2023-08

A Compact ADI Finite Difference Method for 2D Reaction-Diffusion Equations with Variable Diffusion Coefficients

He, Mingyu

UNIVERSITY OF CALGARY

A Compact ADI Finite Difference Method for 2D Reaction-Diffusion Equations with Variable Diffusion Coefficients

by

Mingyu He

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN MATHEMATICS AND STATISTICS

CALGARY, ALBERTA

AUGUST, 2023

Abstract

Reaction-diffusion systems on a spatially heterogeneous domain have been widely used to model various biological applications. However, it is rarely possible to solve such partial differential equations (PDEs) analytically. Therefore, efficient and accurate numerical methods for solving such PDEs are desired. In this paper, we apply the well-known Padé approximation-based operator splitting (ADI) scheme. The new scheme is compact and fourth-order accurate in space. Combined with the Richardson extrapolation, the method can be improved to fourth-order accurate in time. Stability analysis shows that the method is unconditionally stable; thus, a large time step can be used to improve the overall computational efficiency further, Numerical examples have also demonstrated the new scheme's high efficiency and high-order accuracy.

Preface

The research conducted for this thesis is the joint work by the author and Dr. Wenyuan Liao. Part of the material has been presented at the the 2nd International Conference on Computational Method and Applications in Engineering (May 7-8, 2022, Starkville, Mississippi) and Western Canada Math Biology Spring Workshop (May 13-15, 2022, Kelowna, British Columbia).

Acknowledgements

This thesis would not be possible without the help and guidance from Prof. Wenyuan Liao, to whom I am especially grateful.

I am also deeply indebted to all my family members, who have been generously supporting me in numerous ways and encouraging me to pursue my dreams.

I have met lots of amazing people during my studies at the University of Calgary. I sincerely thank them for their help, kindness, friendship, as well as the imperishable memories, and wish them happiness and success in the future.

To Garfield

Table of Contents

List of Figures and Illustrations

List of Tables

List of Symbols, Abbreviations and Nomenclature

Symbol or abbreviation	Definition
d	dimension of the spatial domain
δ_x^2	the central difference operator
h	distance between two neighbour nodes in spatial grid
$K(x, y)$	carrying capacity at (x, y)
$k(x, y)$	critical population density at (x, y)
M	number of time steps
N	number of nodes in each spatial direction (excluding the origin)
Ω	the spatial domain
p	number of equations in an equation system
r	intrinsic growth rate of a population
$\mathbb{R}$	the set of all real numbers
$s_{i,j}^n$	evaluation of the source term at spatial grid point (i, j) and n-th time step
T	the time point for which the equation is solved
τ	distance between each temporal step
tol	error tolerance during iteration
$U_{i,j}^n$	evaluation of the real solution at spatial grid point (i, j) and n-th time step
$s_{i,j}^n$	numerical solution at spatial grid point (i, j) and n-th time step

Chapter 1

Introduction

In this thesis, we consider the following reaction-diffusion equation system in 2D space:

$$u_t = D_1(x, y)u_{xx} + D_2(x, y)u_{yy} + s(u, x, y, t), (x, y) \in \Omega, t > 0 \tag{1.1}$$

where $D_1(x, y)$ and $D_2(x, y)$ are diagonal matrices of dimensions $p \times p$ with positive entries, which describes the diffusion properties of the system, while the source term $s(u, x, y, t) \in \mathbb{R}^p$ is a vector function with describes the reaction part of the system. The function $u : \mathbb{R}^p \to \mathbb{R}^p$ is the unknown vector function to be solved. Ω is a rectangle in d-dimensional domain. In this thesis, we focus on the cases where $d = 2$ and $p = 1, 2$. However, our algorithm can effortlessly be generated to cases of higher p.

The reaction-diffusion system defined in Eq(1.1) is a crucial type of partial differential equations (PDEs) that arise frequently in science and engineering. Illustrations of applications of reaction-diffusion equations in physics, chemistry, pollution modelling, population biology and related fields can be found in various research work [4, 5, 21, 27, 29].

For example, researchers studying population biology are particularly interested in reaction-diffusion equations, for they are able to capture the reproduction, competition, and migration of population, as well as interaction between species if we consider systems of reaction-diffusion equations. Both mathematical properties and its biological implications of such

models have been thoroughly studied by researchers. See [27] and the references therein. Using different forms of reaction-diffusion equations enables ecologists to describe different models based on real ecological conditions, subsequently helping them make informed decisions on real-world problems, such as pest control or biodiversity preservation. Eq(1.2) is an example of reaction-diffusion equation that depicts the dynamics of population on a hypothetical square habitat $[0,1]^2$

$$u_t = a(x,y)u_{xx} + b(x,y)u_{yy} + ru(u - k(x,y))(K(x,y) - u) \tag{1.2}$$

In this equation, $a(x,y) \geqslant a > 0$, $b(x,y) \geqslant b > 0$ for some constants $a, b > 0$, $r > 0$. $K(x,y) > k(x,y) > 0$. We note that the diffusion coefficient functions $a(x,y)$ and $b(x,y)$ are both strictly positive, meaning that the individuals in the population tend to migrate from more populated area to area that is less so, a wise strategy to reduce intra-competition inside the species in order to enlarge the population as a whole. Note that the speed of diffusion of the population in each direction might not be identical due to potential terrain, climate, or resource constraints, so $a(x,y)$ and $b(x,y)$ might not be constant functions. Mathematically speaking, the values of $a(x,y)$ and $b(x,y)$ are larger in places that are less agreeable and individuals are more motivated to diffuse. $K(x,y)$ is the carrying capacity of the population at the place (x,y), and $k(x,y)$ is the critical point for sustainable reproduction. A population at (x,y) greater than $K(x,y)$ means an over-concentration of population at (x,y), leading to a decline in population because of the scarcity of resources and intra-competition within species. On the other hand, a population of u at (x,y) lower than $k(x,y)$ will lead the species to go extinct as the population is smaller than the threshold for sustainable reproduction. It worths noting that this does not imply the population will cease exiting at (x,y) in the long run, as the diffusion terms always try to refill the spots where the population density is low. This equation exhibits *strong Allee effect*, a phenomenon that has been observed in numerous real-world population dynamics. One can readily add more terms to Eq(1.2) in

order to capture more detailed and realistic features in a real-world population, but one will quickly reach a state where an analytical solution ceases to exist. Fortunately, as long as such modifications are only made to the source term, the algorithm presented in this thesis is able to efficiently provide a 4th order accurate numerical solution to the modified model.

Mathematicians have long been working on solving reaction-diffusion equations both analytically and numerically. It is universally known that analytic solution to a reaction-diffusion equation can be challenging to obtain when the source term becomes sophisticated. Therefore, efficient and accurate numerical algorithms to solving reaction-diffusion equations are often desired by researchers, and there has already been a considerable amount of literature on such algorithms.

Among these numerical algorithms, the finite difference method is a popular choice for solving the reaction-diffusion equation, due to its easy implementation and high efficiency and accuracy. It discretizes both the spatial and temporal domain and transforms the problem of solving for a continuous solution on the domain into a problem of successively solving linear systems of finite dimension. In general, explicit and implicit finite difference methods can be employed to solve the time time-dependent PDEs. The explicit method is efficient in each temporal step but prone to stability issues. Hence, only a small time step can be used for time marching, which greatly reduces the overall computational efficiency. The implicit method, on the other hand, is less efficient during each temporal step as a large-size algebraic system needs to be solved in each time step. More specifically, for a d-dimensional problem that has $N+1$ grid points in each direction, the linear system we need to solve at each temporal step is $(N-1)^d$-dimensional. Note that the computational workload of solving a linear system depends heavily on the dimension of the system and such relationship can be cubic at worst, which makes the regular implicit finite difference method unbearably slow in high-dimensional cases. The computational cost is even higher if the algebraic system is nonlinear (when the source term $s(u,x,y,t)$ in Eq(1.1) is nonlinear in u). However, implicit methods may still outperform explicit methods in terms of overall computational cost, as it

is more stable and permits a larger time step.

In the application of numerical algorithms to scientific and engineering problems, accuracy is usually the most fundamental concern. To improve the order of accuracy of a finite difference method, more grid points are generally required for spatial derivatives approximation, which normally will result in a larger stencil [17]. This will significantly increase the computational workload and cause difficulty in handling boundary conditions. One possible solution is to use compact finite difference algorithms. The first method is the so-called combined compact scheme, in which the spatial derivatives are computed through the solution of a linear system relating the unknown function u to its spatial derivatives [7, 29]. The shortcoming of this method is that some extra boundary conditions for the spatial derivatives are needed, which, unfortunately, are not available in general. The second method is to use Padé approximations to improve the conventional second-order central finite difference operator to a compact fourth-order finite difference operator [1, 19, 25]. The advantage of this method is that no extra boundary conditions are required for the spatial derivatives.

Compact finite difference algorithms have accrued increasing popularity due to their higher accuracy and efficiency in solving various PDEs. In the past decades, many researchers were devoted to developing compact finite difference methods for solving wave equations [9, 26], reaction-diffusion equations [2, 18, 34, 35, 36], delayed reaction-diffusion equations [10, 40, 37], parabolic or heat equations [6, 8, 28, 23, 33, 39], convection-diffusion equations [16], Burger's equation [38] and stochastic PDE [13], only to name a few. Being implicit schemes and normally involving the solution of large-scale linear systems, compact finite difference schemes provide high-order accuracy and efficiency at the cost of computational workload. To resolve this issue, some novel technique is needed.

Alternating Directional Implicit (ADI) scheme, first introduced by Peaceman and Rachford [30], can be applied to address this issue. In particular, an operator-splitting type method is used to decompose the original coupled algebraic system into two separate sets of equations, with one along each of x and y directions, so that each decoupled system can

be solved separately with high efficiency [11, 12, 31]. In Eq(1.1), this means solving the algebraic system on a d-dimensional domain can be reduced to solving a sequence of one-dimensional problems. This widely-celebrated technique greatly expedites our algorithm by reducing the workload from $\mathcal{O}(N^{3d})$ to $\mathcal{O}(N^d)$, and is very useful when N or d is large. A review of relevant math background of ADI schemes has been included in Appendix A.

Recently, a lot of efforts have been made to reduce computational complexity by using the ADI technique to solve a variety of time-dependent PDEs. One early work was done by Fairweather and Mitchell [14], who employed the ADI technique to solve wave equations. In [25], the authors developed an efficient fourth-order algorithm based on Padé approximation and ADI technique. The algorithm is compact and fourth-order accurate in both temporal and spatial dimensions and only requires a compact stencil as used by the standard Crank-Nicolson algorithm. Later, this method was extended to 3D reaction-diffusion equations with Dirichlet boundary conditions when diffusion coefficients are constant [18]. In recent years, the demand for a fast algorithm with high-order accuracy for equations of more general forms arose rapidly. Thus, there has emerged some work on the combination of higher order derivative approximation (e.g. Padé approximation), variable coefficients, and ADI technique: Sun [32] studied a high-order algorithm for 1D reaction-diffusion equation, which was 4th-order in space and 2nd-order in time. Deng [10] proposed a 4th-order ADI scheme to solve nonlinear delay reaction-diffusion equations. Liao [24] developed a 4th-order compact ADI scheme for acoustic equation. Li *et al.* [22] extended Liao's work and generalized the algorithm to accommodate variable coefficients through a novel algebraic manipulation of the resulting linear system. It is worth mentioning that the compact ADI scheme for acoustic wave equation os a three-level method, in which the so-called combined compact scheme explicitly calculates the approximation of spatial derivatives. Such treatment is efficient, but cannot be directly applied to the reaction-diffusion equation.

Therefore, in this thesis, we will modify the method and combine it with the Padé approximation-based compact ADI method in [25] to present a new algorithm that is fourth-

order accurate in space and second-order accurate in time and that solves a reaction-diffusion equation with variable diffusion coefficients. This work not only inherits from, but also makes an important improvement to, [25], as non-constant diffusion coefficients frequently appear in scientific research.

In this thesis, I intend to present a FDTD-ADI algorithm for reaction-diffusion equations with variable coefficients. The rest of the paper is organized as follows. The algorithms will be presented in Chapter 2. Chapter 3 will be devoted to stability analysis. Chapter 4 features 5 numerical examples to test the accuracy of our algorithm and 1 numerical example to demonstrate how the algorithm can be applied to an equation picked from the real world. Chapter 5 includes the conclusion of this paper and some final remarks made by the author.

Chapter 2

Method

Consider the following IVP on $[0, T] \times [x_0, x_1] \times [y_0, y_1]$:

$$\begin{cases} u_t = a(x,y)u_{xx} + b(x,y)u_{yy} + s(u,x,y,t) \\ a(x,y) \geqslant a_0 > 0 \quad b(x,y) \geqslant b_0 > 0 \\ u(0,x,y) = f(x,y) \\ u(t,x,y_0) = g_1(t,x) \quad u(t,x_0,y) = h_1(t,y) \\ u(t,x,y_1) = g_2(t,x) \quad u(t,x_1,y) = h_2(t,y) \end{cases} \tag{2.1}$$

where $s(u,t,x,y)$ represents a source term.

The domain is uniformly divided into an $(N_1+1) \times (N_2+1)$ grid in space and $M+1$ discrete time steps. Hence, $h_x = \frac{x_1 - x_0}{N_1}$, $h_y = \frac{y_1 - y_0}{N_2}$, $\tau = \frac{T}{M}$. Fig 2.1 is an illustrative diagram for the discretization. We denote $u_{i,j}^n$ the numerical solution at the space grid point with coordinate (i,j) and at the n-th time step.

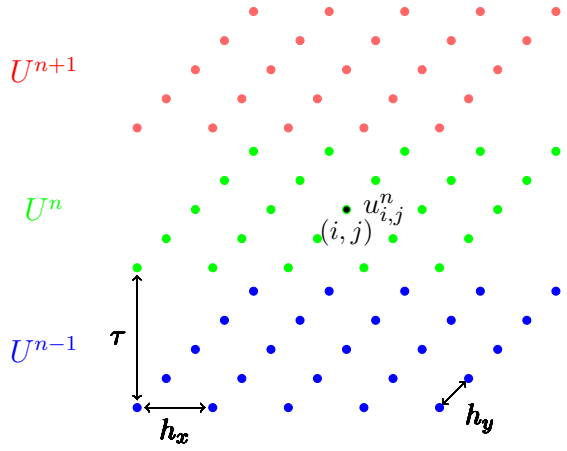


Figure 2.1: Discretization of the time and space

For simplicity, we assume that $h_x = h_y = h$ thereafter. We use the following Taylor's approximation:

$$u_{i-1,j} = u_{i,j} - (u_x)_{i,j} \cdot h + \frac{(u_{xx})_{i,j}}{2} \cdot h^2 - \frac{(u_{xxx})_{i,j}}{6} \cdot h^3 + \frac{(u_{xxxx})_{i,j}}{24} \cdot h^4 - \frac{(u_{xxxxx})_{i,j}}{120} \cdot h^5 + \mathcal{O}(h^6) \tag{2.2a}$$

$$u_{i+1,j} = u_{i,j} + (u_x)_{i,j} \cdot h + \frac{(u_{xx})_{i,j}}{2} \cdot h^2 + \frac{(u_{xxx})_{i,j}}{6} \cdot h^3 + \frac{(u_{xxxx})_{i,j}}{24} \cdot h^4 + \frac{(u_{xxxxx})_{i,j}}{120} \cdot h^5 + \mathcal{O}(h^6) \tag{2.2b}$$

$$u_{i,j-1} = u_{i,j} - (u_y)_{i,j} \cdot h + \frac{(u_{yy})_{i,j}}{2} \cdot h^2 - \frac{(u_{yyy})_{i,j}}{6} \cdot h^3 + \frac{(u_{yyyy})_{i,j}}{24} \cdot h^4 - \frac{(u_{yyyyy})_{i,j}}{120} \cdot h^5 + \mathcal{O}(h^6) \tag{2.2c}$$

$$u_{i,j+1} = u_{i,j} + (u_y)_{i,j} \cdot h + \frac{(u_{yy})_{i,j}}{2} \cdot h^2 + \frac{(u_{yyy})_{i,j}}{6} \cdot h^3 + \frac{(u_{yyyy})_{i,j}}{24} \cdot h^4 + \frac{(u_{yyyyy})_{i,j}}{120} \cdot h^5 + \mathcal{O}(h^6) \tag{2.2d}$$

