

**A Step-By-Step Procedure to The
Numerical Solution for
Time-Dependent Partial
Derivative Equations in Three
Spatial Dimensions**

Xi Wang
Xinyan Lin

Supervisor: Jan Röman
School of Education, Culture and Communication
Mälardalen University
2012.09.17

Declaration

We declare that this thesis was composed by ourselves and that the work contained therein is our own, except where explicitly stated otherwise in the text.

*(Xi Wang
Xinyan Lin)*

Acknowledgement

To our supervisor Jan Röman who is the lecturer of Analytical Finance I and II, assistant vice president and head of market risk in Swedbank in Sweden. First of all, thanks for giving us such a challengeable topic. Secondly, we appreciate your invaluable experience, patience, encouragement and continuous support. Otherwise, this paper would not have been achieved so far. Also, thanks to two PHD students: Karl Lundengård and Christopher Engström in Mälardalen University, they are also the lecturers in the course Applied Matrix Analysis and Numerical Method with Matlab. With their experiences and knowledge, the matrices in programming part have been dealt with successfully.

Abstract

Derivative market has been trading for quite a long time, like options, futures. Meanwhile, interest rate derivative market is the biggest derivative market among the others in the world. In recent years, interest rate derivative arises much concern for various purposes. For example, an investor from one country may have access to getting involved in some derivatives in another country by exposing himself to the exchange rate. However, in this situation, simulation methods to value such derivative might seem inadequate. On the other hand, numerical methods, which have done remarkable job when applied to the valuation, therefore, are strongly recommended in the academic community. This paper consolidates insights from another seminar paper [3] which proposes a Partial Differential Equation (PDE) approach to price cross-currency interest rate derivatives in terms of three correlated processes, uses finite difference methods for the spatial discretization of the PDE, and proves that the *Alternative Direction Implicit* is the most efficient method to solve such PDE. However, no detailed procedure is shown to solve the time-dependent parabolic PDE in three spatial dimensions. Thus, our work focuses on providing step-by-step instructions that enable the reader to follow to solve the parabolic PDE by finite difference method. Meanwhile, this paper is a guide for readers to see how the PDE is applicable to general realistic financial problems.

Contents

Abstract	4
List of Figures	6
1 Introduction	7
2 Derivation of the PDE	8
2.1 Introducing the three dynamic processes and the functions	8
2.2 Ito's lemma	8
3 The Alternative Direction Implicit Scheme	10
3.1 finite differences	10
3.2 Alternating Direction Implicit Method	11
4 The HV scheme	14
4.1 Discover splitting scheme	14
5 Boundary conditions for the ADI method	16
6 Thomas Algorithm	18
7 Summary	19
8 Future Work	20
9 Bibliography	21
A First Appendix	22
B Second Appendix	23
C "Fixed notional" method	24
D Codes in Matlab	25

List of Figures

3.1	The split in three dimensions	12
5.1	Illustration on the boundary conditions and the solution area	17
6.1	Illustration on Diagonal Matrix transformation by Thomas algorithm	18

Chapter 1

Introduction

In this paper, we investigate the cross-currency swap by using the PDE approach. Particularly, it is floating to floating cross-currency swap which involves foreign exchange rate and the interest rates for two currencies. The current standard modeling of such products consist of two one-factor Gaussian models for the term structures and one-factor log-normal model for the spot FX rate [3][10]. However, these three time-dependent variables construct a three dimension problem. Generally, such PDE is considered to be a massively computational challenge if it is solved straightforward. Therefore, we also investigate the efficient and well-known Alternative Direction Implicit (ADI) method to solve the PDE.

In this paper we avoid to discuss the models and the parameters in the models, but only focus on the how the PDE approach is applied to find the value of such derivatives. The outline of the rest of this paper is arranged as follows. In section 2, we present the models and show how the PDE is derived. Section 3 and 4, Alternative Direction Implicit method and the splitting method are demonstrated, so that making a bunch of equations into a nice and acceptable accuracy scheme which results in a convenience computer programming system. Section 5 and 6; generally give a short introduction to the boundary conditions and the Thomas Algorithm. Section 7 and 8 give a short summary and future work.

Chapter 2

Derivation of the PDE

2.1 Introducing the three dynamic processes and the functions

Let $u(s, r_d, r_f, t)$ denote the domestic value function of a security, where s is spot foreign exchange rate; r_d, r_f are the domestic and foreign short rate respectively [3].

Assume the spot FX as follows [9]:

$$ds(t) = (r_d(t) - r_f(t))s(t)dt + \gamma(t, s(t))s(t)dW_s(t) \quad (2.1.1)$$

where $W_s(t)$ is a Brownian motion. r_d and r_f follow the mean-reverting Hull-White model and have the following dynamics [9]:

$$dr_d(t) = (\theta_d(t) - \kappa_d(t)r_d(t))dt + \sigma_d(t)dW_d(t) \quad (2.1.2)$$

$$dr_f(t) = (\theta_f(t) - \kappa_f(t)r_f(t))dt - \rho_{s,f}(t)\sigma_f(t)\gamma(t, s(t))dt + \sigma_f(t)dW_f(t) \quad (2.1.3)$$

where deterministic functions $\kappa_i(t)$ and $\sigma_i(t)$, with $i = d, f$, are the mean reversion rates and the volatility functions. Deterministic function $\theta_i(t)$ is a function of time determining the average direction in which r_i moves. $dW_d(t)$ and $dW_f(t)$ are the other Brownian motions with respect to r_d and r_f .

The "quanto" drift adjustment, $-\rho_{s,f}\sigma_f(t)\gamma(t, s(t))$ for $dr_f(t)$ comes from changing the measure from the foreign risk-neutral measure to the domestic risk neutral one. $\gamma(t, s(t))$ is the local volatility function for the spot FX rate has the functional form [9]

$$\gamma(t, s(t)) = \xi(t) \left(\frac{s(t)}{L(t)} \right)^{\zeta(t)-1} \quad (2.1.4)$$

where $\xi(t)$ is the relative volatility function, $\zeta(t)$ is the time-dependent constant elasticity of variance (CEV) parameter and $L(t)$ is a time-dependent scaling constant [3].

2.2 Ito's lemma

Because the normalized price process of any security is a martingale under the domestic risk-neutral measure, it is easy to apply the Ito's lemma and simply set the drift term to be zero. Ito's lemma gives:

$$\begin{aligned} du &= \frac{\partial u}{\partial t}dt + \frac{\partial u}{\partial s}ds + \frac{\partial u}{\partial r_d}dr_d + \frac{\partial u}{\partial r_f}dr_f + \frac{1}{2}\frac{\partial^2 u}{\partial s^2}ds^2 + \frac{1}{2}\frac{\partial^2 u}{\partial r_d^2}dr_d^2 + \frac{1}{2}\frac{\partial^2 u}{\partial r_f^2}dr_f^2 \\ &+ \frac{\partial^2 u}{\partial r_d \partial s}dsdr_d + \frac{\partial^2 u}{\partial r_f \partial s}dsdr_f + \frac{\partial^2 u}{\partial r_f \partial r_d}dr_d dr_f \end{aligned} \quad (2.2.1)$$

