOUBLE PRECISION::h,XI0,XI2,XI1,a,b,TAU,SUMXI0,MXIa2764Tf11.(k25(N50.E)320.752(P)117.256(R)116.938(
```

```
      b=X(i)

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

      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

            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
      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

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
      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)
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

        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)

        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
```

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




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

END FUNCTION GF



DOUBLE PRECISION FUNCTION UPrime(x,Tau)
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)




      UPrime=-exp(K1*Tau)*(exp(x-(K2-1)    *
```

```
a=X(i
```

```
DO k=1,M-1

        XI=a+k*h

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

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

        END IF

END DO

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

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)
XI1=0.0D0
XI2=0.0D0
XI=0.0

DO k=1,M-1

        XI=a+k*h

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

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

```
        END IF

    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)

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

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

    DO k=1,M-1

        XI=a+k*h

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

                XI2=XI2+func(XI)
        ELSE
            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
    KM(i,i-1)=-1.0D0*(1.0D0/(X(i)-X(i-1)))


END SUBROUTINE MASS_MATRIX
```

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

    func=1.0D0

END FUNCTION func

DOUBLE PRECISION FUNCTION UINTER1(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

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

END FUNCTION UINTER1

 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

        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
    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)
      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
      DOUBLE PRECISION,EXTERNAL::UINTER1,UINTER2
      INTEGER::k,M

      UN=U(i-1)
      UM=U(i (G72.98(N) U(U123702j/R757.97011Tf6.865230G72.98(E(8.726.09342d([U)104.732(N)200.745(=)173.716(U)32.98
```

```
        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)

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
XI=0.0D0

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
                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
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)

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

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
                XI1=XI1+UINTER2(UN,UM,UP,XI,i,a,b)
```

```
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)
```

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


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))
      S(i)=Y(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

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

```
      IMPLICIT NONE
      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

          X(i)=X(i)+k*dPdX(i)

      END DO



END SUBROUTINE EULERS

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

      UN=6.25270T7011Tf16..2021(β4.19U)122.816(M)76.3166(,)110.183(U)60.7188(P)49.7403(,)18625())0.09342774.6[E)65.0729
```

```
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)
XI1=0.0D0
XI2=0.0D0

DO k=1,M-1

        XI=a+k*h

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

                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

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)


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

DO k=1,M-1
```

XI=a+k*h

IF

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



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

DO k=1,M-1

        XI=a+k*h

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

                XI2=XI2+UINTER(UP,UM,XI,i,a,b)

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

        END IF
END DO

SUMC1=h*(XI0+2.0D0*XI2+4.0*
```

```
s = 0.0D0
y = 0.0D0



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

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


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))
      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

      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)
```

```
END DO


END SUBROUTINE TRIDIAG_SOL1

SUBROUTINE TRIDIAG_SOL2(N,Cvec,W,k,KP)
IMPLICIT NONE
INTEGER,INTENT(IN)::N
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,ON¡
```

```
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)

 END DO

 DO i=1,N-1

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


END SUBROUTINE TRIDIAG_SOL2

SUBROUTINE THETA_SOLVE(N,W,Z,HT, ThetaNew)
IM
INTEGER,INTENT(IN)::N
DOUBLE PRECISION,DIMENSION(1:N-1),INTENT(IN)::W,Z,HT
DOUBLE PRECISION,INTENT(OUT)::ThetaNew
DOUBLE PRECISION::A,B
INTEGER::i

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

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

END DO

 ThetaNew=A/B
```

```
END SUBROUTINE THETA_SOLVE

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
    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


    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)


    DO i=2,(N-1)

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

        Alpha(i)=b-(a*c(i-
```

```
   XI2=0.0D0
   XI=0.0

DO k=1,MI-1

        XI=a+k*h

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

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

        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)

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

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

DO k=1,MI-1

        XI=a+k*h

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

                XI2=XI2+((b
```

```
        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)


M(i,i)=SUM1+SUM2


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

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

        XI=a+k*h

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

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

        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)
```

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

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

XI0=(b-
```