By combining Eq(2.2a) and Eq(2.2b), one can get

$$\frac{\delta_x^2}{h^2} u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} - 2u_{i,j}}{h^2} = (u_{xx})_{i,j} + \mathcal{O}(h^2) \tag{2.3a}$$

Similarly, combining Eq(2.2c) and Eq(2.2d) will yield that

$$\frac{\delta_y^2}{h^2}u_{i,j} = \frac{u_{i,j+1} + u_{i,j-1} - 2u_{i,j}}{h^2} = (u_{yy})_{i,j} + \mathcal{O}(h^2) \tag{2.3b}$$

In fact, Eq(2.3a) and Eq(2.3b) are known as the central difference scheme and are widely used in numerical analysis. Clearly, the central difference scheme provides a second-order approximation to spatial derivatives u_{xx} and u_{yy}. In order to upgrade the approximation to 4th order, the Padé approximation need to be used instead:

$$\frac{1}{h^2}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j} = u_{xx} + \mathcal{O}(h^4) \qquad \frac{1}{h^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j} = u_{yy} + \mathcal{O}(h^4) \tag{2.4}$$

Eq(2.4) can also be derived from Taylor's approximation. To wit, consider Taylor's approximation at 2 additional points:

$$\begin{aligned} u_{i-2,j} = u_{i,j} - (u_x)_{i,j}\cdot 2h + \frac{(u_{xx})_{i,j}}{2}\cdot(2h)^2 - \frac{(u_{xxx})_{i,j}}{6}\cdot(2h)^3 \\ + \frac{(u_{xxxx})_{i,j}}{24}\cdot(2h)^4 - \frac{(u_{xxxxx})_{i,j}}{120}\cdot(2h)^5 + \mathcal{O}(h^6) \end{aligned} \tag{2.2e}$$

$$\begin{aligned} u_{i+2,j} = u_{i,j} + (u_x)_{i,j}\cdot 2h + \frac{(u_{xx})_{i,j}}{2}\cdot(2h)^2 + \frac{(u_{xxx})_{i,j}}{6}\cdot(2h)^3 \\ + \frac{(u_{xxxx})_{i,j}}{24}\cdot(2h)^4 + \frac{(u_{xxxxx})_{i,j}}{120}\cdot(2h)^5 + \mathcal{O}(h^6) \end{aligned} \tag{2.2f}$$

By combining Eq(2.2a), Eq(2.2b), Eq(2.2e) and Eq(2.2f), we obtain the following result:

$$\begin{aligned} (u_{xx})_{i,j} + \mathcal{O}(h^4) =& \frac{1}{h^2}\left[-\frac{1}{12}u_{i-2,j} + \frac{4}{3}u_{i-1,j} - \frac{5}{2}u_{i,j} + \frac{4}{3}u_{i+1,j} - \frac{1}{12}u_{i+2,j}\right] \\ =& \frac{1}{h^2}\Bigg[u_{i-1,j} + u_{i+1,j} - 2u_{i,j} - \frac{1}{12}\left(u_{i-2,j} + u_{i,j} - 2u_{i-1,j}\right) \\ & - \frac{1}{12}\left(u_{i,j} + u_{i+2,j} - 2u_{i+1,j}\right) + \frac{2}{12}\left(u_{i-1,j} + u_{i+1,j} - 2u_{i,j}\right)\Bigg] \\ =& \frac{1}{h^2}\left(\delta_x^2 - \frac{1}{12}\delta_x^4\right)u_{i,j} \qquad\qquad (\delta_x^4 = \delta_x^2\delta_x^2) \end{aligned} \tag{2.5}$$

We use the geometric series:

$$\begin{aligned}\frac{1}{h^2}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j} &= \frac{1}{h^2}\left(\delta_x^2 - \frac{1}{12}\delta_x^4 + \frac{1}{144}\delta_x^6 + ...\right)u_{i,j} \\ &= \frac{1}{h^2}\left(\delta_x^2 - \frac{1}{12}\delta_x^4\right)u_{i,j} + \frac{1}{h^2}\left(\frac{1}{144}\delta_x^6 + ...\right)u_{i,j} \end{aligned} \tag{2.6}$$

By Eq(2.3a), we can compute that

$$\delta_x^6 u_{i,j} = \delta_x^2\delta_x^2(\delta_x^2 u_{i,j}) = \delta_x^2\delta_x^2[h^2(u_{xx})_{i,j}+\mathcal{O}(h^4)] = \delta_x^2[h^4(u_{xxxx})_{i,j}+\mathcal{O}(h^6)] = h^6(u_{xxxxxx})_{i,j}+\mathcal{O}(h^8)$$

Therefore, due to the result above and Eq(2.5), Eq(2.6) reduces to

$$\frac{1}{h^2}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j} = (u_{xx})_{i,j} + \mathcal{O}(h^4),$$

which proves the first equation of Eq(2.4). The second equation can be proved in the same way but using the Taylor's approximation in y-direction. So, in addition, we have

$$\frac{1}{h^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j} = (u_{yy})_{i,j} + \mathcal{O}(h^4).$$

Now, using Eq(2.4) and applying Crank-Nicolson algorithm to the reaction-diffusion equation Eq(1.1), we get the algorithm presented in subsection 2.1, which will be further explored for each type of source terms.

2.1 u-independent Source Term

Consider a reaction-diffusion equation that has a source term independent of u:

$$u_t = a(x,y)u_{xx} + b(x,y)u_{yy} + s(x,y,t) \tag{2.7}$$

Using the Crank-Nicolson Algorithm and the approximation in Eq(2.4) above, we have:

$$\begin{aligned}\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\tau} &= \frac{a_{i,j}}{2h^2}\left(\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n}+\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n+1}\right)\\ &+\frac{b_{i,j}}{2h^2}\left(\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n}+\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n+1}\right)+\frac{1}{2}(s_{i,j}^{n}+s_{i,j}^{n+1})\end{aligned}$$

In particular, the Crank-Nicolson Algorithm ensures the second-order accuracy in time and Eq(2.4) is a fourth-order accurate approximation for spatial derivatives. For notational simplicity, let $\alpha_{i,j}=a_{i,j}\frac{\tau}{2h^2}$, $\beta_{i,j}=b_{i,j}\frac{\tau}{2h^2}$, then the equation above reduces to:

$$\begin{aligned}u_{i,j}^{n+1}-u_{i,j}^{n} &= \alpha_{i,j}\left(\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n}+\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n+1}\right)\\ &+\beta_{i,j}\left(\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n}+\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n+1}\right)+\frac{\tau}{2}(s_{i,j}^{n}+s_{i,j}^{n+1})\end{aligned}$$

We rearrange the equation above so that it has the form:

$$\begin{aligned}&u_{i,j}^{n+1}-\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n+1}-\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n+1}\\ =\;& u_{i,j}^{n}+\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n}+\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n}+\frac{\tau}{2}(s_{i,j}^{n}+s_{i,j}^{n+1})\end{aligned}\tag{2.8}$$

We note that the right-hand side of Eq(2.8) only consists of numerical solution to the IVP Eq(2.1) at time $t=n\tau$, which we have already known. The left-hand side of Eq(2.8) consists of the numerical solution at time $t=(n+1)*\tau$, which are the values that we need to solve at this stage. In order to apply ADI to Eq(2.8), a cross term is needed, namely,

$$\begin{aligned}&u_{i,j}^{n+1}-\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n+1}-\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n+1}+\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\left(\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n+1}\right)\\ =&u_{i,j}^{n}+\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n}+\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n}+\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\left(\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n}\right)\\ &+\frac{\tau}{2}(s_{i,j}^{n}+s_{i,j}^{n+1})+\mathcal{O}(h^4)\end{aligned}\tag{2.9}$$

The benefit of introducing cross terms is that Eq(2.9) can be now written in a neat form. That is,

$$\begin{aligned}&\left(1-\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1-\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n+1}\\=&\left(1+\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1+\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n}+\frac{\tau}{2}(s_{i,j}^n+s_{i,j}^{n+1})\end{aligned}\tag{2.10}$$

Eq(2.10) can be solved by ADI technique. More specifically, the equation can be solved at 2 steps and at each step we only need to solve in 1 direction. Mathematically, the following two equations will be solved

$$\begin{cases}\left(1-\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)u_{i,j}^{\star}=\left(1+\alpha_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1+\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n}\\\qquad\qquad\qquad\qquad+\frac{\tau}{2}(s_{i,j}^n+s_{i,j}^{n+1})\\\left(1-\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n+1}=u_{i,j}^{\star}\end{cases}\tag{2.11}$$

It is obvious that Eq(2.10) and Eq(2.11) are actually equivalent. Practically, the second equation of Eq(2.11) is easier to solve. To solve it, we firstly divide both sides by $\beta_{i,j}$, and then multiply both sides by $1+\frac{1}{12}\delta_y^2$, which gives us

$$\left(1+\frac{\delta_y^2}{12}\right)\frac{u_{i,j}^{n+1}}{\beta_{i,j}}-\delta_y^2u_{i,j}^{n+1}=\left(1+\frac{\delta_y^2}{12}\right)u_{i,j}^{\star}$$

or, equivalently,

$$\begin{aligned}&\left(\frac{1}{12\beta_{i,j-1}}-1\right)u_{i,j-1}^{n+1}+\left(\frac{10}{12\beta_{i,j}}+2\right)u_{i,j}^{n+1}+\left(\frac{1}{12\beta_{i,j+1}}-1\right)u_{i,j+1}^{n+1}\\&\qquad\qquad=\frac{u_{i,j-1}^{\star}}{12}+\frac{10u_{i,j}^{\star}}{12}+\frac{u_{i,j+1}^{\star}}{12}\end{aligned}\tag{2.12}$$

Each $u_{i,j}^*$ comes from solving the first equation of System (2.11), the solving process of which will be discussed later. Therefore, the right-hand side of Eq(2.12) can be easily computed.

Combining the Eq(2.12) for each j will result in a tridiagonal linear system for $u_{i,j}^{n+1}$'s, and such system can be efficiently solved by the Thomas algorithm. Next, we observe that $u_{i,0}^{n+1}$ and $u_{i,N}^{n+1}$ are on the boundary $y = x_0$ and $y = x_1$, thus the values of these points can be directly obtained from the boundary conditions in the IVP(2.1). Namely,

$$u_{i,0}^{n+1} = g_1((n+1)\cdot\tau, i\cdot h) \qquad u_{i,N+1}^{n+1} = g_2((n+1)\cdot\tau, i\cdot h)$$

As a result, we only need to solve for $u_{i,j}^{n+1}$ for $1 \leqslant j \leqslant N-1$.

To solve for all values of $u_{i,j}^{n+1}$, $1 \leqslant j \leqslant N-1$, we need the $u_{i,j}^{*}$ for all $0 \leqslant j \leqslant N$, which come from solving the first equation in the System(2.11). Moreover, the values of $u_{0,j}$ and $u_{N,j}$ can be obtained from the boundary conditions, so we do not need to solve the second equation in System(2.11) for $i = 0, N$. Hence, we only need $u_{i,j}^{*}$ for $1 \leqslant i \leqslant N-1$, $0 \leqslant j \leqslant N$.

We solve the first equation in System(2.11) using the same method as what we did for the second equation. After firstly dividing both sides by $\alpha_{i,j}$ and then multiplying both sides by $1 + \frac{1}{12}\delta_y^2$, we have:

$$\begin{aligned}\left(1+\frac{\delta_x^2}{12}\right)\frac{u_{i,j}^{n+1}}{\alpha_{i,j}} - \delta_x^2 u_{i,j}^{n+1} = &\left[\left(1+\frac{\delta_x^2}{12}\right)\frac{1}{\alpha_{i,j}} - \delta_x^2\right]\left(1-\beta_{i,j}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n} \\ &+ \frac{\tau}{2}\left(1+\frac{\delta_x^2}{12}\right)\left(\frac{s_{i,j}^{n}}{\alpha_{i,j}} + \frac{s_{i,j}^{n+1}}{\alpha_{i,j}}\right)\end{aligned} \tag{2.13}$$

To get rid of the δ_y^2 operator in the denominator on the right-hand side, we use the fact that

$$\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n} \approx \delta_y^2\left(1-\frac{\delta_y^2}{12}\right)u_{i,j}^{n} + \mathcal{O}(h^4) \tag{2.14}$$

Note that Eq(2.14) is the y-direction version of Eq(2.6), hence it has been proved before. Plugging Eq(2.14) into Eq(2.13) yields that

$$\begin{aligned}
&\left(1+\frac{\delta_x^2}{12}\right)\frac{u_{i,j}^{n+1}}{\alpha_{i,j}}-\delta_x^2 u_{i,j}^{n+1}\\
=&\left[\left(1+\frac{\delta_x^2}{12}\right)\frac{1}{\alpha_{i,j}}-\delta_x^2\right]\left(1-\beta_{i,j}\delta_y^2\left(1-\frac{\delta_y^2}{12}\right)\right)u_{i,j}^n\\
&\qquad\qquad+\frac{\tau}{2}\left(1+\frac{\delta_x^2}{12}\right)\left(\frac{s_{i,j}^n}{\alpha_{i,j}}+\frac{s_{i,j}^{n+1}}{\alpha_{i,j}}\right)\\
=&\left(1+\frac{\delta_x^2}{12}\right)\frac{u_{i,j}^n}{\alpha_{i,j}}-\delta_x^2 u_{i,j}^n-\left(1+\frac{\delta_x^2}{12}\right)\frac{\beta_{i,j}}{\alpha_{i,j}}\left[\delta_y^2\left(1-\frac{\delta_y^2}{12}\right)u_{i,j}^n\right]\\
&\qquad+\delta_x^2\left[\beta_{i,j}\delta_y^2\left(1-\frac{\delta_y^2}{12}\right)u_{i,j}^n\right]+\frac{\tau}{2}\left(1+\frac{\delta_x^2}{12}\right)\left(\frac{s_{i,j}^n}{\alpha_{i,j}}+\frac{s_{i,j}^{n+1}}{\alpha_{i,j}}\right)
\end{aligned}$$

The cross term $\delta_x^2\delta_y^2\delta_y^2 u_{i,j}^n$ that will appear after expanding the right-hand side of the last equation requires a $3*5$ stencil. Let U^* be a matrix formed by $u_{i,j}^*$. From our previous discussion, we can easily conclude that U^* is an $(N-1)*(N+1)$ matrix. Since the computation of each $u_{i,j}^*$ requires the values of $u_{i,j}^n$ on a $3*5$ stencil centred at $u_{i,j}^n$, the computation of U^* requires a $(N+1)*(N+5)$ matrix for U^n. In particular, in addition to the $u_{i,j}^n$ values in the domain, it also requires $u_{i,j}^n$ for $j=-2,-1,N+1,N+2$. These values are not available as they are outside the domain, but can be approximated using the extrapolation method. To ensure the 4th order accuracy of our method, the approximation needs to be at least 4th order accurate. For a satisfactory performance, we assume that $N\geqslant 5$ and take 5 points as the base of extrapolation. That is, we do:

$$\begin{aligned}
&u_{i,0}^n,\quad u_{i,1}^n,\quad u_{i,2}^n,\quad u_{i,3}^n,\quad u_{i,4}^n \xrightarrow{\text{Extrapolation}} \hat{u}_{i,-1}^n,\quad \hat{u}_{i,-2}^n\\
&u_{i,N-4}^n,\ u_{i,N-3}^n,\ u_{i,N-2}^n,\ u_{i,N-1}^n,\ u_{i,N}^n \xrightarrow{\text{Extrapolation}} \hat{u}_{i,N+1}^n,\ \hat{u}_{i,N+2}^n
\end{aligned}$$

and we use $\hat{u}^n_{i,j}$ in place of $u^n_{i,j}$ for $j = -2, -1, N+1, N+2$. Since the grid is uniform, the error analysis of extrapolation gives that

$$\left|u^n_{i,-1} - \hat{u}^n_{i,-1}\right| = \mathcal{O}(h^5) \qquad \left|u^n_{i,-2} - \hat{u}^n_{i,-2}\right| = \mathcal{O}(32h^5) \tag{2.15}$$

One thing to take caution here is that our algorithm is fourth-order accurate in space ($\mathcal{O}(h^4)$) and, if N is small and h is large, the second error term in Eq(2.15) could practically become the dominant error and sabotage the overall performance of the algorithm. Therefore, it is desirable that

$$N > 32 \cdot (y_1 - y_0)$$

In practice, for better performance, large N should be chosen so that the extrapolation here will not have destructive effect on the accuracy of our algorithm.