Substituting the dynamics gives:

$$\begin{aligned}
du &= \frac{\partial u}{\partial t} dt + \frac{\partial u}{\partial s} ((r_d(t) - r_f(t))s(t)dt + \gamma(t, s(t))s(t)dW_s(t)) \\
&+ \frac{\partial u}{\partial r_d} ((\theta_d(t) - \kappa_d(t))r_d(t)dt + \sigma_d(t)dW_d(t)) \\
&+ \frac{\partial u}{\partial r_f} ((\theta_f(t) - \kappa_f(t))r_f(t)dt - \rho_{fs}(t)\sigma_f(t)\gamma(t, s(t))dt + \sigma_f(t)dW_f(t)) \\
&+ \frac{1}{2} \frac{\partial^2 u}{\partial s^2} (\gamma^2(t, s(t))s^2(t)dt) + \frac{1}{2} \frac{\partial^2 u}{\partial r_d^2} (\sigma_d^2(t)dt) + \frac{1}{2} \frac{\partial^2 u}{\partial r_f^2} (\sigma_f^2(t)dt) \\
&+ \frac{\partial^2 u}{\partial s \partial r_d} (\rho_{s,d}\sigma_d(t)\gamma(t, s(t))sdt) + \frac{\partial^2 u}{\partial s \partial r_f} (\rho_{s,f}\sigma_f(t)\gamma(t, s(t))sdt) \\
&+ \frac{\partial^2 u}{\partial r_d \partial r_f} (\rho_{d,f}\sigma_d(t)\sigma_f(t)dt) \tag{2.2.2}
\end{aligned}$$

Rearrange the above equation and group the $-dt$ terms, because of the martingale property, by setting these terms equal to zero the PDE is derived as:

$$\begin{aligned}
&\frac{\partial u}{\partial t} + (r_d - r_f)s \frac{\partial u}{\partial s} + (\theta_d(t) - \kappa_d(t))r_d \frac{\partial u}{\partial r_d} \\
&+ (\theta_f(t) - \kappa_f(t))r_f - \rho_{f,s}\sigma_f(t)\gamma(t, s(t)) \frac{\partial u}{\partial r_f} \\
&+ \frac{1}{2} \gamma^2(t, s(t))s^2 \frac{\partial^2 u}{\partial s^2} + \frac{1}{2} \sigma_d^2(t) \frac{\partial^2 u}{\partial r_d^2} + \frac{1}{2} \sigma_f^2(t) \frac{\partial^2 u}{\partial r_f^2} \\
&+ \rho_{s,d}\sigma_d(t)\gamma(t, s(t))s \frac{\partial^2 u}{\partial s \partial r_d} + \rho_{s,f}\sigma_f(t)\gamma(t, s(t))s \frac{\partial^2 u}{\partial s \partial r_f} + \rho_{d,f}\sigma_d(t)\sigma_f(t) \frac{\partial^2 u}{\partial r_d \partial r_f} \\
&- r_d u \\
&= 0 \tag{2.2.3}
\end{aligned}$$

Chapter 3

The Alternative Direction Implicit Scheme

3.1 finite differences

In the discretization, the first and second derivative terms can all be approximated by the central difference method as presented following [3][7]:

$$\frac{\partial u}{\partial s} \approx \frac{u_{i+1,j,k}^m - u_{i-1,j,k}^m}{2\Delta s}, \quad \frac{\partial^2 u}{\partial s^2} \approx \frac{u_{i+1,j,k}^m - 2u_{i,j,k}^m + u_{i-1,j,k}^m}{\Delta s^2} \quad (3.1.1)$$

Similarly,

$$\frac{\partial u}{\partial r_d} \approx \frac{u_{i,j+1,k}^m - u_{i,j-1,k}^m}{2\Delta r_d}, \quad \frac{\partial^2 u}{\partial r_d^2} \approx \frac{u_{i,j+1,k}^m - 2u_{i,j,k}^m + u_{i,j-1,k}^m}{\Delta r_d^2} \quad (3.1.2)$$

$$\frac{\partial u}{\partial r_f} \approx \frac{u_{i,j,k+1}^m - u_{i,j,k-1}^m}{2\Delta r_f}, \quad \frac{\partial^2 u}{\partial r_f^2} \approx \frac{u_{i,j,k+1}^m - 2u_{i,j,k}^m + u_{i,j,k-1}^m}{\Delta r_f^2} \quad (3.1.3)$$

$$\frac{\partial u}{\partial t} \approx \frac{u_{i,j,k}^m - u_{i,j,k}^{m-1}}{\Delta t} \quad (3.1.4)$$

While the cross-derivative terms are approximated by a four-point finite difference:

$$\frac{\partial^2 u}{\partial r_d \partial s} = \frac{u_{i+1,j+1,k}^m + u_{i-1,j-1,k}^m - u_{i-1,j+1,k}^m - u_{i+1,j-1,k}^m}{4\Delta s \Delta r_d} \quad (3.1.5)$$

$$\frac{\partial^2 u}{\partial r_f \partial s} = \frac{u_{i+1,j,k+1}^m + u_{i-1,j,k-1}^m - u_{i-1,j,k+1}^m - u_{i+1,j,k-1}^m}{4\Delta s \Delta r_f} \quad (3.1.6)$$

$$\frac{\partial^2 u}{\partial r_d \partial r_f} = \frac{u_{i,j+1,k+1}^m + u_{i,j-1,k-1}^m - u_{i,j-1,k+1}^m - u_{i,j+1,k-1}^m}{4\Delta r_f \Delta r_d} \quad (3.1.7)$$

Where, m denotes the time index, and i, j, k denote the s^- , r_d^- , r_f^- direction.

Because we solve the PDE backward in time, by changing variable $\tau = T_{end} - T_{start}$ [3] the above differential equation 2.2.3 can be rewritten as following when substitute the finite differences in

the corresponding terms:

$$\begin{aligned}
& (r_d - r_f)s \left(\frac{u_{i+1,j,k}^m - u_{i-1,j,k}^m}{2\Delta s} \right) + \frac{1}{2}\gamma^2(t, s(t))s^2 \left(\frac{u_{i+1,j,k}^m - 2u_{i,j,k}^m + u_{i-1,j,k}^m}{\Delta s^2} \right) \\
+ & (\theta_d(t) - \kappa_d(t)r_d) \left(\frac{u_{i,j+1,k}^m - u_{i,j-1,k}^m}{2\Delta r_d} \right) + \frac{1}{2}\sigma_d^2(t) \left(\frac{u_{i,j+1,k}^m - 2u_{i,j,k}^m + u_{i,j-1,k}^m}{\Delta r_d^2} \right) \\
+ & (\theta_f(t) - \kappa_f(t)r_f - \rho_{f,s}\sigma_f(t)\gamma(t, s(t))) \left(\frac{u_{i,j,k+1}^m - u_{i,j,k-1}^m}{2\Delta r_f} \right) \\
+ & \frac{1}{2}\sigma_f^2(t) \left(\frac{u_{i,j,k+1}^m - 2u_{i,j,k}^m + u_{i,j,k-1}^m}{\Delta r_f^2} \right) \\
+ & \rho_{s,d}\sigma_d(t)\gamma(t, s(t))s \left(\frac{u_{i+1,j+1,k}^m + u_{i-1,j-1,k}^m - u_{i-1,j+1,k}^m - u_{i+1,j-1,k}^m}{4\Delta s\Delta r_d} \right) \\
+ & \rho_{s,f}\sigma_f(t)\gamma(t, s(t))s \left(\frac{u_{i+1,j,k+1}^m + u_{i-1,j,k-1}^m - u_{i-1,j,k+1}^m - u_{i+1,j,k-1}^m}{4\Delta s\Delta r_f} \right) \\
+ & \rho_{d,f}\sigma_d(t)\sigma_f(t) \left(\frac{u_{i,j+1,k+1}^m + u_{i,j-1,k-1}^m - u_{i,j-1,k+1}^m - u_{i,j+1,k-1}^m}{4\Delta r_f\Delta r_d} \right) \\
- & r_d u \\
= & \left(\frac{u_{i,j,k}^m - u_{i,j,k}^{m-1}}{\Delta\tau} \right) \tag{3.1.8}
\end{aligned}$$