In short, the algorithm for solving IVP from time step n to $n+1$ can be summarized as follows:

Step 1. Using extrapolation on U^n to approximate $u_{i,j}$, $j = -2, -1, N+1, N+2$

Step 2. Solving for U^* using U^n and the first equation in System 2.11

Step 3. Solving for U^{n+1} using U^* and the second equation in System 2.11

2.2 u-dependent Linear Source Term

Consider the following equation whose source term is linear in u:

$$u_t = a(x,y)u_{xx} + b(x,y)u_{yy} + cu + s(x,y,t) \qquad c \in \mathbb{R}^* \tag{2.16}$$

In this subsection, we will show that one can transform Eq(2.16) into a new reaction-diffusion equation for which the preceding subsection can be applied. Firstly, one can move cu to the

left-hand side and multiply an integration factor on both sides. Specifically,

$$\begin{aligned} u_t - cu &= a(x,y)u_{xx} + b(x,y)u_{yy} + s(x,y,t) \\ e^{-ct}u_t - e^{-ct}cu &= a(x,y)e^{-ct}u_{xx} + b(x,y)e^{-ct}u_{yy} + e^{-ct}s(x,y,t) \\ (e^{-ct}u)_t &= a(x,y)(e^{-ct}u)_{xx} + b(x,y)(e^{-ct}u)_{yy} + e^{-ct}s(x,y,t) \end{aligned} \tag{2.17}$$

Note that in Eq(2.17), $e^{-ct}u_{xx} = (e^{-ct}u)_{xx}$ because e^{-ct} is independent of x. We let $w(x,t) = e^{-ct}u(x,t)$, $\tilde{s}(x,y,t) = e^{-ct}s(x,y,t)$, then Eq(2.17) can be rewritten as

$$w_t = a(x,y)w_{xx} + b(x,y)w_{yy} + \tilde{s}(x,y,t) \tag{2.18}$$

Note that our new source term is independent of w, so Eq(2.18) can be solved using the method discussed in subsection 2.1. Also note that the initial conditions and boundary conditions can be easily taken care of in this transformation. In particular,

$$\begin{aligned} w(0,x,y) &= (e^{-ct}u(t,x,y))\big|_{t=0} = u(0,x,y) \\ w(t,x,0) &= e^{-ct}u(t,x,0) = e^{-ct}g_1(t,x) \end{aligned}$$

All boundary conditions in Problem(2.1) can be dealt in the same way. Eventually, we may use $u(t,x,y) = e^{ct}w(t,x,y)$ to recover the solution of u.

2.3 Nonlinear Source Term

With a few modifications, the algorithm presented before can also accommodate equations with nonlinear source terms. Consider the IVP Eq(2.1) with the following Reaction-Diffusion Equation:

$$u_t = a(x,y)u_{xx} + b(x,y)u_{yy} + s(u,x,y,t) \tag{2.19}$$

After applying the ADI algorithm, we get the following system of equations:

$$\begin{cases} \left(1-\alpha_{i,j}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)u_{i,j}^{\star} = \left(1+\alpha_{i,j}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1+\beta_{i,j}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n} \\ \qquad\qquad +\dfrac{\tau}{2}(s_{i,j}^{n}(u_{i,j}^{n})+s_{i,j}^{n+1}(u_{i,j}^{n+1})) \\ \left(1-\beta_{i,j}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n+1} = u_{i,j}^{\star} \end{cases} \tag{2.20}$$

The difference between System(2.11) and System(2.20) only lies in the first equation. Namely, to compute the source term in System(2.20), we also need to know $u_{i,j}^{n+1}$, which is not known and is exactly what we are solving for. However, we can approximate $u_{i,j}^{n+1}$ and plug the approximation value into the left-hand side.

One way to approximate $u_{i,j}^{n+1}$ is to use the extrapolation method. At this time, $u_{i,j}^{n-1}$ and $u_{i,j}^{n}$ are already known. Hence, we do

$$u_{i,j}^{n-1},\ u_{i,j}^{n} \xrightarrow{\text{Extrapolation}} \overline{u}_{i,j}^{n+1}$$

We use $\overline{u}_{i,j}^{n+1}$ instead of the exact value $u_{i,j}^{n+1}$ on the left-hand side of the first equation in System(2.20), then we can proceed as before to solve for $u_{i,j}^{n+1}$. It is known that

$$|u_{i,j}^{n+1}-\overline{u}_{i,j}^{n+1}| = \mathcal{O}(\tau^2)$$

Since our algorithm is intended to be 2nd order in time, this extrapolation will not result in a deduction in the order of accuracy.

Another way to approximate the problem is to use the Prediction-Correction Method. To be specific, we use $u_{i,j}^{(0)} = u_{i,j}^{n}$ as an initial guess for $u_{i,j}^{n+1}$, solve for $u_{i,j}^{n+1}$ based on $u_{i,j}^{(0)}$, and denote the solution $u_{i,j}^{(1)}$. Now, we use $u_{i,j}^{(1)}$ as the new guess for $u_{i,j}^{n+1}$, and solve for the solution, which will be iteratively used as a new guess. The sequence $\{u_{i,j}^{(k)}\}_{k=0}^{\infty}$ will converge

to $u_{i,j}^{n+1}$. This process can be summarized by the following algorithm:

$0<\text{tol}\ll 1$

$U = \bar{U} = U^n$

while($\|\bar{U} - U\| < tol$ or the first time){

$\bar{U} = U$

Solve for U^{n+1} using $\bar{U}$, store the solution in U

}

$U^{n+1} = U$

2.4 System of Equations

With the two approximation techniques introduced above, we can solve for a system of equations. Consider the following IVP Eq(2.1) with two coupled equations:

$$\begin{cases} u_t = a(x,y)u_{xx} + b(x,y)u_{yy} + s_1(u,v,x,y,t) \\ v_t = c(x,y)v_{xx} + d(x,y)v_{yy} + s_2(u,v,x,y,t) \end{cases}$$

Using the ADI method we discussed before, this system decomposes into two system of equations:

$$\begin{cases} \left(1 - \alpha_{i,j}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right) u_{i,j}^{\star} = \left(1 + \alpha_{i,j}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1 + \beta_{i,j}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n} \\ \qquad\qquad + \dfrac{\tau}{2}(s_1{}_{i,j}^{n}(u_{i,j}^{n}, v_{i,j}^{n}) + s_1{}_{i,j}^{n+1}(u_{i,j}^{n+1}, v_{i,j}^{n+1})) \\ \left(1 - \beta_{i,j}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n+1} = u_{i,j}^{\star} \end{cases} \tag{2.21}$$

$$\begin{cases} \left(1-\alpha_{i,j}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)v_{i,j}^{\star} = \left(1+\alpha_{i,j}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1+\beta_{i,j}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)v_{i,j}^{n} \\ \qquad\qquad +\dfrac{\tau}{2}(s_{2i,j}^{n}(u_{i,j}^{n},v_{i,j}^{n})+s_{2i,j}^{n+1}(u_{i,j}^{n+1},v_{i,j}^{n+1})) \\ \left(1-\beta_{i,j}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)v_{i,j}^{n+1} = v_{i,j}^{\star} \end{cases} \tag{2.22}$$

Both $u_{i,j}^{n+1}$ and $v_{i,j}^{n+1}$ on the right-hand side of the first equations of System(2.21) and System(2.22) can be approximated by extrapolation based on the values at time step $n-1$ and n. On the other hand, we can also use the Prediction-Correction method with $u_{i,j}^{n}$ and $v_{i,j}^{n}$ as initial guesses for $u_{i,j}^{n+1}$ and $v_{i,j}^{n+1}$, and then iteratively and alternatively solve the System(2.21) and System(2.22) until the the predicted solutions converge to the real values.

2.5 Time-dependent Diffusion Coefficients

We can further generalize the algorithm so that the diffusion coefficients also vary temporally. Namely, consider the following equation:

$$u_t = a(x,y,t)u_{xx} + b(x,y,t)u_{yy} + s(u,x,y,t) \tag{2.23}$$

A standard discretization to Eq(2.23) at time $t = n\cdot\tau$ is

$$\begin{aligned} u_{i,j}^{n+1} - u_{i,j}^{n} = \frac{\tau}{2h^2}\Bigg(& a_{i,j}^{n}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n} + a_{i,j}^{n+1}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n+1} \\ & + b_{i,j}^{n}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n} + b_{i,j}^{n+1}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n+1}\Bigg) + \frac{\tau}{2}(s_{i,j}^{n}+s_{i,j}^{n+1}), \end{aligned} \tag{2.24}$$

where $a_{i,j}^{n}$ and $b_{i,j}^{n}$ are diffusion coefficients evaluated at time $t = n\cdot\tau$. Assuming diffusion coefficients $a(x,y,t)$ and $b(x,y,t)$ are twice continuously differentiable, then we have the

following approximation:

$$a_{i,j}^{n} = a_{i,j}^{n+\frac{1}{2}} - (a')_{i,j}^{n+\frac{1}{2}} + \mathcal{O}(\tau^2) \quad a_{i,j}^{n+1} = a_{i,j}^{n+\frac{1}{2}} + (a')_{i,j}^{n+\frac{1}{2}} + \mathcal{O}(\tau^2)$$
$$b_{i,j}^{n} = b_{i,j}^{n+\frac{1}{2}} - (b')_{i,j}^{n+\frac{1}{2}} + \mathcal{O}(\tau^2) \quad b_{i,j}^{n+1} = b_{i,j}^{n+\frac{1}{2}} + (b')_{i,j}^{n+\frac{1}{2}} + \mathcal{O}(\tau^2)$$

Here, $(a')_{i,j}^{n+\frac{1}{2}}$ and $(b')_{i,j}^{n+\frac{1}{2}}$ are first-order time derivative of $a(x,y,t)$ and $b(x,y,t)$ evaluated at $x = i \cdot h$, $y = j \cdot h$ and $t = (n+\frac{1}{2}) \cdot \tau$. Plugging the approximate above into Eq(2.24) yields

$$\begin{aligned} u_{i,j}^{n+1} - u_{i,j}^{n} = \frac{\tau}{2h^2}\Bigg(& a_{i,j}^{n+\frac{1}{2}} \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n} + a_{i,j}^{n+\frac{1}{2}} \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n+1} \\ & + b_{i,j}^{n+\frac{1}{2}} \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n} + b_{i,j}^{n+\frac{1}{2}} \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n+1} \Bigg) + \frac{\tau}{2}(s_{i,j}^{n} + s_{i,j}^{n+1}) \\ & + \frac{\tau}{2h^2}\Bigg[(a')_{i,j}^{n+\frac{1}{2}} \left(\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n+1} - \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n} \right) \\ & + (b')_{i,j}^{n+\frac{1}{2}} \left(\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n+1} - \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n} \right) \Bigg] + \mathcal{O}(\tau^2) \end{aligned} \tag{2.25}$$

Assuming that the solution u is smooth and all of its third-order derivatives are bounded, by Eq(2.4), we have

$$\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n+1} - \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n} = (u_{xx})_{i,j}^{n+1} - (u_{xx})_{i,j}^{n} + \mathcal{O}(h^4) = (u_{txx})_{i,j}^{n} \cdot \tau + \mathcal{O}(h^4) \tag{2.26}$$
$$\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n+1} - \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n} = (u_{yy})_{i,j}^{n+1} - (u_{yy})_{i,j}^{n} + \mathcal{O}(h^4) = (u_{tyy})_{i,j}^{n} \cdot \tau + \mathcal{O}(h^4) \tag{2.27}$$

By Eq(2.26) and Eq(2.27), Eq(2.25) simplifies to

$$\begin{aligned} u_{i,j}^{n+1} - u_{i,j}^{n} = \frac{\tau}{2h^2}\Bigg(& a_{i,j}^{n+\frac{1}{2}} \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n} + a_{i,j}^{n+\frac{1}{2}} \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} u_{i,j}^{n+1} + b_{i,j}^{n+\frac{1}{2}} \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n} \\ & + b_{i,j}^{n+\frac{1}{2}} \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} u_{i,j}^{n+1} \Bigg) + \frac{\tau}{2}(s_{i,j}^{n} + s_{i,j}^{n+1}) + \mathcal{O}(h^4) + \mathcal{O}(\tau^2) \end{aligned} \tag{2.28}$$

Since our original algorithm is fourth-order accurate in space and second-order accurate in time, the error term in Eq(2.28) can be strategically neglected without affecting the order of accuracy, which has been shown by a numerical example in Section 4.5. Eq(2.28) can be solved using the ADI technique previously proposed in Section 2.1 and the two equations to be solved are

$$\begin{cases} \left(1-\alpha_{i,j}^{n+\frac{1}{2}}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)u_{i,j}^{\star}=\left(1+\alpha_{i,j}^{n+\frac{1}{2}}\dfrac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1+\beta_{i,j}^{n+\frac{1}{2}}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n} \\ \qquad\qquad +\dfrac{\tau}{2}(s_{i,j}^{n}(u_{i,j}^{n})+s_{i,j}^{n+1}(u_{i,j}^{n+1})) \\ \left(1-\beta_{i,j}^{n+\frac{1}{2}}\dfrac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n+1}=u_{i,j}^{\star} \end{cases}$$

Chapter 3

Stability Analysis

It is essential that a numerical method is stable when it is applied to solve a time-dependent problem. The new scheme in this paper was developed based on the Crank-Nicolson algorithm, which is supposed to be unconditionally stable and can be proved by Von Neumann's analysis. However, the popular Von Neumann analysis is not applicable here due to the usage of the Padé approximation-based splitting algorithm and the spatially varying coefficients. Therefore, we adopted the energy method in [3] to analyze and prove the stability of the new method.

For the sake of simplicity, assume zero source and zero Dirichlet boundary conditions for the linear reaction-diffusion equation. We justify here why it suffices to only consider reaction-diffusion equations without source term. Consider the reaction-diffusion equation in Eq(2.1). Firstly, if the source term $s(u, x, y, t)$ is non-linear in u, the equation will be solved using linearization, resulting in a linear source term in u. Hence, stability analysis will always be conducted for linear reaction cases, for which we can write the linear reaction term as $s(u, x, y, t) = cu + \tilde{s}(x, y, t)$, with c being a constant. According to the discussion in Section 2.2, we may further use the function transformation $w(x, y, t) = e^{-ct}u(x, y, t)$ to eliminate the linear term cu. Thus, without loss of generality, we can assume that the reaction term is independent of u and perform the stability analysis to Eq(2.7).

Let $u_{i,j}^n$ represent the numerical solution obtained by out algorithm Eq(2.11) and $U_{i,j}^n = u(i \cdot h_x, j \cdot h_y, n \cdot \tau)$ be the exact solution at the spatial grid point (i, j) and time $t = n \cdot \tau$, we have

$$\left(1 - \alpha_{i,j} \frac{\delta_x^2}{1 + \frac{1}{12}\delta_x^2}\right) \left(1 - \beta_{i,j} \frac{\delta_y^2}{1 + \frac{1}{12}\delta_y^2}\right) u_{i,j}^{n+1} =$$
$$\left(1 + \alpha_{i,j} \frac{\delta_x^2}{1 + \frac{1}{12}\delta_x^2}\right) \left(1 + \beta_{i,j} \frac{\delta_y^2}{1 + \frac{1}{12}\delta_y^2}\right) u_{i,j}^n + \frac{\tau}{2}(s_{i,j}^n + s_{i,j}^{n+1}) \tag{3.1}$$

and

$$\left(1 - \alpha_{i,j} \frac{\delta_x^2}{1 + \frac{1}{12}\delta_x^2}\right) \left(1 - \beta_{i,j} \frac{\delta_y^2}{1 + \frac{1}{12}\delta_y^2}\right) U_{i,j}^{n+1} =$$
$$\left(1 + \alpha_{i,j} \frac{\delta_x^2}{1 + \frac{1}{12}\delta_x^2}\right) \left(1 + \beta_{i,j} \frac{\delta_y^2}{1 + \frac{1}{12}\delta_y^2}\right) U_{i,j}^n + \frac{\tau}{2}(s_{i,j}^n + s_{i,j}^{n+1}) + \mathcal{O}(\tau^2 + h^4) \tag{3.2}$$

The error term $\mathcal{O}(\tau^2 + h^4)$ in Eq(3.2) is the truncation errors from Crank-Nicolson scheme and Padè approximation. The numerical error is defined by $e_{i,j}^n = u_{i,j}^n - U_{i,j}^n$. Subtracting Eq(3.2) from Eq(3.1), we obtain

$$\left(1 - \alpha_{i,j} \frac{\delta_x^2}{1 + \frac{1}{12}\delta_x^2}\right) \left(1 - \beta_{i,j} \frac{\delta_y^2}{1 + \frac{1}{12}\delta_y^2}\right) e_{i,j}^{n+1} =$$
$$\left(1 + \alpha_{i,j} \frac{\delta_x^2}{1 + \frac{1}{12}\delta_x^2}\right) \left(1 + \beta_{i,j} \frac{\delta_y^2}{1 + \frac{1}{12}\delta_y^2}\right) e_{i,j}^n + \mathcal{O}(\tau^2 + h^4) \tag{3.3}$$

The numerical stability analysis is concerned with the growth of error during time marching. We note that Eq(3.3) is independent of the source term s, which indicates that the behaviour of error growth is independent of the source term. Therefore, it suffices to prove the stability result only for equations with zero source term.

Now we return from digression and continue to prove the stability of the algorithm. Assume the uniform step h is used in both x and y directions. ($h = h_x = h_y$). Consider the

Padé approximation based fourth-order finite difference scheme

$$\frac{u_{i,j}^{n+1}-u_{i,j}^{n}}{\tau}=\left[\frac{a_{i,j}}{2h^2}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}+\frac{b_{i,j}}{2h^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right](u_{i,j}^{n+1}+u_{i,j}^{n}) \tag{3.4}$$

If we let

$$\mathcal{L}=\frac{\tau\cdot a_{i,j}}{2h^2}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}+\frac{\tau\cdot b_{i,j}}{2h^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}=\frac{\tau}{2h^2}\cdot T_x+\frac{\tau}{2h^2}\cdot T_y \tag{3.5}$$

the scheme can be written as

$$u^{n+1}-u^{n}=\mathcal{L}(u^{n+1}+u^{n}) \tag{3.6}$$

where u^n is the numerical solution at time level t_n:

$$u^n=(u_{i,j}^n)_{N\times N}$$

Here we assume that $N_1=N_2=N$.

To prove the stability result, we first state the following lemma, which is part (c) of Proposition 2 in [20].

Lemma 3.1. *For any non-zero real matrix $w\in\mathbb{R}^{N\times N}$, if $a_{i,j}\geqslant a_0>0$, $b_{i,j}\geqslant b_0>0$, then*

$$\langle w,\mathcal{L}w\rangle<0$$

where $\mathcal{L}$ is a linear operator defined in Eq.(3.5)

Proof. Since $\mathcal{L}=\frac{\tau}{2h^2}\cdot T_x+\frac{\tau}{2h^2}\cdot T_y$. $\langle w,\mathcal{L}w\rangle=\frac{\tau}{2h^2}\langle w,T_xw\rangle+\frac{\tau}{2h^2}\langle w,T_yw\rangle$. Let's first prove that $\langle w,T_xw\rangle<0$ for any non-zero w.

We firstly consider the inner product $\langle w,T_xw\rangle$, which is defined as

$$\langle w,T_xw\rangle=\sum_{j=1}^{N}\langle w_j,a_{i,j}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}w_j\rangle$$

where $w_j = [w_{1,j}, w_{2,j}, \ldots, w_{N,j}] \in \mathbb{R}^N$ is a vector for a fixed j.

It is known that the spectrum of δ_x^2 with the homogeneous Dirichlet coundary condition is given by

$$\sigma(\delta_x^2) = \left\{ -4\sin^2\left(\frac{j\pi}{2(N+1)}\right) \right\} \subset (-4, 0),$$

where N is the number of grid points in the x-direction, $j = 1, \ldots, N$.

Then the spectrum of the operator $\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}$ is given by

$$\sigma\left(\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right) \subset (-6, 0).$$

Let $A_j = \mathrm{Diag}(a_{1,j}, a_{2,j}, \ldots, a_{N,j})$ be a diagonal matrix corresponding to the coefficients $a_{i,j}$ for a fixed j. Since all diagonal entries are positive, A_j is a symmetric and positive definite matrix. By proposition 2 in [20], all eigenvalues of $A_j \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}$ are real and bounded. In fact, for any eigenvalue μ_i, we have

$$-6 \cdot \max_{1 \leqslant i \leqslant N} a_{i,j} \leqslant \mu_i < 0.$$

Nevertheless, we can see that the eigenvalues of $A_j \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}$ are real and negative. Therefore, for any fixed j, and non-zero vector w_j, we have

$$\langle w_j, a_{i,j} \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} w_j \rangle < 0.$$

Subsequently, we have

$$\langle w, T_x w \rangle = \sum_{j=1}^{N} \langle w_j, a_{i,j} \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} w_j \rangle < 0. \tag{3.7}$$

Similarly, we can also prove that

$$\langle w, T_y w \rangle = \sum_{i=1}^{N} \langle w_i, b_{i,j} \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} w_i \rangle < 0. \tag{3.8}$$

Combining Eq.(3.7) and Eq.(3.8), we have

$$\langle w, \mathcal{L}x\rangle = \frac{\tau}{2h^2}(\langle w, T_x w\rangle + \langle w, T_y w\rangle) \tag{3.9}$$

Now we state the main result on the stability of the new method in the following theorem.

Theorem 3.2. *Assume that the solution of the reaction-diffusion equation Eq.(2.1) is sufficiently smooth, the new scheme given in Eq.(2.8) is unconditionally stable.*

Proof. First, let's denote the L^2 norm by $\|\cdot\|$, the inner product on L^2 by $\langle\cdot,\cdot\rangle$. In particular, for $u = (u_{i,j})_{N\times N}$ and $v = (v_{i,j})_{N\times N}$, the inner product is defined by

$$\langle u, v\rangle = \sum_{i=1}^{N}\sum_{j=1}^{N} u_{i,j}v_{i,j}.$$

Note that even if $u, v \in \mathbb{R}^{N\times N}$, they are viewed as vectors and the norm is the L^2 vector norm. Taking inner product with $u^{n+1} - u^2$ on both sides of Eq.(3.4),

$$\langle u^{n+1} - u^n, u^{n+1} + u^n\rangle = \langle u^{n+1} + u^n, \mathcal{L}(u^{n+1} + u^n)\rangle \tag{3.10}$$

Expanding the left-hand side of Eq.(3.10) gives

$$\langle u^{n+1}, u^{n+1}\rangle - \langle u^n, u^n\rangle = \langle u^{n+1} + u^n, \mathcal{L}(u^{n+1} + u^n)\rangle \tag{3.11}$$

Since

$$\langle u^{n+1}, u^{n+1}\rangle = \|u^{n+1}\|^2, \quad \langle u^n, u^n\rangle = \|u^n\|^2,$$

by Lemma 3.1, Eq.(3.11) indicates

$$\|u^{n+1}\|^2 - \|u^n\|^2 = \langle u^{n+1} + u^n, \mathcal{L}(u^{n+1} + u^n)\rangle < 0, \ n = 0, 1, 2, \ldots$$

which is equivalent to

$$\|u^{n+1}\|^2 < \|u^n\|^2,\ n = 0, 1, 2, \ldots$$

for any grid size h and time step τ. Therefore, the fourth-order compact scheme is unconditionally stable. □

Although, in each time step, a sequence of tridiagonal linear systems needs to be solved to march the numerical solution, the high-order ADI methods outperforms other existing methods in terms of overall efficiency because the unconditional stability allows the use of larger time step hence reducing the overall computational cost. In the next section, several numerical examples have been rovided to show that the new method is convergent for large τ, and the overall computational efficiency has been improved significantly.

Chapter 4

Numerical Examples

In this section, we will perform and discuss 6 numerical examples to verify the order of accuracy of our algorithms and, when applicable, study the efficiency of our algorithm.

4.1 Linear u-independent source term

We start from the simplest case and gradually build up on this case by adding increasingly more complicated source terms. Firstly, consider the following Reaction-Diffusion equation:

$$u_t = \frac{x+y+1}{2}u_{xx} + \frac{2}{\sqrt{x^2+y^2+1}}u_{yy} + s(t,x,y) \tag{4.1}$$

We are interested in solving this equation on the domain $[0,1]\times[0,\pi]^2$ subject to Dirichlet boundary condtions:

$$\begin{cases} u(0,x,y) = \sin\left(x+\frac{\pi}{4}\right)\cos(2y) \\ u(t,0,y) = \frac{\sqrt{2}}{2}e^{-t}\cos(2y) \quad u(t,x,0) = e^{-t}\sin\left(x+\frac{\pi}{4}\right) \\ u(t,\pi,y) = \frac{\sqrt{2}}{2}e^{-t}\cos(2y) \quad u(t,x,\pi) = e^{-t}\sin\left(x+\frac{\pi}{4}\right) \end{cases} \tag{4.2}$$

The source term $s(t, x, y)$ is purposely chosen such that the analytic solution is given by:

$$u(t, x, y) = e^{-t} \sin\left(x + \frac{\pi}{4}\right) \cos(2y)$$

To verify the order of convergence, we also compute the empirical order of convergence from the numerical results. The empirical convergence order in space is calculated as

$$\text{Order} = \frac{\log[Error(N_1)/Error(N_2)]}{\log(N_2/N_1)},$$

while, in time, the formula takes the form:

$$\text{Order} = \frac{\log[Error(M_1)/Error(M_2)]}{\log(M_2/M_1)}.$$

The table below demonstrates the numerical result:

Table 4.1: Numerical results of **Example 1** at $T = 1$ with $h = \frac{\pi}{N}$, $\tau = 0.0001$

N	Error (L^∞ norm)	Order (L^∞ norm)	Error (L^2 norm)	cputime(s)
20	2.274×10^{-5}	–	8.952×10^{-6}	59.63
25	9.423×10^{-6}	3.95	3.681×10^{-6}	85.15
32	3.532×10^{-6}	3.97	1.372×10^{-6}	127.57
40	1.455×10^{-6}	3.97	5.618×10^{-7}	189.91
50	5.989×10^{-7}	3.98	2.299×10^{-7}	274.88

We discretize the spatial domain $[0, \pi]^2$ into a $(N + 1) \times (N + 1)$ grid and the temporal direction into M steps, then $h = \frac{\pi}{N}$ and $\tau = \frac{1}{M}$. We notice from the last column in Table 4.1 that our algorithm is empirically 4th order accurate in space, as desired. This implies that if we reduce the the space between two neighbouring nodes in the grid by half, the numerical error will decrease to about $\frac{1}{16}$ of its original size.

The following table demonstrates the order of accuracy in time:

Table 4.2: Numerical results of **Example 1** at $T = 1$ with $\tau = \frac{1}{M}$, $h = \frac{\pi}{500}$

M	Error (L^∞ norm)	Order (L^∞)	CPU time (s)
5	4.425×10^{-3}	–	23.02
10	1.094×10^{-3}	2.02	39.42
20	2.729×10^{-4}	2.00	80.16
40	6.818×10^{-5}	2.00	159.33
80	1.704×10^{-5}	2.00	292.15

We notice from the last column in Table 4.2 that our method is indeed 2nd order accurate in time.

We may also apply the Richardson Extrapolation method to this problem to improve the accuracy in time to 4th order. Table 4.3 shows the result:

Table 4.3: Numerical results of **Example 1** at $T = 1$ based on Richardson Extrapolation with $\tau = \frac{1}{M}$ and $h = \frac{\pi}{500}$

M	Error (L^∞ norm)	Order	CPU time (s)
5	1.642×10^{-5}	–	54.18
10	1.057×10^{-6}	3.96	103.62
20	6.676×10^{-8}	3.98	213.56
40	4.115×10^{-9}	4.02	404.49

We observe that, after using the Richardson extrapolation method, our algorithm is 4-th order in time. The numerical error decays significantly more faster to 0 that the results we presented in 4.2, where Richardson extrapolation was not used. By comparing Table 4.2 and Table 4.3, to reach a certain desired level of computational accuracy (comparable L^∞ error), the algorithm takes significantly less time when supplemented by Richardson extrapolation

method, indicating the method has introduced a favourable improvement to the efficiency of the algorithm.

4.2 Linear u-dependent source term

Next, we add a linear source term of u to the right-hand side of the Eq(4.1):

$$u_t = \frac{x+y+1}{2}u_{xx} + \frac{2}{\sqrt{x^2+y^2+1}}u_{yy} + u + s(t,x,y)$$

We continue to solve this equation on $[0,1] \times [0,\pi]^2$ and impose the boundary condition Eq(4.2). We deliberately choose $s(t,x,y)$ such that the analytic solution is given by

$$u(t,x,y) = e^{-t}\sin\left(x+\frac{\pi}{4}\right)\cos(2y)$$

Using the same discretization method as before, we obtain the following numerical results:

Table 4.4: Numerical results of **Example 2** at $T = 1$ with $h = \frac{\pi}{N}$, $\tau = 0.0001$

N	Error (L^∞ norm)	Order (L^∞ norm)	Error (L^2 norm)	cputime(s)
20	3.050×10^{-5}	–	1.255×10^{-5}	58.79
25	1.271×10^{-5}	3.92	5.178×10^{-6}	85.20
32	4.764×10^{-6}	3.97	1.936×10^{-6}	131.24
40	1.966×10^{-6}	3.97	7.936×10^{-7}	193.60
50	8.092×10^{-7}	3.98	3.252×10^{-7}	292.64

Table 4.5: Numerical results of **Example 2** at $T = 1$ with $\tau = \frac{1}{M}$, $h = \frac{\pi}{500}$

M	Error (L^∞ norm)	Order
5	4.425×10^{-3}	–
10	1.094×10^{-3}	2.02
20	2.729×10^{-4}	2.00
40	6.818×10^{-5}	2.00

Based on the numerical result presented in Table 4.4 and Table 4.5, we conclude that our algorithm is 4th order accurate in space and 2nd order accurate in time. Note that the linear source term $+u$ does not introduce extra amount of work into computation compared to solving Eq(4.1) in section 4.1, due to our exclusive way of coping with linear source terms.

4.3 Nonlinear source term

Next, we add a nonlinear source term to the problem Eq(4.1):

$$u_t = \frac{x+y+1}{2}u_{xx} + \frac{2}{\sqrt{x^2+y^2+1}}u_{yy} + u^2 + s(t,x,y)$$

We solve this equation on $[0,1] \times [0,\pi]^2$ subject to boundary conditions Eq(4.2), and we choose appropriate $s(t,x,y)$ such that the analytic solution to the IVP problem is given by

$$u(t,x,y) = e^{-t} \sin\left(x + \frac{\pi}{4}\right) \cos(2y)$$

In this example, we employ the extrapolation method to deal with the source term. The numerical results are summarized in following tables:

Table 4.6: Numerical results of **Example 3** at $T = 1$ with $h = \frac{\pi}{N}$, $\tau = 0.0001$

N	Error (L^∞ norm)	Order (L^∞ norm)	Error (L^2 norm)	cputime(s)
20	1.929×10^{-5}	–	7.557×10^{-6}	64.31
25	7.992×10^{-6}	3.95	3.106×10^{-6}	91.52
32	3.001×10^{-6}	3.97	1.158×10^{-7}	138.19
40	1.236×10^{-6}	3.97	4.741×10^{-7}	207.67
50	5.098×10^{-7}	3.97	1.942×10^{-7}	309.37

Table 4.7: Numerical results of **Example 3** at $T = 10$ with $\tau = \frac{10}{M}$, $h = \frac{\pi}{500}$

M	Error (L^∞ norm)	Order
5	6.729×10^{-3}	–
10	1.646×10^{-3}	2.03
20	4.064×10^{-4}	2.02
40	1.010×10^{-4}	2.01

The Table 4.6 and Table 4.7 collectively illustrate that our methods are 4th order accurate in space and 2nd order accurate in time. Since we used extrapolation method to accommodate the nonlinear source term, there are extra workload from extrapolation at each time step. More specifically, extrapolations need to be performed on $(N+1)^2$ nodes at

each time step. However, since the temporal grid is uniform, the extrapolation has a closed form, which can be coded into the program beforehand and only takes up a small portion of the total workload. Since the extra workload at each time step is $\mathcal{O}(N^2)$ and the complexity of our algorithm is also $\mathcal{O}(N^2)$, which is the size of the spatial grid, the overall complexity of the algorithm still meets the requirement for ADI method.