Rearrange in terms of the directions i^- , j^- , k^- :

$$\begin{aligned}
& \left(\frac{(r_d - r_f)s}{2\Delta s} + \frac{\gamma^2(t, s(t))s^2}{2\Delta s^2} \right) u_{i+1,j,k}^m + \left(-\frac{(r_d - r_f)s}{2\Delta s} + \frac{\gamma^2(t, s(t))s^2}{2\Delta s^2} \right) u_{i-1,j,k}^m \\
- & \frac{\gamma^2(t, s(t))s^2}{\Delta s^2} u_{i,j,k}^m + \left(\frac{\theta_d(t) - \kappa_d(t)r_d}{2\Delta r_d} + \frac{\sigma_d^2(t)}{2\Delta r_d^2} \right) u_{i,j+1,k}^m \\
+ & \left(-\frac{\theta_d(t) - \kappa_d(t)r_d}{2\Delta r_d} + \frac{\sigma_d^2(t)}{2\Delta r_d^2} \right) u_{i,j-1,k}^m - \frac{\sigma_d^2(t)}{\Delta r_d^2} u_{i,j,k}^m \\
+ & \left(\frac{\theta_f(t) - \kappa_f(t)r_f - \rho_{f,s}\sigma_f(t)\gamma(t, s(t))}{2\Delta r_f} + \frac{\sigma_f^2(t)}{2\Delta r_f^2} \right) u_{i,j,k+1}^m - \frac{\sigma_f^2(t)}{\Delta r_f^2} u_{i,j,k}^m \\
+ & \left(-\frac{\theta_f(t) - \kappa_f(t)r_f - \rho_{f,s}\sigma_f(t)\gamma(t, s(t))}{2\Delta r_f} + \frac{\sigma_f^2(t)}{2\Delta r_f^2} \right) u_{i,j,k-1}^m \\
+ & \frac{\rho_{s,d}\sigma_d(t)\gamma(t, s(t))s}{4\Delta s\Delta r_d} (u_{i+1,j+1,k}^m + u_{i-1,j-1,k}^m - u_{i-1,j+1,k}^m - u_{i+1,j-1,k}^m) \\
+ & \frac{\rho_{s,f}\sigma_f(t)\gamma(t, s(t))s}{4\Delta s\Delta r_f} (u_{i+1,j,k+1}^m + u_{i-1,j,k-1}^m - u_{i-1,j,k+1}^m - u_{i+1,j,k-1}^m) \\
+ & \frac{\rho_{d,f}\sigma_d(t)\sigma_f(t)}{4\Delta r_f\Delta r_d} (u_{i,j+1,k+1}^m + u_{i,j-1,k-1}^m - u_{i,j-1,k+1}^m - u_{i,j+1,k-1}^m) \\
- & r_d u \\
= & \frac{u_{i,j,k}^m - u_{i,j,k}^{m-1}}{\Delta\tau} \tag{3.1.9}
\end{aligned}$$

3.2 Alternating Direction Implicit Method

The basic idea behind the ADI method is to split the finite difference equations into different directions. Since in our case, the three dimensions form a cube for every time step, Figure 3.1 demonstrates the idea.

For each split, only one direction is taken implicit. As the procedure continues, the exact solution will be found. From the finite difference method, Equation 3.1.9 can be written in the

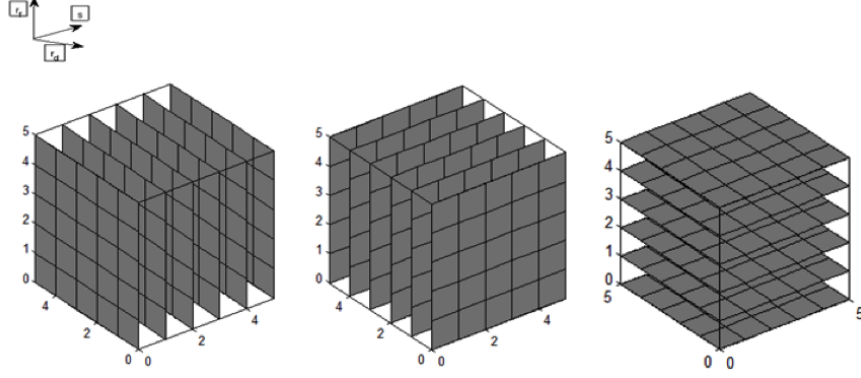


Figure 3.1: The split in three dimensions

form:

$$A_1^m + A_2^m + A_3^m + A_0^m = \frac{u^m - u^{m-1}}{\Delta\tau} \quad (3.2.1)$$

Where A_1^m, A_2^m, A_3^m denote the terms in each direction. A_0^m denotes the cross derivative terms, are treated in an explicit fashion. And assume the term $r_d u$ is distributed evenly over A_1^m, A_2^m, A_3^m [3]. Therefore:

$$\begin{aligned} A_1^m &= \frac{\partial u}{\partial s} + \frac{\partial^2 u}{\partial s^2} - \frac{1}{3}r_d \\ A_2^m &= \frac{\partial u}{\partial r_d} + \frac{\partial^2 u}{\partial r_d^2} - \frac{1}{3}r_d \\ A_3^m &= \frac{\partial u}{\partial r_f} + \frac{\partial^2 u}{\partial r_f^2} - \frac{1}{3}r_d \\ A_0^m &= \frac{\partial^2 u}{\partial r_d \partial s} + \frac{\partial^2 u}{\partial r_d \partial r_f} + \frac{\partial^2 u}{\partial r_f \partial s} \end{aligned} \quad (3.2.2)$$

In order to see the tri-diagonal, we can do the following transformation:

$$\begin{aligned} A_1^m &= a * u_{i+1,j,k}^m + b * u_{i-1,j,k}^m - \frac{\gamma^2 s^2}{\Delta s^2} * u_{i,j,k}^m - \frac{1}{3}r_d * u_{i,j,k}^m \\ A_2^m &= c * u_{i,j+1,k}^m + d * u_{i,j-1,k}^m - \frac{\sigma_d^2}{\Delta r_d^2} * u_{i,j,k}^m - \frac{1}{3}r_d * u_{i,j,k}^m \\ A_3^m &= e * u_{i,j,k+1}^m + f * u_{i,j,k-1}^m - \frac{\sigma_f^2}{\Delta r_f^2} * u_{i,j,k}^m - \frac{1}{3}r_d * u_{i,j,k}^m \end{aligned} \quad (3.2.3)$$

In such form, the following matrices represent the A_1^m, A_2^m, A_3^m respectively:

$$\begin{aligned} A_1^m &= \begin{bmatrix} -\frac{\gamma^2 s^2}{\Delta s^2} - \frac{1}{3}r_d & a & \cdots & 0 \\ b & \ddots & & \vdots \\ 0 & \cdots & & -\frac{\gamma^2 s^2}{\Delta s^2} - \frac{1}{3}r_d \end{bmatrix} \\ A_2^m &= \begin{bmatrix} -\frac{\sigma_d^2}{\Delta r_d^2} - \frac{1}{3}r_d & c & \cdots & 0 \\ d & \ddots & & \vdots \\ 0 & \cdots & & -\frac{\sigma_d^2}{\Delta r_d^2} - \frac{1}{3}r_d \end{bmatrix} \end{aligned}$$

$$A_3^m = \begin{bmatrix} -\frac{\sigma_f^2}{\Delta r_f^2} - \frac{1}{3}r_d & e & \cdots & 0 \\ f & \ddots & & \vdots \\ 0 & \cdots & & -\frac{\sigma_f^2}{\Delta r_f^2} - \frac{1}{3}r_d \end{bmatrix}$$

Where

$$\begin{aligned} a &= \frac{(r_d - r_f)s}{2\Delta s} + \frac{\gamma^2 s^2}{2\Delta s^2} \\ b &= -\frac{(r_d - r_f)s}{2\Delta s} + \frac{\gamma^2 s^2}{2\Delta s^2} \\ c &= \frac{(\theta_d - \kappa_d r_d)}{2\Delta r_d} + \frac{\sigma_d^2}{2\Delta r_d^2} \\ d &= -\frac{(\theta_d - \kappa_d r_d)}{2\Delta r_d} + \frac{\sigma_d^2}{2\Delta r_d^2} \\ e &= \frac{(\theta_f - \kappa_f r_f - \rho_{s,f} \sigma_f \gamma)}{2\Delta r_f} + \frac{\sigma_f^2}{2\Delta r_f^2} \\ f &= -\frac{(\theta_f - \kappa_f r_f - \rho_{s,f} \sigma_f \gamma)}{2\Delta r_f} + \frac{\sigma_f^2}{2\Delta r_f^2} \end{aligned}$$

Chapter 4

The HV scheme

The Hundsdorfer and Verwer (HV) [5] splitting scheme as suggested in [3] approximates the exact solution U^m :

$$\begin{cases} V_0 & = U^{m-1} + \Delta(A^{m-1}U^{m-1} + g^{m-1}) \\ (I - \frac{1}{2}\Delta\tau A_i^m)V_i & = V_{i-1} - \frac{1}{2}\Delta\tau A_i^{m-1} + \frac{1}{2}\Delta\tau(g_i^m - g_i^{m-1}), & i = 1, 2, 3 \\ \tilde{V}_0 & = V_0 + \frac{1}{2}\Delta\tau(A^m V_3 - A^{m-1}U^{m-1}) + \frac{1}{2}\Delta\tau(g^m - g^{m-1}) \\ (I - \frac{1}{2}\Delta\tau A_i^m)\tilde{V}_i & = \tilde{V}_{i-1} - \frac{1}{2}\Delta\tau A_i^m V_3, & i = 1, 2, 3 \\ U^m & = \tilde{V}_3. \end{cases}$$

Here the vector g^m is the boundary conditions for the corresponding directions, will be discussed in the later section. In our case, the boundary conditions form a cube. Besides, from now on, the A matrices are defined in a slightly different manner because of the computational and programming purpose; they are presented in the Appendix A.

4.1 Discover splitting scheme

Applying the Douglas [4] ADI scheme to the equation 3.2.1, obtain a first evaluate of the solution at time $m + 1$ by taking implicitly half of the A_1 :

$$\frac{1}{2}A_1(u^{m+\frac{1}{3}} + u^m) + A_2u^m + A_3u^m + A_0u^m = \frac{u^{m+\frac{1}{3}} - u^m}{\Delta\tau} \quad (4.1.1)$$

Then, taking half of the A_2, A_3 respectively,

$$\frac{1}{2}A_1(u^{m+\frac{1}{3}} + u^m) + \frac{1}{2}A_2(u^{m+\frac{2}{3}} + u^m) + A_3u^m + A_0u^m = \frac{u^{m+\frac{2}{3}} - u^m}{\Delta\tau} \quad (4.1.2)$$

$$\frac{1}{2}A_1(u^{m+\frac{1}{3}} + u^m) + \frac{1}{2}A_2(u^{m+\frac{2}{3}} + u^m) + \frac{1}{2}A_3(u^{m+1} + u^m) + A_0u^m = \frac{u^{m+1} - u^m}{\Delta\tau} \quad (4.1.3)$$

The intermediate values $u^{m+\frac{1}{3}}, u^{m+\frac{2}{3}}$ can be eliminated. The reader is strongly recommended to the reference [4] for detailed elimination procedures.

For splitting purpose, the above can be rewritten as:

$$(I - \frac{1}{2}\Delta\tau A_1)u^{m+\frac{1}{3}} = u^m + \Delta\tau(A_2u^m + A_3u^m + A_0u^m) + \frac{1}{2}\Delta\tau A_1u^m \quad (4.1.4)$$

$$(I - \frac{1}{2}\Delta\tau A_2)u^{m+\frac{2}{3}} = u^m + \frac{1}{2}\Delta\tau A_1(u^{m+\frac{1}{3}} + u^m) + \frac{1}{2}A_2u^m + A_3u^m + A_0u^m \quad (4.1.5)$$

$$\begin{aligned} (I - \frac{1}{2}\Delta\tau A_3)u^{m+1} &= u^m + \frac{1}{2}\Delta\tau A_1(u^{m+\frac{1}{3}} + u^m) + \frac{1}{2}A_2(u^{m+\frac{2}{3}} + u^m) \\ &\quad + \frac{1}{2}A_3u^m + A_0u^m \end{aligned} \quad (4.1.6)$$

For the programming purpose, the above can be sorted as:

$$\begin{cases} V_0 = U^m + \Delta\tau(A^m U^m) \\ (I - \frac{1}{2}\Delta\tau A_i)V_i = V_{i-1} - \frac{1}{2}\Delta\tau A_i^{m-1}U^m \quad i = 1, 2, 3 \end{cases} \quad (4.1.7)$$

Where A^m is the sum of the A_0, A_1, A_2, A_3 , and V_i with $i = 1, 2, 3$ is corresponding to $u^{m+\frac{1}{3}}, u^{m+\frac{2}{3}}, u^{m+1}$ in equation 4.1.4, 4.1.5, 4.1.6.

The scheme 4.1.7 is so called Douglas-Rachford method (DR method). The HV method is derived in a similar manner, while, taking more steps to stabilize the solution on each step to ensure the final solution, can be viewed as an extension of DR method. In 4.1.4, there are actually $I + 1$ equations, but only three unknown variables for each equation regardless the boundary conditions, namely $u_{i-1,j,k}^{m+\frac{1}{3}}, u_{i,j,k}^{m+\frac{1}{3}}, u_{i+1,j,k}^{m+\frac{1}{3}}$. Similarly, in j and k directions, $u_{i,j-1,k}^{m+\frac{2}{3}}, u_{i,j,k}^{m+\frac{2}{3}}, u_{i,j+1,k}^{m+\frac{2}{3}}, u_{i,j,k-1}^{m+1}, u_{i,j,k}^{m+1}, u_{i,j,k+1}^{m+1}$ are unknown. As stated earlier, A_1, A_2, A_3 are tri-diagonal matrices, therefore, the equations 4.1.4, 4.1.5, 4.1.6 can be expressed below:

$$\left(I - \frac{1}{2}\Delta\tau A_1\right) \begin{bmatrix} u_{1,j,k}^{m+\frac{1}{3}} \\ m+\frac{1}{3} \\ u_{2,j,k}^{m+\frac{1}{3}} \\ m+\frac{1}{3} \\ u_{3,j,k}^{m+\frac{1}{3}} \\ m+\frac{1}{3} \\ u_{4,j,k}^{m+\frac{1}{3}} \\ \vdots \\ m+\frac{1}{3} \\ u_{i,j,k}^{m+\frac{1}{3}} \end{bmatrix} = [\omega_1]$$

$$\left(I - \frac{1}{2}\Delta\tau A_2\right) \begin{bmatrix} u_{i,1,k}^{m+\frac{2}{3}} \\ m+\frac{2}{3} \\ u_{i,2,k}^{m+\frac{2}{3}} \\ m+\frac{2}{3} \\ u_{i,3,k}^{m+\frac{2}{3}} \\ m+\frac{2}{3} \\ u_{i,4,k}^{m+\frac{2}{3}} \\ \vdots \\ m+\frac{2}{3} \\ u_{i,j,k}^{m+\frac{2}{3}} \end{bmatrix} = [\omega_2]$$

$$\left(I - \frac{1}{2}\Delta\tau A_3\right) \begin{bmatrix} u_{i,j,1}^{m+1} \\ m+1 \\ u_{i,j,2}^{m+1} \\ m+1 \\ u_{i,j,3}^{m+1} \\ m+1 \\ u_{i,j,4}^{m+1} \\ \vdots \\ m+1 \\ u_{i,j,k}^{m+1} \end{bmatrix} = [\omega_3]$$