4.4 System of equations

Consider the following system of equations:

$$\begin{aligned} u_t &= \frac{x+y+1}{2}u_{xx} + \frac{2}{\sqrt{x^2+y^2+1}}u_{yy} - v + s_1(t,x,y) \\ v_t &= \frac{1}{y^2+1}v_{xx} + \frac{x^2+1}{3}v_{yy} + u + s_2(t,x,y) \end{aligned}$$

We solve the system on $[0,1] \times [0,\pi]^2$ with the boundary conditions:

$$\begin{cases} u(0,x,y) = \cos(x)\sin(2y) \\ u(t,0,y) = e^{-t}\sin(2y) \qquad u(t,x,0) = 0 \\ u(t,\pi,y) = -e^{-t}\sin(2y) \quad u(t,x,\pi) = 0 \end{cases}$$

$$\begin{cases} v(0,x,y) = \cos(x)\cos(y) \\ v(t,0,y) = e^{-t}\cos(y) \qquad v(t,x,0) = e^{-t}\cos(x) \\ v(t,\pi,y) = -e^{-t}\cos(y) \quad v(t,x,\pi) = -e^{-t}\cos(x) \end{cases}$$

We choose $s_1(t,x,y)$ and $s_2(t,x,y)$ carefully such that the analytic solutions of u, v are given by:

$$u(t,x,y) = e^{-t}\cos(x)\sin(2y) \qquad v(t,x,y) = e^{-t}\cos(x)\cos(y)$$

In this example, we use the Prediction-Correction method (Error tolerance= 10^{-8}) to solve this system of equations. The tables below summarize the numerical result:

Table 4.8: Numerical results of **Example 4** at $T = 1$ with $h = \frac{\pi}{N}$, $\tau = 0.0001$

N	Error (L^∞ norm)	Order (L^∞ norm)	Error (L^2 norm)	cputime(s)	Ave. Iter. #
20	1.295×10^{-5}	–	2.833×10^{-6}	65.39	3
25	5.423×10^{-6}	3.90	1.195×10^{-6}	88.50	3
32	2.098×10^{-6}	3.85	4.578×10^{-7}	139.91	3
40	8.857×10^{-7}	3.86	1.928×10^{-7}	189.78	3
50	3.809×10^{-7}	3.78	8.404×10^{-8}	300.97	3

Table 4.9: Numerical results of **Example 4** at $T = 1$ with $\tau = \frac{1}{M}$, $h = \frac{\pi}{300}$

M	Error (L^∞ norm)	Order	Ave. Iter. #
5	4.402×10^{-3}	–	5
10	1.088×10^{-3}	2.02	4
20	2.711×10^{-4}	2.00	4
40	6.773×10^{-5}	2.00	4

The Table 4.8 and Table 4.9 clearly show that our method is 4th order accurate in space and 2nd order accurate in time, as expected. In Prediction-Correction Method, we have to do the entire process of ADI using guess value to find out the next guess value in each iteration and there are about 4 iterations (for a given problem, this number depends on the error tolerance) required before the guesses converge close enough to the real value, the computation workload in this case is about 3 times more than the case where extrapolation method is used. Hence, Prediction-Correction method is relatively expensive in workload. However, the workload remains to be a multiple of the workload in previous sections, so the complexity of our algorithm is still proportional to N^2, namely the size of our 2D spatial

grid.

4.5 Time-dependent diffusion coefficients

Next, we consider the reaction-diffusion equation:

$$u_t = \frac{x+y+t+1}{2}u_{xx} + \frac{2}{\sqrt{x^2+y^2+(t+1)^2}}u_{yy} + s(x,y,t) \tag{4.3}$$

The diffusion process governed by Eq(4.3) takes place in a time-heterogeneous medium, meaning that the diffusion speed varies with time. We once again solve this on $[0,1]\times[0,\pi]^2$ subject to boundary condition Eq(4.2) and choose the source term $s(x,y,t)$ carefully such that the solution is given by:

$$u(t,x,y) = e^{-t}\sin\left(x+\frac{\pi}{4}\right)\cos(2y)$$

The numerical results are presented below:

Table 4.10: Numerical results of **Example 5** at $T = 1$ with $h = \frac{\pi}{N}$, $\tau = 0.0001$

N	Error (L^∞ norm)	Order (L^∞ norm)	Error (L^2 norm)	cputime(s)
20	1.833×10^{-5}	–	7.197×10^{-6}	57.74
25	7.593×10^{-6}	3.95	2.952×10^{-6}	85.58
32	2.843×10^{-6}	3.98	1.098×10^{-7}	131.55
40	1.169×10^{-6}	3.98	4.491×10^{-7}	194.66
50	4.815×10^{-7}	3.98	1.837×10^{-7}	297.23

Table 4.11: Numerical results of **Example 5** at $T = 1$ with $\tau = \frac{1}{M}$, $h = \frac{\pi}{500}$

M	Error (L^∞ norm)	Order
5	4.616×10^{-3}	–
10	1.142×10^{-3}	2.02
20	2.847×10^{-4}	2.00
40	7.112×10^{-4}	2.00

Table 4.10 and Table 4.11 have demonstrated that our algorithm is 4th order accurate in space and 2nd order accurate in time. We remark that Richardson extrapolation is still applicable in this example, hence 4th order accuracy in time is practically attainable.

4.6 Numerically solving an equation

Consider the following system of equations on $\Gamma = [0, \infty) \times [0, \pi]^2$:

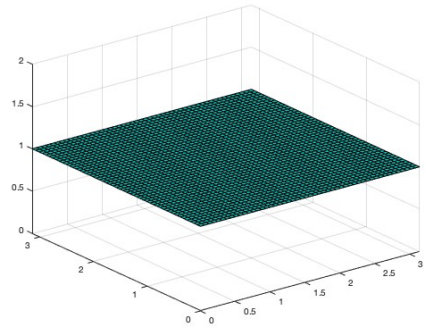
$$u_t = \frac{u_{xx}}{50 + 10x^2 + 10y^2} + \frac{u_{yy}}{300} - u^4 v \qquad v_t = \frac{v_{xx}}{150} + \frac{v_{yy}}{150 + 50(x^2 + y^2)} + u^4 v - 0.5v \quad (4.4)$$

with initial and boundary conditions

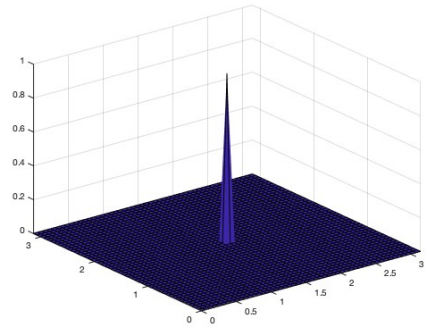
$$\text{Initial Conditions:} \quad u(0,x,y) = 1 \qquad v(0,x,y) = e^{-1600[(x-\frac{\pi}{2})^2+(y-\frac{\pi}{2})^2]}$$

$$\text{Boundary Conditions:} \quad u\big|_{\partial\Gamma} = 1 \qquad v\big|_{\partial\Gamma} = 0$$

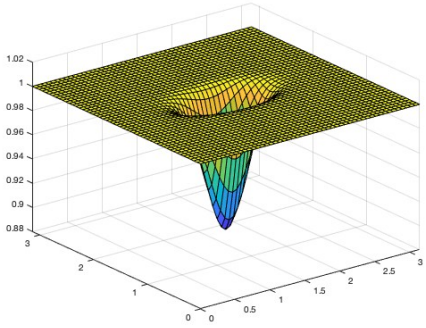
We take $N = 50$ and $\Delta t = \frac{1}{120}$, where Δt is the step length in time. The plot of u, v at $T = 0, 4, 8, 12$ can be found in Fig. 4.1. We can clearly observe the diffusion behaviour of the system from subplots Fig. 4.1 (b)(d)(f)(h). Note that in the second equation of Eq(4.4), the source term $-0.5v$ serves as a sink. Since the mass of v concentrates at the centre, the effect of source term is also most detectable at the centre. In particular, we can observe a saddle-shaped peak formed in the centre of the domain from Fig.4.1(h) due to the existence of the sink. Besides, we can observe from Fig. 4.1 (e)(g) that the mass of u diffuses more quickly in x direction than in y direction, which can be explained by a larger diffusion coefficient in x-direction ($\frac{1}{50+10x^2+10y^2}$ v.s. $\frac{1}{300}$).



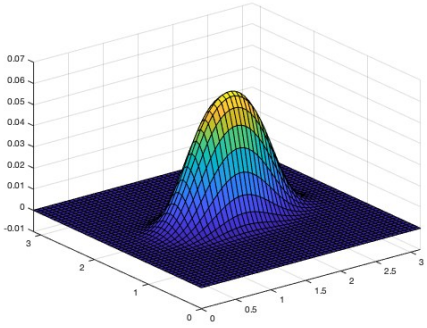
(a) Plot of u at $T = 0$



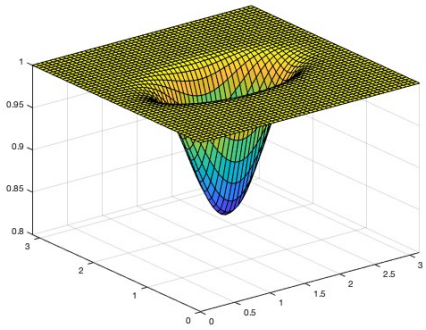
(b) Plot of v at $T = 0$



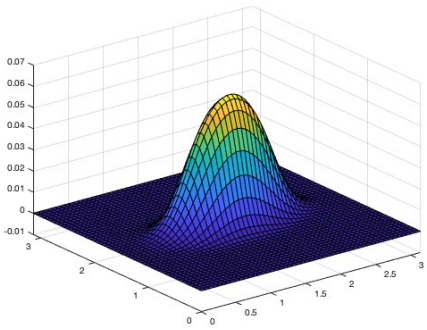
(c) Plot of u at $T = 4$



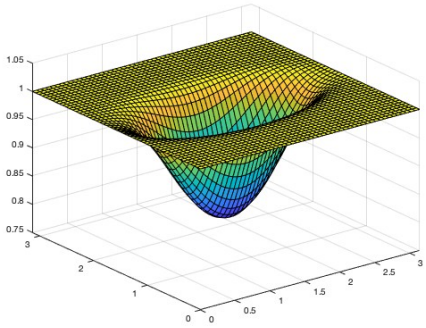
(d) Plot of v at $T = 4$



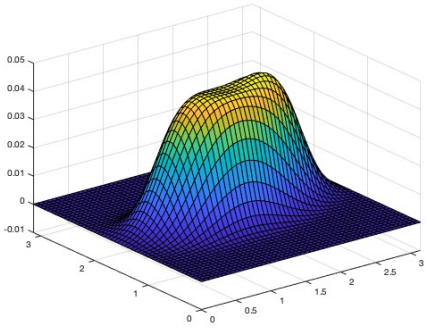
(e) Plot of u at $T = 8$



(f) Plot of v at $T = 8$



(g) Plot of u at $T = 12$



(h) Plot of v at $T = 12$

Figure 4.1: Evolution of u, v

4.7 Application to Biological Model

Finally, we solve the following Fisher-KPP equation subject to zero Dirichlet boundary condition to demonstrate the application of our algorithm to real ecological model on $[0, \pi]^2$:

$$u_t = \frac{1 + x^2 + \sin(y)^2}{10} u_{xx} + \frac{1 + y^2 + \sin(x)^2}{10} u_{yy} + ru(1 - u)(u - k),$$

where the population carrying capacity is normalized to 1 throughout the domain, and k is the minimum subsistence level for the population. For simplicity, we assume the minimum subsistence level for the population is constant over the domain, but it worths noting that our algorithm is able to deal with more complicated case where k varies both spatially and temporally. In order to conform to the boundary condition, we carefully choose the initial condition:

$$u(0, x, y) = \frac{x(\pi - x)y(\pi - y)}{5}$$

Firstly, we fix $k = 0.2$ and vary the scaled population growth rate r to study the effect of different growth rates on population growth. We solve the model until $T = 30$ to observe the long-term behaviour of the population density. The plot of evolution of average population density over time is presented below:

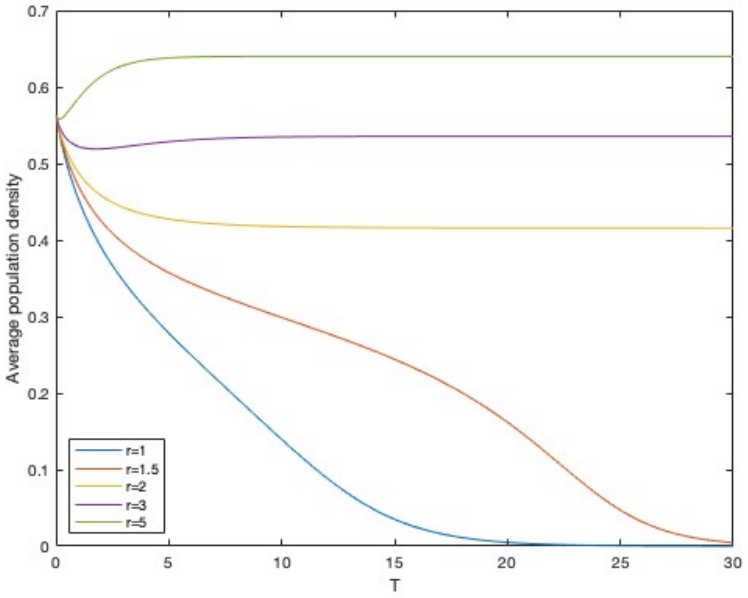
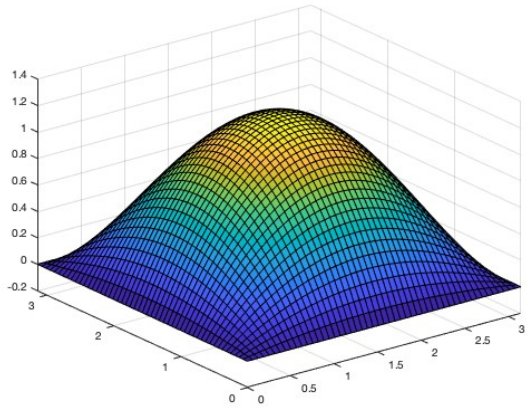
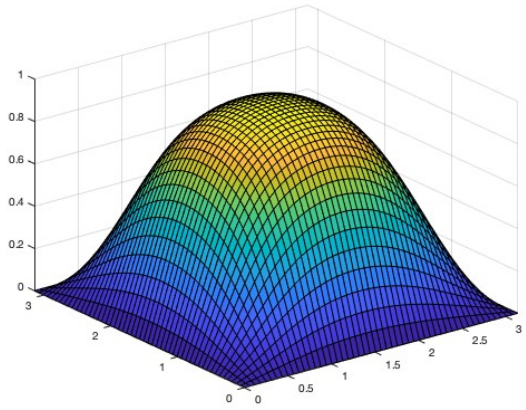


Figure 4.2: Evolution of average population density over time for different r

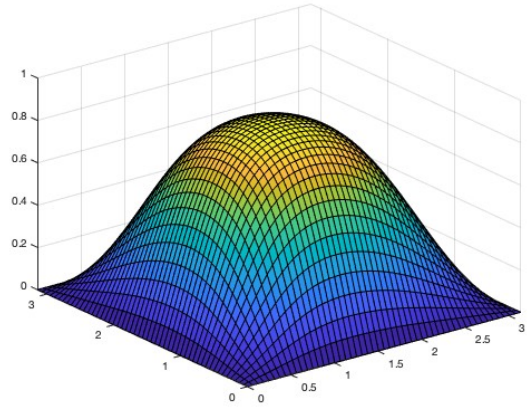
We observe that, given the same initial condition, the population goes extinct for small r ($r = 1, 1.5$). Also, the smaller the r, the faster the population density diminishes. For large r ($r = 2, 3, 5$), the species can eventually survive due to high reproduction rate and the average population density quickly converges to its equilibrium level, indicating that the population reaches a steady state. Furthermore, higher growth rate results in a higher equilibrium level of population density, meaning a more populous steady state. To better illustrate the evolution of population over time, we particularly look into the case when $r = 2$ and plot the population density at 4 time points.