ω_1 is a $(I + 1) \times 1$ dimension vector with components:

$$\omega_1 = u^m + \Delta\tau(A_2 u^m + A_3 u^m + A_0 u^m) + \frac{1}{2}\Delta\tau A_1 u^m$$

Similar to ω_2 and ω_3 , but the dimensions are $(J + 1) \times 1$ and $(K + 1) \times 1$ respectively.

$$\omega_2 = u^m + \frac{1}{2}\Delta\tau A_1(u^{m+\frac{1}{3}} + u^m) + \frac{1}{2}A_2 u^m + A_3 u^m + A_0 u^m$$

$$\omega_3 = u^m + \frac{1}{2}\Delta\tau A_1(u^{m+\frac{1}{3}} + u^m) + \frac{1}{2}A_2(u^{m+\frac{2}{3}} + u^m) + \frac{1}{2}A_3 u^m + A_0 u^m$$

Chapter 5

Boundary conditions for the ADI method

In numerical method, it is unnecessary to construct the whole universe; therefore, work must be done to decide the region we are only interested in. Proper boundary conditions can significantly reduce the computation time and increase the accuracy of the results. In other words, the boundary condition is one of the most significant parts of such PDE functions. There are many boundary conditions can be applied to PDE function, for example, Dirichlet boundary conditions, Neumann boundary conditions and mixed boundary conditions. Nevertheless, some boundary conditions can be very complicated. For the Neumann boundary conditions, instead of having fixed values on the bonded surfaces, it uses the derivatives. For instance, in the s^- direction, the second-order one-sided approximation gives the following:

$$\begin{aligned}\frac{\partial u}{\partial s} &\approx \frac{4 * u_{i+1,j,k}^m - u_{i+2,j,k}^m - 3 * u_{i,j,k}^m}{2\Delta s} = g_1 \quad \text{forward} \\ \frac{\partial u}{\partial s} &\approx \frac{u_{i-2,j,k}^m - 4 * u_{i-1,j,k}^m + 3 * u_{i,j,k}^m}{2\Delta s} = g_2 \quad \text{backward}\end{aligned}$$

The start and end surfaces are $u_{0,j,k}^m$, $u_{I,j,k}^m$ and then, the $u_{0,j,k}^m$ and $u_{I,j,k}^m$ are solved from the forward and backward differences which will result in the different coefficients in the matrix on the left hand side of the equation as well as the right hand side. This boundary condition is more complicated in implementations. We will continue to discuss this topic in the next paper, where we will do the calibrations and show different boundary conditions result in comparable results. For now, in this paper, we simply apply the Dirichlet boundary conditions where there are fixed values on the end surfaces. This type of the boundary condition takes the values on the boundary points as known, but only changes the right hand side of the equation when the boundary conditions are considered.

$$\begin{bmatrix} F_1 - g_1 \\ F_2 \\ F_3 \\ \vdots \\ \vdots \\ F_{n-1} \\ F_n - g_n \end{bmatrix}$$

To be more explicit, in our case, the boundary conditions are surfaces. Figure 5.1 below illustrates the boundaries and the area we are really concern about which is indicated by the red hypercube. Readers can get helps to understand the boundary conditions by having a look at the [1, 2, 6] in the reference.

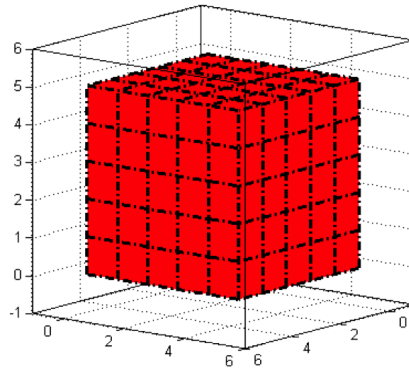


Figure 5.1: Illustration on the boundary conditions and the solution area

Chapter 6

Thomas Algorithm

HV scheme has made the algorithm clear enough to solve the U^m , but since the algorithm involves solving a large number of systems of equations which may cause difficulties in the implementation. Fortunately, knowing that A_1, A_2, A_3 are three tri-diagonal matrices, the Thomas algorithm [11] has been invented and recommended to help us to do the job. The basic idea behind the algorithm is substitution which will result in a new tri-diagonal matrix. Of course, the vector on the right hand side of the equation will be substituted as well. However, the advantage is, the elements in sub-diagonal in new tri-diagonal matrix are zeros, and the diagonal elements are ones. Figure 6.1 illustrates how the diagonal matrix is transformed using the Thomas algorithm.


$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 \\ 0 & 0 & 0 & 0 & b_6 & c_6 \end{pmatrix}$$

$$\begin{pmatrix} 1 & \gamma_1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \gamma_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \gamma_3 & 0 & 0 \\ 0 & 0 & 0 & 1 & \gamma_4 & 0 \\ 0 & 0 & 0 & 0 & 1 & \gamma_5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 6.1: Illustration on Diagonal Matrix transformation by Thomas algorithm

Chapter 7

Summary

Up until here we have made complete procedures to the numerical method for three dimensional partial derivative equations. As can be seen, such problems could be extremely complex. The matrices we illustrated on this paper only reflect the general forms. When the indices are considered, we are actually dealing with hundreds of equations. Thus, to find the final exact solution, every step is critical; our goal is to efficiently obtain the solution with high accuracy. Thanks to the other academicians; the problems in the real situations can successfully be solved even with different order of errors. The programming languages have been playing a vital part, for example the MATLAB, C++, etc., in the appendix; we enclose the MATLAB program for solving such problems. However, the program has to be modified for any other specific type of instruments. Therefore, in this paper, we skip analyzing the results from the program but only give a short insight into it. More details will be covered in the next paper.

Chapter 8

Future Work

So far, this paper only gives the general understanding to the numerical method of solving partial differential equations in three dimensions. As has been said in the introduction, this paper has not talked about any parameters in the formulas; obviously, those parameters will significantly influence the results. In the next paper, will test the program and try to calibrate the parameters from the real market data. When dealing with the real market data, the boundary conditions we consider in the paper may not be suitable anymore because of the time-dependent variables. Thus, to achieve great accuracy, more specific boundary conditions will be included. Finally, the MATLAB program built in the way that suits for general instruments whichever involves three dimensions, estimating some of such popularly traded instruments will be illustrated in the next paper as well.

Chapter 9

Bibliography

- [1] Nasser M. Abbasi. Neumann boundary conditions on 2d grid with non-uniform mesh space for elliptic pde, 2012.
- [2] Wolfgang Arendt and Mahamadi Warma. Dirichlet and neumann boundary conditions: what is in between? *Journal of Evolution Equations*, 3:119–135, 2003.
- [3] Duy Minh Dang, Christina C. Christara, Kenneth R. Jackson, and Asif Lakhany. A pde pricing framework for cross-currency interest rate derivatives. *Procedia CS*, 1(1):2371–2380, 2010.
- [4] Jim Douglas. Alternating direction methods for three space variables. *Numerische Mathematik*, 4:41–63, 1962.
- [5] K.J. in 't Hout and B.D. Welfert. Unconditional stability of second-order adi schemes applied to multi-dimensional diffusion equations with mixed derivative terms. *Applied Numerical Mathematics*, 59(34):677 – 692, 2009. `{ce:title}Selected Papers from NUMDIFF-11{/ce:title}`.
- [6] J.Izadian and S.S.Jalalian. A new method for solving 3d elliptic problem with dirichlet or neumann boundary conditions using finite difference method. *Applied Mathematical Sciences*, 6(34):1655–1666, 2012.
- [7] Zhilin Li. Finite difference method basics.
- [8] Sensen Lin. Finite difference schemes for heston model, 2008.
- [9] Vladimir Piterbarg. Smiling hybrids. *Risk magazine*, pages 66–70, May 2006.
- [10] Jason Sippel and Shoichi Ohkoshi. All power to prdc notes. *Risk magazine*, pages 31–33, November 2002.
- [11] W.T.Lee. Tridiagonal matrices: Thomas algorithm.

Appendix A

First Appendix

For the programming purpose, the following transformations are needed which are based on Lin's work[8]. We let:

$$\begin{aligned}
\frac{\Delta t}{\Delta s} &= R_s, \frac{\Delta t}{\Delta s^2} = R_{ss}, u_{i+1,j,k}^m - u_{i-1,j,k}^m = \delta_s * u_{i,j,k}^m, \\
u_{i+1,j,k}^m - 2u_{i,j,k}^m + u_{i-1,j,k}^m &= \delta_{ss} * u_{i,j,k}^m \\
\frac{\Delta t}{\Delta r_d} &= R_d, \frac{\Delta t}{\Delta r_d^2} = R_{dd}, u_{i,j+1,k}^m - u_{i,j-1,k}^m = \delta_d * u_{i,j,k}^m, \\
u_{i,j+1,k}^m - 2u_{i,j,k}^m + u_{i,j-1,k}^m &= \delta_{dd} * u_{i,j,k}^m \\
\frac{\Delta t}{\Delta r_f} &= R_f, \frac{\Delta t}{\Delta r_f^2} = R_{ff}, u_{i,j,k+1}^m - u_{i,j,k-1}^m = \delta_f * u_{i,j,k}^m, \\
u_{i,j,k+1}^m - 2u_{i,j,k}^m + u_{i,j,k-1}^m &= \delta_{ff} * u_{i,j,k}^m \\
\frac{\Delta t}{\Delta r_d \Delta s} &= R_{sd}, \frac{\Delta t}{\Delta r_f \Delta s} = R_{sf}, \frac{\Delta t}{\Delta r_d \Delta f} = R_{df} \\
u_{i+1,j+1,k}^m + u_{i-1,j-1,k}^m - u_{i-1,j+1,k}^m - u_{i+1,j-1,k}^m &= \delta_{sd} u_{i,j,k}^m \\
u_{i+1,j,k+1}^m + u_{i-1,j,k-1}^m - u_{i-1,j,k+1}^m - u_{i+1,j,k-1}^m &= \delta_{sf} u_{i,j,k}^m \\
u_{i,j+1,k+1}^m + u_{i,j-1,k-1}^m - u_{i,j+1,k-1}^m - u_{i,j-1,k+1}^m &= \delta_{df} u_{i,j,k}^m \\
A_1 &= \frac{(r_d - r_f)s}{2} R_s \delta_s + \frac{\gamma^2 s^2}{2} R_{ss} \delta_{ss} - \frac{1}{3} r_d \\
A_2 &= \frac{\theta_d - \kappa_d r_d}{2} R_d \delta_d + \frac{\sigma_d^2}{2} R_{dd} \delta_{dd} - \frac{1}{3} r_d \\
A_3 &= \frac{\theta_f - \kappa_f r_f - \rho_{s,f} \sigma_f \gamma}{2} R_f \delta_f + \frac{\sigma_f^2}{2} R_{ff} \delta_{ff} - \frac{1}{3} r_d \\
A_0 &= \frac{\rho_{s,d} \sigma_d \gamma s}{4} R_{sd} \delta_{sd} + \frac{\rho_{s,f} \sigma_f \gamma s}{4} R_{sf} \delta_{sf} + \frac{\rho_{d,f} \sigma_f \sigma_d}{4} R_{df} \delta_{df}
\end{aligned}$$

Represents A_1, A_2, A_3 in the tri-diagonal matrix form as:

$$\begin{aligned}
A_1 &= \begin{bmatrix} -\frac{\gamma^2 s^2}{\Delta s^2} - \frac{1}{3} r_d & \frac{(r_d - r_f)s}{2\Delta s} + \frac{\gamma^2 s^2}{2\Delta s^2} & \cdots & 0 \\ -\frac{(r_d - r_f)s}{2\Delta s} + \frac{\gamma^2 s^2}{2\Delta s^2} & \ddots & & \vdots \\ \vdots & \cdots & & -\frac{\gamma^2 s^2}{\Delta s^2} - \frac{1}{3} r_d \end{bmatrix} \\
A_2 &= \begin{bmatrix} -\frac{\sigma_d^2}{\Delta r_d^2} - \frac{1}{3} r_d & \frac{\theta_d - \kappa_d r_d}{2\Delta r_d} + \frac{\sigma_d^2}{2\Delta r_d^2} & \cdots & 0 \\ -\frac{\theta_d - \kappa_d r_d}{2\Delta r_d} + \frac{\sigma_d^2}{2\Delta r_d^2} & \ddots & & \vdots \\ \vdots & \cdots & & -\frac{\sigma_d^2}{\Delta r_d^2} - \frac{1}{3} r_d \end{bmatrix} \\
A_3 &= \begin{bmatrix} -\frac{\sigma_f^2}{\Delta r_f^2} - \frac{1}{3} r_d & \frac{\theta_f - \kappa_f r_f - \rho_{s,f} \sigma_f \gamma}{2\Delta r_f} + \frac{\sigma_f^2}{2\Delta r_f^2} & \cdots & 0 \\ -\frac{\theta_f - \kappa_f r_f - \rho_{s,f} \sigma_f \gamma}{2\Delta r_f} + \frac{\sigma_f^2}{2\Delta r_f^2} & \ddots & & \vdots \\ \vdots & \cdots & & -\frac{\sigma_f^2}{\Delta r_f^2} - \frac{1}{3} r_d \end{bmatrix}
\end{aligned}$$

Appendix B

Second Appendix

For the boundary conditions, approximations to the partial derivative have to be forward and backward second-order one-sided finite difference. More specifically, for the boundary on the first points, use the forward difference, for the last points, use the backward points Secondorder onesided approximation for the derivatives:

$$\begin{aligned}\frac{\partial u}{\partial s} &\approx \frac{4 * u_{i+1,j,k}^m - u_{i+2,j,k}^m - 3 * u_{i,j,k}^m}{2\Delta s} && \text{forward} \\ \frac{\partial u}{\partial s} &\approx \frac{u_{i-2,j,k}^m - 4 * u_{i-1,j,k}^m + 3 * u_{i,j,k}^m}{2\Delta s} && \text{backward} \\ \frac{\partial u}{\partial r_d} &\approx \frac{4 * u_{i,j+1,k}^m - u_{i,j+2,k}^m - 3 * u_{i,j,k}^m}{2\Delta r_d} && \text{forward} \\ \frac{\partial u}{\partial r_d} &\approx \frac{u_{i,j-2,k}^m - 4 * u_{i,j-1,k}^m + 3 * u_{i,j,k}^m}{2\Delta r_d} && \text{backward} \\ \frac{\partial u}{\partial r_f} &\approx \frac{4 * u_{i,j,k+1}^m - u_{i,j,k+2}^m - 3 * u_{i,j,k}^m}{2\Delta r_f} && \text{forward} \\ \frac{\partial u}{\partial r_f} &\approx \frac{u_{i,j,k-2}^m - 4 * u_{i,j,k-1}^m + 3 * u_{i,j,k}^m}{2\Delta r_f} && \text{backward}\end{aligned}$$

Appendix C

”Fixed notional” method

In our paper, we have been discussing the floating-to-floating cross-currency swap, this constructs a structure that the two parties receive the floating from one currency and pay the floating in another. The ”fixed notional” method can be used to approximate the cash flows from the floating payment receives. Take the domestic investor for example; the cash inflow for each time period is $v_\tau L_d(T_\tau)N_d$, where v_τ is the year fraction, and $L_d(T_\tau)$ is the domestic Libor rate, N_d is the principal. However, this cash inflow is equivalent to: getting N_d at $T_{\tau-1}$, and then invest it at rate L_d , at time T_τ , return N_d . The interest rate generated from this transaction will be exactly the same as $v_\tau L_d(T_\tau)N_d$. If we do the same for each period, will result in only the principals receives at T_{start} and pays at T_{end} are relevant, where the all cash flows between are cancelled out. Therefore, to be more specifically, only the discount factor from T_{start} to T_{end} is needed to estimate the cash flows on this side.