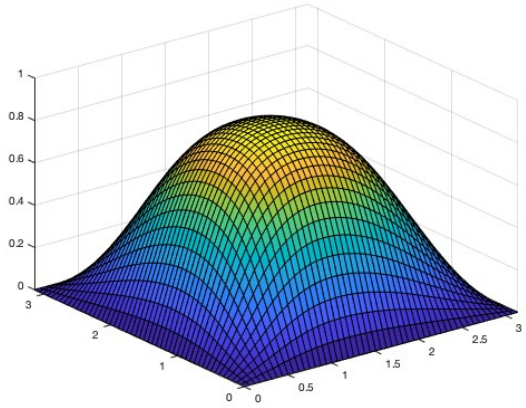
(a) $T = 0$



(b) $T = 1$



(c) $T = 5$



(d) $T = 15$

Figure 4.3: Evolution of population density ($r = 2, k = 0.2$)

We can tell that from $T = 0$ to $T = 1$ that the population density in the middle of the domain is decreasing. This is primarily due to over-crowdedness, where the highest population density point is over the carrying capacity ($u(0, \frac{\pi}{2}, \frac{\pi}{2}) = (\frac{\pi}{2})^4 > 1$). From $T = 1$ to $T = 5$, the population density keeps decreasing, as the diffusion effect drives the population from the populous interior to the unpopulated boundary and the Allee's effect removes the population near the boundary since the population density is below the minimum subsistence level ($u < k = 0.2$). Lower population density in the interior leads to higher reproduction rate and more population are being produced. Finally, when the reproduction is strong enough in the middle to balance with the absorption effect at the boundary, the population

distribution arrives at a steady state. This can be observed from the fact that the population distributions at $T = 5$ and $T = 15$ are very close.

Next, we fix $r = 2$ and study how different levels of minimum subsistence level (k) affect the evolution of population over time. The plot of average population density over time for different k is presented below:

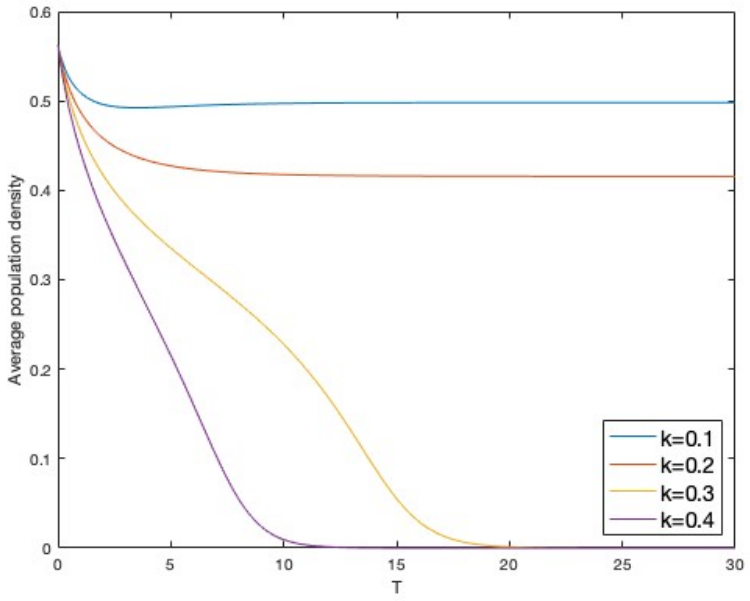


Figure 4.4: Evolution of average population density over time for different r

We note from the diagram that lower minimum subsistence level leads to a higher equilibrium population density. Since the population can only grow at a point when the population density is above k, a smaller k means the species requires easier conditions to grow and survive, hence the absorption effect at the boundary due to Allee's effect is weaker and the species can afford a lower reproduction rate in the middle of the domain, thus the equilibrium population density can be higher than the case when k is large. Moreover, when k is large enough ($k = 0.3, 0.4$), the survival condition for the species is so demanding and absorption effect at the boundary is so strong that no attainable reproduction rate in the interior can balance out the absorption effect, the population will eventually go extinct and the population density converge to 0.

Chapter 5

Conclusion

An efficient and highly accurate numerical algorithm has been developed in this thesis. The new scheme is fourth-order accurate in spatial dimension due to the use of Padé approximation on the second-order standard finite difference operator, while the fourth-order accuracy in temporal dimension can be obtained through Richardson extrapolation. One important feature of the new method is that it can deal with the case when the diffusion coefficients are spatially and temporally varying and different in x, y and t dimensions ($a(x, y) \neq b(x, y)$ in Eq.(2.1)).

A rigid theoretical proof based on the energy method and spectrum theory has been provided to show that the new numerical method is unconditionally stable. The stability of the new method was also validated by several numerical examples in which the computational process is convergent even when the time step is large. This feature is particularly important for applications that need to be simulated for a long time.

Extensive numerical examples have been solved to verify the order of convergence in both time and space. The new scheme is highly accurate in space and time, which makes it suitable for numerical simulation of computationally intensive tasks involving reaction-diffusion models. As indicated in Table 4.3, the new scheme becomes even more efficient when the Richardson extrapolation is implemented.

As of now, we considered the 2D problem with the Dirichlet boundary condition. In the future, we plan to extend the new scheme to 3D problems with other types of boundary conditions, and apply the new scheme to more realistic applications, in particular, the mathematical models from biology and global infectious disease models.

Bibliography

[1] Adams, Y. (1977). Highly accurate compact implicit methods and boundary conditions, *Journal of Computational Physics*, 24, 10-22

[2] Bhatt, H. P., & Khaliq, A.Q.M. (2016). A compact fourth-order L-stable scheme for reaction-diffusion systems with nonsmooth data, *Journal of Computational and Applied Mathematics*, 299, 176-193

[3] Britt, S., Turkel, E., & Tsynkov, S. (2018). A high-order compact time/space finite difference scheme for the wave equation with variable speed of sound, *Journal of Scienfic Computing*, 76(2), 777-811

[4] Britton, N. F. (1986). Reaction-diffusion equations with their applications in biology. Academic Press.

[5] Cantrell, R. S., & Cosner, C. (2004). Spatial ecology via reaction-diffusion equations. John Wiley & Sons.

[6] Chen, J., & Ge, Y. (2018). High order locally one-dimensional methods for solving two-dimensional parabolic equations, *Advances in Difference Equations*, 2018(1), 1-17.

[7] Chu, P., & Fan, C. (1998). A three-point combined compact difference scheme, *Journal of Computational Physics*, 140, 370–399.

[8] Dai, W., & Nassar, R. (2000). A compact finite difference scheme for solving a three-dimensional heat transport equation in a thin film, *Numerical Methods for Partial Differential Equations: An International Journal*, 16(5), 441-458.

[9] Das, S., Liao, W., & Gupta, A. (2014). An efficient fourth-order low dispersive finite difference scheme for a 2-D acoustic wave equation, *Journal of Computational and Applied Mathematics*, 258, 151-167.

[10] Deng, D. (2015). The study of a fourth-order multistep ADI method applied to nonlinear delay reaction–diffusion equations, *Applied Numerical Mathematics*, 96, 118-133.

[11] Douglas, J. Jr. (1955). On the numerical integration of $\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 = \partial u/\partial t$ by implicit methods, *Journal of Society for Industrial and Applied Mathematics*, 3, 42-65.

[12] Douglas, J. Jr., & Gunn, J. (1966). A general formulation of alternating direction methods part I. Parabolic and hyperbolic problems, *Numerische Mathematik*, 6, 428-453.

[13] Düring, B., & Fournié, M. (2012). High-order compact finite difference scheme for option pricing in stochastic volatility models, *Journal of Computational and Applied mathematics*, 236(17), 4462–4473.

[14] Fairweather, G., & Mitchell, A.R. (1965). A high accuracy alternating direction method for the wave equation, *Journal of the Institute of Mathematics and its Applications*, 1, 309-316.

[15] Fisher, R.A. (1937). The wave of advance of advantageous genes, *Annals of Eugenics*, 7.4, 355-369.

[16] Ge, Y., Zhao, F., & Wei, J. (2018). A high order compact ADI method for solving 3D unsteady convection-diffusion problems, *Applied and Computational Mathematics*, 7(1), 1-10.

[17] Gustafson, B., Kreiss, H., & Oliger, J. (1995). Time Dependent Problems and Difference Methods, John Wiley & Sons, New York.

[18] Gu, Y., Liao, W., & Zhu, J. (2003). An efficient high order algorithm for solving systems of 3D reaction-diffusion equations, *Journal of Computational and Applied Mathematics*, 155, 1-17.

[19] Hirsch, R.S. (1975). Higher order accurate difference solutions of fluid mechanics problems by a compact differencing technique, *Journal Of Computational Physics*, 19, 90-109.

[20] Hladnik, M., & Omladič, M. (1988). Spectrum of the product of operators, *Proceedings of the American Mathematical Society*, 102(2), 300-302.

[21] Kondo, S., & Miura, T. (2010). Reaction-diffusion model as a framework for understanding biological pattern formation, *Science*, 329(5999), 1616-1620.

[22] Li, K., Liao, W., & Lin, Y. (2019). A compact high order alternating direction implicit method for three-dimensional acoustic wave equation with variable coefficient, *Journal of Computational and Applied Mathematics*, 361, 113-129.

[23] Liao, H. L., & Sun, Z. Z. (2010). Maximum norm error bounds of ADI and compact ADI methods for solving parabolic equations, *Numerical Methods for Partial Differential Equations: An International Journal*, 26(1), 37-60.

[24] Liao, W. (2014). On the dispersion, stability and accuracy of a compact higher-order finite difference scheme for 3D acoustic wave equation, *Journal of Computational and Applied Mathematics*, 270, 571–583.

[25] Liao, W., Zhu, J., & Kahliq, A.Q.M. (2002). An efficient high-order algorithm for solving systems of reaction-diffusion equations, *Numerical Methods for Partial Differential Equations*, 18(3), 340–354

[26] Liao, W., Yong, P., Dastour, H., & Huang, J. (2018). Efficient and accurate numerical simulation of acoustic wave propagation in a 2D heterogeneous media, *Applied Mathematics and Computation*, 321, 385-400.

[27] Korobenko, L., Kamrujjaman, Md., & Braverman, E. (2013). Persistence and extinction in spatial models with a carrying capacity driven diffusion and harvesting, *Journal of Mathematical Analysis and Applications*, 399(1), 352–368

[28] Ju, L., Liu, X., & Leng, W. (2014). Compact implicit integration factor methods for a family of semilinear fourth-order parabolic equations, *Discrete & Continuous Dynamical Systems*-B, 19(6), 1667.

[29] Lele, S.K. (1992). Compact finite difference schemes with spectral-like resolution, *Journal of Computational Physics*, 103, 16-42.

[30] Peaceman, G.W., & Rachford, H.H. (1955). The numerical solution of parabolic and elliptic differential equations, *Journal of the Society for Industrial and Applied Mathematics* 3(1), 28–41.

[31] Ramos, J.I. (1998). Implicit, compact, linearized θ-methods with factorization for multidimensional reaction-diffusion equations, *Applied Mathematics and Computation*, 94, 17-43.

[32] Sun, Z. Z. (2001), An unconditionally stable and $\mathcal{O}(\tau^2+h^4)$ order L^∞ convergenct difference scheme for linear parabolic equations with variable coefficients, *Numerical Methods for Partial Differential Equations*, 17(6), 619–631.

[33] Sun, Z. Z. (2009). Compact difference schemes for heat equation with Neumann boundary conditions, *Numerical Methods for Partial Differential Equations: An International Journal*, 25(6), 1320-1341.

[34] Wang, Y., & Guo, B. (2008). A monotone compact implicit scheme for nonlinear reaction-diffusion equations, *Journal of Computational Mathematics*, 26(2), 123-148.

[35] Wang, Y., & Zhang, H. (2009). Higher-order compact finite difference method for systems of reaction-diffusion equations, *Journal of Computational and Applied Mathematics*, 233(2), 502-518.

[36] Wu, Y., Ge, Y., & Zhang, L. (2022). A high-order compact LOD difference method for solving the two-dimensional diffusion reaction equation with nonlinear source term, *Journal of Computational Science*, 62, 101748.

[37] Xie, J., & Zhang, Z. (2018). The high-order multistep ADI solver for two-dimensional nonlinear delayed reaction-diffusion equations with variable coefficients, *Computer & Mathematics with Applications*, 75(10), 3558–3570.

[38] Yang, X., Ge, Y., & Zhang, L. (2019). A class of high-order compact difference schemes for solving the Burgers' equations, *Applied Mathematics and Computation*, 358, 394-417.

[39] Zhao, J., Dai, W., & Niu, T. (2007). Fourth-order compact schemes of a heat conduction problem with Neumann boundary conditions, *Numerical Methods for Partial Differential Equations: An International Journal*, 23(5), 949-959.

[40] Zhang, Q., Zhang, C., & Wang, L. (2016). The ADI methods for two-dimensional nonlinear multidelay parabolic equations, *Journal of Computational and Applied Mathematics*, 306, 217–230.

Appendix A

Review of ADI Method

While in this paper we present a 4th order compact ADI scheme for reaction-diffusion equations with variable coefficients, such scheme for equations with constant coefficients has been well-established by researchers long before. See [25] for two-dimensional case and [18] for three-dimensional case. As our work primarily builds on their work, it might be helpful to provide a review for the previous work.

Consider a two-dimensional reaction-diffusion equation with constant coefficients

$$u_t = au_{xx} + bu_{yy} + f(t, x, y, u) \quad a > 0, b > 0 \tag{A.1}$$

We equip this equation with appropriate Dirichlet boundary conditions and initial condition to obtain an Initial Value Problem(IVP).

The finite difference method requires a discretization of Eq(A.1) in time. A commonly-used discretization is the Crank-Nicolson scheme, which is 2nd-order accurate temporally:

$$\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} = \frac{a}{2}((u_{xx})_{i,j}^{n+1} + (u_{xx})_{i,j}^{n}) + \frac{b}{2}((u_{yy})_{i,j}^{n+1} + (u_{yy})_{i,j}^{n}) + \frac{1}{2}(f_{i,j}^{n+1} + f_{i,j}^{n}) \tag{A.2}$$

where ${}_{i,j}^{n}$ denotes the numerical value of relevant function evaluated at the spatial grid point with indices i, j and at the nth time step. Note that Eq(A.2) involves the value of the

second-order derivatives, which cannot be obtained directly. One way to overcome this issue is to approximate the second-order derivatives by function values of u in a neighbourhood. For example, a well-known 2nd order approximation is the central difference scheme, namely

$$(u_{xx})_{i,j}^n = \frac{1}{h_x^2}(u_{i-1,j}^n - 2u_{i,j}^n + u_{i+1,j}^n) + \mathcal{O}(h_x^2)$$
$$(u_{yy})_{i,j}^n = \frac{1}{h_y^2}(u_{i,j-1}^n - 2u_{i,j}^n + u_{i,j+1}^n) + \mathcal{O}(h_y^2)$$

h_x and h_y denote the distance between two neighbouring grid points in x and y direction. However, to obtain a higher order accuracy, higher-order approximation must be used. A popular 4th order approximation, known as Padé approximation, is

$$(u_{xx})_{i,j}^n = \frac{1}{h_x^2}\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^n + \mathcal{O}(h_x^4) \tag{A.3}$$
$$(u_{yy})_{i,j}^n = \frac{1}{h_y^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^n + \mathcal{O}(h_y^4) \tag{A.4}$$

Plugging Eq(A.3) and Eq(A.4) into Eq(A.2) yields a discretization of Eq(A.1) that is 4th order in space and 2nd order in time. The discretization reads

$$\begin{aligned}\frac{u_{i,j}^{n+1}-u_{i,j}^n}{\Delta t} = &\frac{a}{2h_x^2}\left(\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^{n+1} + \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}u_{i,j}^n\right)\\ &+\frac{b}{2h_y^2}\left(\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^{n+1} + \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}u_{i,j}^n\right) + \frac{1}{2}(f_{i,j}^{n+1}+f_{i,j}^n)\end{aligned} \tag{A.5}$$

For simplicity, we assume that $h_x = h_y = h$. One may multiply both sides by Δt and rearrange Eq(A.5) to get an equation with all $u_{i,j}^{n+1}$ on the left-hand side and $u_{i,j}^n$ terms on the right hand side. Namely,

$$\begin{aligned}&\left(1 - r_x\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} - r_y\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^{n+1}\\ =&\left(1 + r_x\frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} + r_y\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right)u_{i,j}^n + \frac{\Delta t}{2}(f_{i,j}^{n+1}+f_{i,j}^n)\end{aligned} \tag{A.6}$$