Appendix D

Codes in MATLAB

```
function f = TransformA0(u0, rhosd, rhosf, rhodf, sigmad, sigmaf, gamma, dt)
    clear f
    ns = size(u0, 1) - 1;
    nrd = size(u0, 2) - 1;
    nrf = size(u0, 3) - 1;
    sline = 0:1:ns;
    dline = 0:1:nrd;
    fline = 0:1:nrf;
    u1 = u0 (3:ns + 1, 3:nrd + 1, 2:nrf) ...
        + u0 (1:ns - 1, 1:nrd - 1, 2:nrf) ...
        - u0 (1:ns - 1, 3:nrd + 1, 2:nrf) ...
        - u0 (3:ns + 1, 1:nrd - 1, 2:nrf);
    u2 = u0 (3:ns + 1, 2:nrd, 3:nrf + 1) ...
        + u0 (1:ns - 1, 2:nrd, 1:nrf - 1) ...
        - u0 (1:ns - 1, 2:nrd, 3:nrf + 1) ...
        - u0 (3:ns + 1, 2:nrd, 1:nrf - 1);
    u3 = u0 (2:ns, 3:nrd + 1, 3:nrf + 1) ...
        + u0 (2:ns, 1:nrd - 1, 1:nrf - 1) ...
        - u0 (2:ns, 1:nrd - 1, 3:nrf + 1) ...
        - u0 (2:ns, 3:nrd + 1, 1:nrf - 1);
    u4 = zeros (ns + 1, nrd + 1, nrf + 1);
    u4(2:ns, 2:nrd, 2:nrf) = u1;
    u5 = zeros (ns + 1, nrd + 1, nrf + 1);
    u5(2:ns, 2:nrd, 2:nrf) = u2;
    u6 = zeros (ns + 1, nrd + 1, nrf + 1);
    u6(2:ns, 2:nrd, 2:nrf) = u3;
    sd = sline' * dline;
    sf = sline' * fline;
    df = dline' * fline;

    f1 = zeros (ns + 1, nrd + 1, nrf + 1);
    f2 = zeros (ns + 1, nrd + 1, nrf + 1);
    f3 = zeros (ns + 1, nrd + 1, nrf + 1);
    for i = 1 : ns + 1
        f1(:, :, i) = sd * u4(:, :, i);
        f2(:, :, i) = sf * u5(:, :, i);
        f3(:, :, i) = df * u6(:, :, i);
    end
    f11 = rhosd * sigmad * gamma * dt * f1 / 4;
    f22 = rhosf * sigmaf * gamma * dt * f2 / 4;
    f33 = rhodf * sigmaf * sigmad * dt * f3 / 4;
    f0 = f11 + f22 + f33;
    f = f0;
end
```

```

function f = TransformA1(u0,rd,rf,gamma,dt)
    clear f

    ns = size(u0, 1) - 1;
    nrd = size(u0, 2) - 1;
    nrf = size(u0, 3) - 1;
    sline = 1:ns-1;

    f = zeros (size (u0));
    B1 = sline.^2*gamma.^2/2;
    B1 = [B1';0;0];
    B2 = -sline.^2*gamma.^2;
    B2 = [0;B2';0];
    B3 = sline.^2*gamma.^2/2;
    B3 = [0;0;B3'];
    B = spdiags ([B1 B2 B3], [-1 0 1], ns + 1, ns + 1);

    C1 = - (rd - rf)*sline/2;
    C1 = [C1'; 0; 0];
    C2 = -rd*ones (ns - 1, 1)/3;
    C2 = [0;C2;0];
    C3 = (rd - rf)*sline/2;
    C3 = [0;0;C3'];
    C = spdiags ([C1 C2 C3], [-1 0 1], ns + 1, ns + 1);

    for j = 2 : nrd
        for k = 2 : nrf
            P1 = u0(:, j, k);
            G1 = B + C;
            temp1 = dt*G1*P1;
            f(2 : ns, j, k ) = temp1(2:ns);
        end
    end
end

```

```

function f = TransformA2(u0,rd,sigmad,thetad,kappad,dt)
    clear f
    ns = size(u0, 1) - 1;
    nrd = size(u0, 2) - 1;
    nrf = size(u0, 3) - 1;
    drd = rd/nrd;
    dline = 1:nrd - 1;
    f = zeros (size (u0));

    D1 = sigmad^2/(2*drd.^2) - thetad/(2*drd) + kappad*dline/2;
    D1 = [D1'; 0; 0];
    D2 = (-sigmad.^2/drd.^2 - rd/3)*ones (1, nrd - 1);
    D2 = [0;D2';0];
    D3 = sigmad.^2/(2*drd.^2) + thetad/(2*drd) - kappad*dline/2;
    D3 = [0;0;D3'];
    D = spdiags ([D1 D2 D3], [-1 0 1], nrd + 1, nrd + 1);

    for i = 2 : ns
        for k = 2 : nrf
            P2 = u0 (i, :, k)';
            temp2 = dt* D*P2;

            f (i, 2 : nrd, k) = temp2(2 : nrd);
        end
    end
end

```

```

function f = TransformA3(u0,rd,rf,sigmaf,rhosf,gamma,thetaf,kappaf,dt)
    clear f
    ns = size(u0, 1) - 1;
    nrd = size(u0, 2) - 1;
    nrf = size(u0, 3) - 1;
    drf = rf/nrf;
    fline = 1 : nrf - 1;
    f = zeros (size (u0));

    E1 = sigmaf.^2/(2*drf.^2)-thetaf/(2*drf)+kappaf*fline/2+rhosf*sigmaf*
        gamma/(2*drf);
    E1 = [E1'; 0; 0];
    E2 = (-sigmaf.^2/drf.^2 - rd/3)*ones (1, nrf - 1);
    E2 = [0;E2';0];
    E3 = sigmaf.^2/(2*drf.^2)+thetaf/(2*drf)-kappaf*fline/2-rhosf*sigmaf*
        gamma/(2*drf);
    E3 = [0;0;E3'];
    E = spdiags ([E1 E2 E3], [-1 0 1], nrf + 1, nrf + 1);

    for i = 2 : ns
        for j = 2 : nrd
            P3 = u0 (i, j, :);
            P3 = reshape (P3, [nrf + 1, 1]);
            temp3 = dt* E*P3;
            f (i, j, 2 : nrf) = temp3(2 : nrf);
        end
    end
end