Here, $r_x = \frac{a\Delta t}{2h^2}$, $r_y = \frac{b\Delta t}{2h^2}$. We notice that we can introduce cross terms so that the operators in Eq(A.6) split as follows:

$$\begin{aligned}
&\left(1 - r_x \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} - r_y \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} + r_x r_y \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n+1} \\
&= \left(1 - r_x \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1 - r_y \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n+1} \qquad \text{(A.7)} \\
&\left(1 + r_x \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2} + r_y \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2} + r_x r_y \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n} \\
&= \left(1 + r_x \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1 + r_y \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n} \qquad \text{(A.8)}
\end{aligned}$$

In [25], it has been computed that the difference between two cross terms is

$$r_x r_y \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}(u_{i,j}^{n+1} - u_{i,j}^{n}) \approx \frac{ab\Delta t^3}{4}(u_{txxyy} + \mathcal{O}(h^4))$$

Since the error term above is $\mathcal{O}(\Delta t^3)$ and that of the algorithm is $\mathcal{O}(\Delta t^2 + h^4)$, the discrepancy between the two error terms does not downgrade the level of accuracy of our algorithm. Introducing the cross terms into Eq(A.6) and using Eq(A.7) and Eq(A.8), we have

$$\begin{aligned}
\left(1 - r_x \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1 - r_y \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n+1} = \left(1 + r_x \frac{\delta_x^2}{1+\frac{1}{12}\delta_x^2}\right)\left(1 + r_y \frac{\delta_y^2}{1+\frac{1}{12}\delta_y^2}\right) u_{i,j}^{n} \\
+ \frac{\Delta t}{2}(f_{i,j}^{n+1} + f_{i,j}^{n}) \qquad \text{(A.9)}
\end{aligned}$$

Multiplying $\left(1 + \frac{1}{12}\delta_x^2\right)\left(1 + \frac{1}{12}\delta_y^2\right)$ on both sides of Eq(A.9) results in

$$\begin{aligned}
\left(1 + \frac{1}{12}\delta_x^2 - r_x\delta_x^2\right)\left(1 + \frac{1}{12}\delta_y^2 - r_y\delta_y^2\right) u_{i,j}^{n+1} = \left(1 + \frac{1}{12}\delta_x^2 + r_x\delta_x^2\right)\left(1 + \frac{1}{12}\delta_y^2 + r_y\delta_y^2\right) u_{i,j}^{n} \\
+ \frac{\Delta t}{2}\left(1 + \frac{1}{12}\delta_x^2\right)\left(1 + \frac{1}{12}\delta_y^2\right)(f_{i,j}^{n+1} + f_{i,j}^{n}) \qquad \text{(A.10)}
\end{aligned}$$

We may solve Eq(A.10) in two steps:

$$\left(1+\frac{1}{12}\delta_x^2-r_x\delta_x^2\right)u_{i,j}^*=\left(1+\frac{1}{12}\delta_x^2+r_x\delta_x^2\right)\left(1+\frac{1}{12}\delta_y^2+r_y\delta_y^2\right)u_{i,j}^n + \frac{\Delta t}{2}\left(1+\frac{1}{12}\delta_x^2\right)\left(1+\frac{1}{12}\delta_y^2\right)(f_{i,j}^{n+1}+f_{i,j}^n) \tag{A.11}$$

$$\left(1+\frac{1}{12}\delta_y^2-r_y\delta_y^2\right)u_{i,j}^{n+1}=u_{i,j}^* \tag{A.12}$$

Note that the right hand side of Eq(A.11) only involves u at time step n, which is known as the solution from previous time iteration. $f_{i,j}^{n+1}$ can be approximated efficiently by Prediction-Correction method [31]. Therefore, the right-hand side of Eq(A.11) can be easily computed. Additionally, both operators on the left-hand side of Eq(A.11) and Eq(A.12) give rise to a tridiagonal linear system, hence can be solved using the Thomas algorithm. Solving Eq(A.11) and Eq(A.12) will give us the desired numerical solution $u_{i,j}^{n+1}$ at time step $n+1$.

Appendix B

Code

For **Example 1** (Table 4.1, 4.2)

```
%% Fourth-order ADI
%% General diffusion coefficients
%% u_t = alpha(x,y)*u_xx+beta(x,y)*u_yy+s(x,y,t) on [0,1]*[0,pi]*[0,pi]
% N>=5
N = 50; h = pi/N;
T = 1; M = 10000; tau = T/M;
% Save numerical solution at each step here
U = {};

%% Coeffcients
% alpha = x+y+1/2; beta = 2/sqrt(x^2+y^2+1)
% Consume the (x,y) and produces the coefficient value at that point
gamma = @(x,y) tau/(2*h^2)*(x+y+1)/2;
epsilon = @(x,y) tau/(2*h^2)*2/sqrt(x^2+y^2+1);
% Initial conditions and Boundary Conditions
% u(0,x,y) = f(x,y)
% u(t,x,0) = g1(t,x)
% u(t,x,pi) = g2(t,x)
% u(t,0,y) = h1(t,y)
% u(t,pi,y) = h2(t,y)
f = @(x,y) sin(x+pi/4)*cos(2*y);
g1 = @(t,x) exp(-t)*sin(x+pi/4);
g2 = @(t,x) exp(-t)*sin(x+pi/4);
h1 = @(t,y) sqrt(2)/2*exp(-t)*cos(2*y);
h2 = @(t,y) -sqrt(2)/2*exp(-t)*cos(2*y);

% Analytic solution
u = @(t,x,y) exp(-t)*sin(x+pi/4)*cos(2*y);
% Source term
s = @(t,x,y) -u(t,x,y)+(x+y+1)/2*u(t,x,y)+8/sqrt(x^2+y^2+1)*u(t,x,y);

tStart = cputime;
```

```
for i = 1:M
    % Implement the initial condition
    if i == 1
        % u_now is a (N+1)*(N+1) matrix
        u_now = zeros(N+1,N+1);
        for j = 1:N+1  % y coordinate
            for k = 1:N+1  % x coordinate
                u_now(j,k) = f((k-1)*h, (j-1)*h);
            end
        end
        U{end+1} = u_now;
    end

    % Add four extra rows on u_now by interpolation
    u_modified = [zeros(2,N+1);u_now;zeros(2,N+1)];
    for j = 1:N+1
        p1 = polyfit(0:h:4*h,u_now(1:5,j),4);
        u_modified(1,j) = polyval(p1,-2*h);
        u_modified(2,j) = polyval(p1,-h);
        p2 = polyfit((N-4)*h:h:N*h,u_now(N-3:N+1,j),4);
        u_modified(N+4,j) = polyval(p2,(N+1)*h);
        u_modified(N+5,j) = polyval(p2,(N+2)*h);
    end

    % Calculate the inner part (x neq 0, pi) of u_prime
    u_prime = zeros(N+1, N+1);
    for j = 1:N+1 % y coordinate
        for k = 2:N % x coordinate
            u_prime(j,k) = 1/12*u_modified(j+2,k-1)/gamma((k-2)*h,(j-1)*h)
   +10/12*u_modified(j+2,k)/gamma((k-1)*h,(j-1)*h)+...
                1/12*u_modified(j+2,k+1)/gamma(k*h,(j-1)*h)+(u_modified(j
   +2,k-1)-2*u_modified(j+2,k)+u_modified(j+2,k+1))...
                +1/12*(-1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*
   h)*u_modified(j,k-1)+...
                4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+1,k-1)-...
                5/2*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+2,k-1)+...
                4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+3,k-1)-...
                1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+4,k-1))...
                +10/12*(-1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)
   *h)*u_modified(j,k)+...
                4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+1,k)-...
                5/2*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+2,k)+...
                4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+3,k)-...
                1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+4,k))...
                +1/12*(-1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*
   u_modified(j,k+1)+...
```

```
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+1,k+1)-...
                5/2*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+2,k+1)+...
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+3,k+1)-...
                1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+4,k+1))...
                +(-1/12*u_modified(j,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
u_modified(j+1,k-1)*epsilon((k-2)*h,(j-1)*h)-...
                5/2*u_modified(j+2,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
u_modified(j+3,k-1)*epsilon((k-2)*h,(j-1)*h)-...
                1/12*u_modified(j+4,k-1)*epsilon((k-2)*h,(j-1)*h))...
                -2*(-1/12*u_modified(j,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
u_modified(j+1,k)*epsilon((k-1)*h,(j-1)*h)-...
                5/2*u_modified(j+2,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
u_modified(j+3,k)*epsilon((k-1)*h,(j-1)*h)-...
                1/12*u_modified(j+4,k)*epsilon((k-1)*h,(j-1)*h))...
                +(-1/12*u_modified(j,k+1)*epsilon(k*h,(j-1)*h)+4/3*
u_modified(j+1,k+1)*epsilon(k*h,(j-1)*h)-...
                5/2*u_modified(j+2,k+1)*epsilon(k*h,(j-1)*h)+4/3*
u_modified(j+3,k+1)*epsilon(k*h,(j-1)*h)-...
                1/12*u_modified(j+4,k+1)*epsilon(k*h,(j-1)*h))...
                +tau/2*(1/12*s((i-1)*tau,(k-2)*h,(j-1)*h)/gamma((k-2)*h,(j
-1)*h)+...
                10/12*s((i-1)*tau,(k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)
+...
                1/12*s((i-1)*tau,k*h,(j-1)*h)/gamma(k*h,(j-1)*h))...
                +tau/2*(1/12*s(i*tau,(k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*
h)+...
                10/12*s(i*tau,(k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)+...
                1/12*s(i*tau,k*h,(j-1)*h)/gamma(k*h,(j-1)*h));
        end
    end
    % u_prime at boundary
    % x = 0
    b_0 = zeros(N+5,1);
    for j = 1:N+1
        b_0(j+2) = h1(i*tau,(j-1)*h);
    end
    p1 = polyfit(0:h:4*h,b_0(3:7),4);
    b_0(1) = polyval(p1,-2*h);
    b_0(2) = polyval(p1,-h);
    p2 = polyfit((N-4)*h:h:N*h,b_0(N-1:N+3),4);
    b_0(N+4) = polyval(p2,(N+1)*h);
    b_0(N+5) = polyval(p2,(N+2)*h);
    for j = 1:N+1
        u_prime(j,1)= b_0(j+2)-epsilon(0,(j-1)*h)*(-1/12*b_0(j)+4/3*b_0(j
+1)-...
            5/2*b_0(j+2)+4/3*b_0(j+3)-1/12*b_0(j+4));
    end

    % x = pi
    b_pi = zeros(N+5,1);
```

```
    for j = 1:N+1
        b_pi(j+2) = h2(i*tau,(j-1)*h);
    end
    p1 = polyfit(0:h:4*h,b_pi(3:7),4);
    b_pi(1) = polyval(p1,-2*h);
    b_pi(2) = polyval(p1,-h);
    p2 = polyfit((N-4)*h:h:N*h,b_pi(N-1:N+3),4);
    b_pi(N+4) = polyval(p2,(N+1)*h);
    b_pi(N+5) = polyval(p2,(N+2)*h);
    for j = 1:N+1
        u_prime(j,N+1)= b_pi(j+2)-epsilon(pi,(j-1)*h)*(-1/12*b_pi(j)+4/3*
b_pi(j+1)-...
            5/2*b_pi(j+2)+4/3*b_pi(j+3)-1/12*b_pi(j+4));
    end

    % Solve in x direction
    u_star = [];
    for j = 1:N+1
        b = u_prime(j,:)';
        % Construct the matrix A
        A = eye(N+1);
        for k = 2:N
            A(k,k-1) = 1/(12*gamma((k-2)*h,(j-1)*h))-1;
            A(k,k) = 10/(12*gamma((k-1)*h,(j-1)*h))+2;
            A(k,k+1) = 1/(12*gamma(k*h,(j-1)*h))-1;
        end

        % Solve
        u_star = [u_star;(A\b)'];
    end
    u_star = u_star(:,2:end-1);

    u_str = zeros(N+1,N-1);
    for k = 1:N-1
        for j = 2:N
            u_str(j,k) = 1/12*u_star(j-1,k)/epsilon(k*h,(j-2)*h)+10/12*
u_star(j,k)/epsilon(k*h,(j-1)*h)...
                +1/12*u_star(j+1,k)/epsilon(k*h,j*h);
        end
        u_str(1,k) = g1(i*tau,k*h);
        u_str(N+1,k) = g2(i*tau,k*h);
    end

    % Solve in y direction
    u_final = [];
    for k = 1:N-1
        c = u_str(:,k);
        % Construct the big matrix A
        A = eye(N+1);
        for j = 2:N
            A(j,j-1) = 1/(12*epsilon(k*h,(j-2)*h))-1;
            A(j,j) = 10/(12*epsilon(k*h,(j-1)*h))+2;
            A(j,j+1) = 1/(12*epsilon(k*h,j*h))-1;
        end
```

```
        % solve
        u_final = [u_final A\c];
    end
    % Our final solution
    u_now = [b_0(3:end-2) u_final b_pi(3:end-2)];
    % store the solution
    U{end+1} = u_now;
end
tEnd = cputime - tStart;
%% Real solution
u_real = zeros(N+1,N+1);
for j = 1:N+1  % y coordinate
    for k = 1:N+1  % x coordinate
        u_real(j,k) = u(T,(k-1)*h, (j-1)*h);
    end
end
Er(end+1) = max(max(abs(u_real-U{end})));
Er(end+1) = norm(reshape(abs(u_real-U{end}),1,[]))/(N-1);
Time(end+1) = tEnd;
```