```

```

function f = ThomasA1(u0,rd,rf,gamma,boundary1,boundary11,dt)
    clear f
    ns = size(u0, 1) - 1;
    nrd = size(u0, 2) - 1;
    nrf = size(u0, 3) - 1;
    iline = 1 : ns - 1;
    f = zeros (size (u0));
    for j = 2 : nrd
        for k = 2 : nrf
            stp = u0(2 : ns, j, k);
            a = (rd - rf)*iline*dt/2 - gamma.^2*iline.^2*dt/2;
            b = 1 + gamma^2*iline.^2*dt + rd*dt/3;
            c = - (rd - rf)*iline*dt/2 - gamma.^2*iline.^2*dt/2;
            stp(1) = stp(1) - a(1)*boundary1;
            stp(ns-1) = stp(ns-1)-c(ns-1)*boundary11;
            for l = 2 : ns - 1
                m = a (l)/b (l - 1);
                b (l) = b(l) - m*c(l - 1);
                stp (l) = stp(l) - m*stp(l - 1);
            end

            f(ns, j, k) = stp(ns - 1)/b(ns - 1);
            for l = ns - 2: -1: 1
                f(l + 1, j, k) = (stp(l) - c(l)*f(l + 1, j, k))/b(l);
            end
        end
    end
end

```

```

function f = ThomasA2(u0,rd ,drd ,sigmad ,thetad ,kappad ,boundary2 , boundary2
, dt)
clear f
ns = size(u0, 1) - 1;
nrd = size(u0, 2) - 1;
nrf = size(u0, 3) - 1;
jline = 1 : nrd - 1;
f = zeros (size (u0));

for i = 2 : ns
    for k = 2 : nrf
        stp = u0(i, 2 : nrd, k);
        a = thetad*dt/(2*drd)-kappad*jline*dt/2-sigmad^2*dt/(2*drd
^2);
        b = (1+sigmad^2*dt/drd^2+rd*dt/3)*ones(1 ,nrd-1);
        c = -thetad*dt/(2*drd)+kappad*jline*dt/2-sigmad^2*dt/(2*drd
^2);
        stp(1) = stp(1) - a(1)*boundary2;
        stp(nv-1) = stp(nv-1) - c(nv-1)*boundary22;
        for l = 2 : nrd - 1
            m = a(1)/b(1 - 1);
            b(1) = b(1) - m*c(1 - 1);
            stp(1) = stp(1) - m*stp(1 - 1);
        end
        f(i, nrd, k) = stp(nrd - 1)/b(nrd - 1);

        for l = ns - 2 : -1 : 1
            f(i, l + 1, k) = (stp(1) - c(1)*f(i, l + 1, k))/b(1);
        end
    end
end
end

function f = ThomasA3(u0,rd ,drf ,sigmaf ,thetaf ,kappaf ,rhosf , ...
gamma,boundary3 ,boundary33 , dt)
clear f
ns = size(u0, 1) - 1;
nrd = size(u0, 2) - 1;
nrf = size(u0, 3) - 1;
kline = 1 : nrf - 1;
f = zeros (size (u0));
for i = 2 : ns
    for j = 2 : nrd
        stp = u0(i, j, 2 : nrf);
        a =(thetaf*dt/(2*drf)-kappaf*kline*dt/2-rhosf*sigmaf*gamma)*dt
/(2*drf)-sigmaf^2*dt/(2*drf^2);
        b=(1+ sigmaf^2*dt/drf^2 + rd*dt/3)*ones(1 ,nrf - 1);
        c=thetaf*dt/ (2*drf)-kappaf*kline*dt/2-rhosf*sigmaf*gamma*dt
/(2*drf)-sigmaf^2*dt/(2*drf^2);
        stp(1) = stp(1) - a(1)*boundary3;
        stp(nv-1) = stp(nv-1) - c(nv-1)*boundary33;
        for l = 2 : nrf - 1
            m = a(1)/b(1 - 1);
            b(1) = b(1) - m*c(1 - 1);
            stp(1) = stp(1) - m*stp(1 - 1);
        end
        f(i, j, nrf) = stp(nrf - 1)/b(nrf - 1);
        for l = ns - 2 : -1 : 1
            f(i, j, l + 1) = (stp(1) - c(1)*f(i, j, l + 1))/b(1);
        end
    end
end
end
end

```

```

function f = Hund (thetad, thetaf, kappad, kappaf, rhosf, rhosd, rhodf,...
    sigmad, sigmaf, gamma, s, ns, rd, nrd, rf, nrf, T, nt, Dd, Df, Nd,
    boundary1, boundary11, boundary2, boundary22, boundary3,
    boundary33)
clear f;
dt      = T/nt;
ds      = s/ns;
drd     = rd/nrd;
drf     = rf/nrf;
svalue  = 0: ds :s;
c       = (1-Dd)*svalue-1 +Df;
q       = ones (ns + 1, nrd + 1, nrf + 1);
b       = ones (1,nrd+1,nrf+1);
for i = 1 : nrf + 1
    q (:,:,i) = c'*b(:,:,i);
end
        u0 = Nd*q;
for m = 1 : nt
    % first phase
    v0 =u0 + TransformA0(u0,rhosd,rhosf,rhodf,sigmad,sigmaf,gamma, dt)...
        + TransformA1(u0, rd, rf, gamma, dt) ...
        + TransformA2(u0, rd, sigmad, thetad, kappad,
            dt)...
        + TransformA3(u0, rd, rf, sigmaf, rhosf, gamma,
            ... thetaf, kappaf, dt);
    W1 = v0 - 0.5*TransformA1(u0, rd, rf, gamma, dt);
    v1 = ThomasA1(W1, rd, rf, gamma, boundary1, boundary11, dt);
    W2 = v1 - 0.5*TransformA2(u0, rd, sigmad, thetad, kappad,
        dt);
    v2 = ThomasA2(W2,rd, drd, sigmad, thetad, kappad, boundary2, ... boundary22,
        dt);
    W3 = v2-0.5*TransformA3(u0, rd, rf, sigmaf, rhosf, gamma, thetaf, ...
        kappaf, dt);
    v3 = ThomasA3(W3,rd, drf, sigmaf, thetaf, kappaf, rhosf, gamma, ...
        boundary3, boundary33, dt);
    % second phase
    v_0 =v0+0.5*(TransformA0(W3,rhosd,rhosf,rhodf,sigmad,sigmaf,gamma, dt)
        ...
        + TransformA1(W3, rd, rf, gamma, dt) ...
        + TransformA2(W3, rd, sigmad, thetad, kappad, dt) ...
        + TransformA3(W3, rd, rf, sigmaf, rhosf, gamma, thetaf, kappaf, dt)
        ) ...
        - 0.5*(TransformA0(u0,rhosd,rhosf,rhodf, sigmad,sigmaf,gamma, dt)
        ...
        + TransformA1(u0, rd, rf, gamma, dt) ...
        + TransformA2(u0, rd, sigmad, thetad, kappad, dt) ...
        + TransformA3(u0, rd, rf, sigmaf, rhosf, gamma, thetaf, kappaf, dt
        ));
    Z1 = v_0 - 0.5*TransformA1(v3, rd, rf, gamma, dt);
    v_1 = ThomasA1(Z1, rd, rf, gamma, boundary1, boundary11, dt);
    Z2 = v_1 - 0.5*TransformA2(v3, rd, sigmad, thetad, kappad,
        dt);
    v_2 = ThomasA2(Z2, rd, drd, sigmad, thetad, kappad, boundary2, ...
        boundary22, dt);
    Z3 = v_2 -0.5*TransformA3(v3, rd, rf, sigmaf, rhosf, gamma, thetaf,
        ... kappaf, dt);
    v_3 = ThomasA3(Z3, rd, drf, sigmaf, thetaf, kappaf, rhosf, gamma, ...
        boundary3, boundary33, dt);
    u0 = v_3;
end
    f = u0;
end

```