For **Example 2** (Table 4.4, 4.5)

```
%% Fourth-order ADI
%% General diffusion coefficients
%% u_t = alpha(x,y)*u_xx+beta(x,y)*u_yy+C*u+s(x,y,t) on [0,1]*[0,pi]*[0,pi
    ]
% N>=5
N = 50; h = pi/N;
T = 1; M = 10000; tau = T/M;
C = 1;
% Save numerical solution at each step here
U = {};

%% Coeffcients
% alpha = x+y+1/2; beta = 2/sqrt(x^2+y^2+1)
% Consume the (x,y) and produces the coefficient value at that point
gamma = @(x,y) tau/(2*h^2)*(x+y+1)/2/(1-tau*C/2);
epsilon = @(x,y) tau/(2*h^2)*2/sqrt(x^2+y^2+1)/(1-tau*C/2);
% Initial conditions and Boundary Conditions
% u(0,x,y) = f(x,y)
% u(t,x,0) = g1(t,x)
% u(t,x,pi) = g2(t,x)
% u(t,0,y) = h1(t,y)
% u(t,pi,y) = h2(t,y)
f = @(x,y) sin(x+pi/4)*cos(2*y);
g1 = @(t,x) exp(-t)*sin(x+pi/4);
g2 = @(t,x) exp(-t)*sin(x+pi/4);
h1 = @(t,y) sqrt(2)/2*exp(-t)*cos(2*y);
h2 = @(t,y) -sqrt(2)/2*exp(-t)*cos(2*y);

% Analytic solution
u = @(t,x,y) exp(-t)*sin(x+pi/4)*cos(2*y);
% Source term
s = @(t,x,y) -(1+C)*u(t,x,y)+(x+y+1)/2*u(t,x,y)+8/sqrt(x^2+y^2+1)*u(t,x,y)
    ;

tStart = cputime;
for i = 1:M
    % Implement the initial condition
    if i == 1
        % u_now is a (N+1)*(N+1) matrix
        u_now = zeros(N+1,N+1);
        for j = 1:N+1  % y coordinate
            for k = 1:N+1  % x coordinate
                u_now(j,k) = f((k-1)*h, (j-1)*h);
            end
        end
        U{end+1} = u_now;
    end

    % Add four extra rows on u_now by interpolation
    u_modified = [zeros(2,N+1);u_now;zeros(2,N+1)];
    for j = 1:N+1
        p1 = polyfit(0:h:4*h,u_now(1:5,j),4);
```

```
        u_modified(1,j) = polyval(p1,-2*h);
        u_modified(2,j) = polyval(p1,-h);
        p2 = polyfit((N-4)*h:h:N*h,u_now(N-3:N+1,j),4);
        u_modified(N+4,j) = polyval(p2,(N+1)*h);
        u_modified(N+5,j) = polyval(p2,(N+2)*h);
    end

    % Calculate the inner part (x neq 0, pi) of u_prime
    u_prime = zeros(N+1, N+1);
    for j = 1:N+1 % y coordinate
        for k = 2:N % x coordinate
            u_prime(j,k) = 1/12*u_modified(j+2,k-1)/gamma((k-2)*h,(j-1)*h)
+10/12*u_modified(j+2,k)/gamma((k-1)*h,(j-1)*h)+...
                1/12*u_modified(j+2,k+1)/gamma(k*h,(j-1)*h)+(u_modified(j
+2,k-1)-2*u_modified(j+2,k)+u_modified(j+2,k+1))...
                +1/12*(-1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*
h)*u_modified(j,k-1)+...
                4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
u_modified(j+1,k-1)-...
                5/2*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
u_modified(j+2,k-1)+...
                4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
u_modified(j+3,k-1)-...
                1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
u_modified(j+4,k-1))...
                +10/12*(-1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)
*h)*u_modified(j,k)+...
                4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
u_modified(j+1,k)-...
                5/2*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
u_modified(j+2,k)+...
                4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
u_modified(j+3,k)-...
                1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
u_modified(j+4,k))...
                +1/12*(-1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*
u_modified(j,k+1)+...
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+1,k+1)-...
                5/2*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+2,k+1)+...
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+3,k+1)-...
                1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+4,k+1))...
                +(-1/12*u_modified(j,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
u_modified(j+1,k-1)*epsilon((k-2)*h,(j-1)*h)-...
                5/2*u_modified(j+2,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
u_modified(j+3,k-1)*epsilon((k-2)*h,(j-1)*h)-...
                1/12*u_modified(j+4,k-1)*epsilon((k-2)*h,(j-1)*h))...
                -2*(-1/12*u_modified(j,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
u_modified(j+1,k)*epsilon((k-1)*h,(j-1)*h)-...
                5/2*u_modified(j+2,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
u_modified(j+3,k)*epsilon((k-1)*h,(j-1)*h)-...
```

```
            1/12*u_modified(j+4,k)*epsilon((k-1)*h,(j-1)*h))...
            +(-1/12*u_modified(j,k+1)*epsilon(k*h,(j-1)*h)+4/3*
u_modified(j+1,k+1)*epsilon(k*h,(j-1)*h)-...
            5/2*u_modified(j+2,k+1)*epsilon(k*h,(j-1)*h)+4/3*
u_modified(j+3,k+1)*epsilon(k*h,(j-1)*h)-...
            1/12*u_modified(j+4,k+1)*epsilon(k*h,(j-1)*h))...
            +tau/(2-tau*C)*(1/12*s((i-1)*tau,(k-2)*h,(j-1)*h)/gamma((k
-2)*h,(j-1)*h)+...
            10/12*s((i-1)*tau,(k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)
+...
            1/12*s((i-1)*tau,k*h,(j-1)*h)/gamma(k*h,(j-1)*h))...
            +tau/(2-tau*C)*(1/12*s(i*tau,(k-2)*h,(j-1)*h)/gamma((k-2)*
h,(j-1)*h)+...
            10/12*s(i*tau,(k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)+...
            1/12*s(i*tau,k*h,(j-1)*h)/gamma(k*h,(j-1)*h))+...
            tau*C/(1-tau*C/2)*(1/12*u_modified(j+2,k-1)/gamma((k-2)*h
,(j-1)*h)+...
            10/12*u_modified(j+2,k)/gamma((k-1)*h,(j-1)*h)+1/12*
u_modified(j+2,k+1)/gamma(k*h,(j-1)*h));
    end
end
% u_prime at boundary
% x = 0
b_0 = zeros(N+5,1);
for j = 1:N+1
    b_0(j+2) = h1(i*tau,(j-1)*h);
end
p1 = polyfit(0:h:4*h,b_0(3:7),4);
b_0(1) = polyval(p1,-2*h);
b_0(2) = polyval(p1,-h);
p2 = polyfit((N-4)*h:h:N*h,b_0(N-1:N+3),4);
b_0(N+4) = polyval(p2,(N+1)*h);
b_0(N+5) = polyval(p2,(N+2)*h);
for j = 1:N+1
    u_prime(j,1)= b_0(j+2)-epsilon(0,(j-1)*h)*(-1/12*b_0(j)+4/3*b_0(j
+1)-...
        5/2*b_0(j+2)+4/3*b_0(j+3)-1/12*b_0(j+4));
end

% x = pi
b_pi = zeros(N+5,1);
for j = 1:N+1
    b_pi(j+2) = h2(i*tau,(j-1)*h);
end
p1 = polyfit(0:h:4*h,b_pi(3:7),4);
b_pi(1) = polyval(p1,-2*h);
b_pi(2) = polyval(p1,-h);
p2 = polyfit((N-4)*h:h:N*h,b_pi(N-1:N+3),4);
b_pi(N+4) = polyval(p2,(N+1)*h);
b_pi(N+5) = polyval(p2,(N+2)*h);
for j = 1:N+1
    u_prime(j,N+1)= b_pi(j+2)-epsilon(pi,(j-1)*h)*(-1/12*b_pi(j)+4/3*
b_pi(j+1)-...
        5/2*b_pi(j+2)+4/3*b_pi(j+3)-1/12*b_pi(j+4));
```

```
    end

    % Solve in x direction
    u_star = [];
    for j = 1:N+1
        b = u_prime(j,:)';
        % Construct the matrix A
        A = eye(N+1);
        for k = 2:N
            A(k,k-1) = 1/(12*gamma((k-2)*h,(j-1)*h))-1;
            A(k,k) = 10/(12*gamma((k-1)*h,(j-1)*h))+2;
            A(k,k+1) = 1/(12*gamma(k*h,(j-1)*h))-1;
        end

        % Solve
        u_star = [u_star;(A\b)'];
    end
    u_star = u_star(:,2:end-1);

    u_str = zeros(N+1,N-1);
    for k = 1:N-1
        for j = 2:N
            u_str(j,k) = 1/12*u_star(j-1,k)/epsilon(k*h,(j-2)*h)+10/12*
   u_star(j,k)/epsilon(k*h,(j-1)*h)...
                +1/12*u_star(j+1,k)/epsilon(k*h,j*h);
        end
        u_str(1,k) = g1(i*tau,k*h);
        u_str(N+1,k) = g2(i*tau,k*h);
    end

    % Solve in y direction
    u_final = [];
    for k = 1:N-1
        c = u_str(:,k);
        % Construct the big matrix A
        A = eye(N+1);
        for j = 2:N
            A(j,j-1) = 1/(12*epsilon(k*h,(j-2)*h))-1;
            A(j,j) = 10/(12*epsilon(k*h,(j-1)*h))+2;
            A(j,j+1) = 1/(12*epsilon(k*h,j*h))-1;
        end
        % solve
        u_final = [u_final A\c];
    end
    % Our final solution
    u_now = [b_0(3:end-2) u_final b_pi(3:end-2)];
    % store the solution
    U{end+1} = u_now;
end
tEnd = cputime - tStart;

%% Real solution
u_real = zeros(N+1,N+1);
for j = 1:N+1  % y coordinate
```

```
    for k = 1:N+1  % x coordinate
        u_real(j,k) = u(T,(k-1)*h, (j-1)*h);
    end
end
Er(end+1) = max(max(abs(u_real-U{end})));
Time(end+1) = tEnd;
Er(end+1) = norm(reshape(abs(u_real-U{end}),1,[]))/(N-1);
```

For **Example 3** (Table 4.6, 4.7). The numerical example in Section 4.7 used the same algorithm, however with different coefficients and source function, initial condition and boundary conditions.

```
%% Fourth-order ADI
%% General diffusion coefficients
%% u_t = alpha(x,y)*u_xx+beta(x,y)*u_yy+fu+s(x,y,t) on [0,1]*[0,pi]*[0,pi]
% N>=5
N = 50; h = pi/N;
T = 1; M = 10000; tau = T/M;
% Save numerical solution at each step here
U = {};
% Collection of all source term
F = {};

%% Coeffcients
% alpha = x+y+1/2; beta = 2/sqrt(x^2+y^2+1)
% Consume the (x,y) and produces the coefficient value at that point
gamma = @(x,y) tau/(2*h^2)*(x+y+1)/2;
epsilon = @(x,y) tau/(2*h^2)*2/sqrt(x^2+y^2+1);
% Initial conditions and Boundary Conditions
% u(0,x,y) = f(x,y)
% u(t,x,0) = g1(t,x)
% u(t,x,pi) = g2(t,x)
% u(t,0,y) = h1(t,y)
% u(t,pi,y) = h2(t,y)
f = @(x,y) sin(x+pi/4)*cos(2*y);
g1 = @(t,x) exp(-t)*sin(x+pi/4);
g2 = @(t,x) exp(-t)*sin(x+pi/4);
h1 = @(t,y) sqrt(2)/2*exp(-t)*cos(2*y);
h2 = @(t,y) -sqrt(2)/2*exp(-t)*cos(2*y);

% Analytic solution
u = @(t,x,y) exp(-t)*sin(x+pi/4)*cos(2*y);
% Source term
fu = @(x) x.^2;
s = @(t,x,y) -u(t,x,y)+(x+y+1)/2*u(t,x,y)+8/sqrt(x^2+y^2+1)*u(t,x,y)-fu(u(
    t,x,y));

%% Constructing the first few matrices
for p = 0:1
    u_now = zeros(N+1,N+1);
    f_now = zeros(N+1,N+1);
    for j = 1:N+1  % y coordinate
        for k = 1:N+1  % x coordinate
            u_now(j,k) = exp(-p*tau)*f((k-1)*h, (j-1)*h);
            f_now(j,k) = fu(u_now(j,k));
        end
    end
    U{end+1} = u_now;
    F{end+1} = f_now;
end
```

```
tStart = cputime;
for i = 2:M
    u_now = U{end};

    % Use extrapolation to get f^{n+1}
    f_next = 2*F{end}-F{end-1};

    % Add four extra rows on u_now by interpolation
    u_modified = [zeros(2,N+1);u_now;zeros(2,N+1)];
    for j = 1:N+1
        p1 = polyfit(0:h:4*h,u_now(1:5,j),4);
        u_modified(1,j) = polyval(p1,-2*h);
        u_modified(2,j) = polyval(p1,-h);
        p2 = polyfit((N-4)*h:h:N*h,u_now(N-3:N+1,j),4);
        u_modified(N+4,j) = polyval(p2,(N+1)*h);
        u_modified(N+5,j) = polyval(p2,(N+2)*h);
    end

    % Calculate the inner part (x neq 0, pi) of u_prime
    u_prime = zeros(N+1, N+1);
    for j = 1:N+1 % y coordinate
        for k = 2:N % x coordinate
            u_prime(j,k) = 1/12*u_modified(j+2,k-1)/gamma((k-2)*h,(j-1)*h)
   +10/12*u_modified(j+2,k)/gamma((k-1)*h,(j-1)*h)+...
                1/12*u_modified(j+2,k+1)/gamma(k*h,(j-1)*h)+(u_modified(j
   +2,k-1)-2*u_modified(j+2,k)+u_modified(j+2,k+1))...
                +1/12*(-1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*
   h)*u_modified(j,k-1)+...
                4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+1,k-1)-...
                5/2*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+2,k-1)+...
                4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+3,k-1)-...
                1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
   u_modified(j+4,k-1))...
                +10/12*(-1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)
   *h)*u_modified(j,k)+...
                4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+1,k)-...
                5/2*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+2,k)+...
                4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+3,k)-...
                1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
   u_modified(j+4,k))...
                +1/12*(-1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*
   u_modified(j,k+1)+...
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
   +1,k+1)-...
                5/2*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
   +2,k+1)+...
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
```

```
+3,k+1)-...
              1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*u_modified(j
+4,k+1))...
              +(-1/12*u_modified(j,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
u_modified(j+1,k-1)*epsilon((k-2)*h,(j-1)*h)-...
              5/2*u_modified(j+2,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
u_modified(j+3,k-1)*epsilon((k-2)*h,(j-1)*h)-...
              1/12*u_modified(j+4,k-1)*epsilon((k-2)*h,(j-1)*h))...
              -2*(-1/12*u_modified(j,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
u_modified(j+1,k)*epsilon((k-1)*h,(j-1)*h)-...
              5/2*u_modified(j+2,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
u_modified(j+3,k)*epsilon((k-1)*h,(j-1)*h)-...
              1/12*u_modified(j+4,k)*epsilon((k-1)*h,(j-1)*h))...
              +(-1/12*u_modified(j,k+1)*epsilon(k*h,(j-1)*h)+4/3*
u_modified(j+1,k+1)*epsilon(k*h,(j-1)*h)-...
              5/2*u_modified(j+2,k+1)*epsilon(k*h,(j-1)*h)+4/3*
u_modified(j+3,k+1)*epsilon(k*h,(j-1)*h)-...
              1/12*u_modified(j+4,k+1)*epsilon(k*h,(j-1)*h))...
              +tau/2*(1/12*(s((i-1)*tau,(k-2)*h,(j-1)*h)+fu(u_now(j,k-1)
))/gamma((k-2)*h,(j-1)*h)+...
              10/12*(s((i-1)*tau,(k-1)*h,(j-1)*h)+fu(u_now(j,k)))/gamma
((k-1)*h,(j-1)*h)+...
              1/12*(s((i-1)*tau,k*h,(j-1)*h)+fu(u_now(j,k+1)))/gamma(k*h
,(j-1)*h))...
              +tau/2*(1/12*(s(i*tau,(k-2)*h,(j-1)*h)+f_next(j,k-1))/
gamma((k-2)*h,(j-1)*h)+...
              10/12*(s(i*tau,(k-1)*h,(j-1)*h)+f_next(j,k))/gamma((k-1)*h
,(j-1)*h)+...
              1/12*(s(i*tau,k*h,(j-1)*h)+f_next(j,k+1))/gamma(k*h,(j-1)*
h));
      end
 end
 % u_prime at boundary
 % x = 0
 b_0 = zeros(N+5,1);
 for j = 1:N+1
     b_0(j+2) = h1(i*tau,(j-1)*h);
 end
 p1 = polyfit(0:h:4*h,b_0(3:7),4);
 b_0(1) = polyval(p1,-2*h);
 b_0(2) = polyval(p1,-h);
 p2 = polyfit((N-4)*h:h:N*h,b_0(N-1:N+3),4);
 b_0(N+4) = polyval(p2,(N+1)*h);
 b_0(N+5) = polyval(p2,(N+2)*h);
 for j = 1:N+1
     u_prime(j,1)= b_0(j+2)-epsilon(0,(j-1)*h)*(-1/12*b_0(j)+4/3*b_0(j
+1)-...
         5/2*b_0(j+2)+4/3*b_0(j+3)-1/12*b_0(j+4));
 end

 % x = pi
 b_pi = zeros(N+5,1);
 for j = 1:N+1
     b_pi(j+2) = h2(i*tau,(j-1)*h);
```

```
end
p1 = polyfit(0:h:4*h,b_pi(3:7),4);
b_pi(1) = polyval(p1,-2*h);
b_pi(2) = polyval(p1,-h);
p2 = polyfit((N-4)*h:h:N*h,b_pi(N-1:N+3),4);
b_pi(N+4) = polyval(p2,(N+1)*h);
b_pi(N+5) = polyval(p2,(N+2)*h);
for j = 1:N+1
    u_prime(j,N+1)= b_pi(j+2)-epsilon(pi,(j-1)*h)*(-1/12*b_pi(j)+4/3*
b_pi(j+1)-...
        5/2*b_pi(j+2)+4/3*b_pi(j+3)-1/12*b_pi(j+4));
end

% Solve in x direction
u_star = [];
for j = 1:N+1
    b = u_prime(j,:)';
    % Construct the matrix A
    A = eye(N+1);
    for k = 2:N
        A(k,k-1) = 1/(12*gamma((k-2)*h,(j-1)*h))-1;
        A(k,k) = 10/(12*gamma((k-1)*h,(j-1)*h))+2;
        A(k,k+1) = 1/(12*gamma(k*h,(j-1)*h))-1;
    end

    % Solve
    u_star = [u_star;(A\b)'];
end
u_star = u_star(:,2:end-1);

u_str = zeros(N+1,N-1);
for k = 1:N-1
    for j = 2:N
        u_str(j,k) = 1/12*u_star(j-1,k)/epsilon(k*h,(j-2)*h)+10/12*
u_star(j,k)/epsilon(k*h,(j-1)*h)...
            +1/12*u_star(j+1,k)/epsilon(k*h,j*h);
    end
    u_str(1,k) = g1(i*tau,k*h);
    u_str(N+1,k) = g2(i*tau,k*h);
end

% Solve in y direction
u_final = [];
for k = 1:N-1
    c = u_str(:,k);
    % Construct the big matrix A
    A = eye(N+1);
    for j = 2:N
        A(j,j-1) = 1/(12*epsilon(k*h,(j-2)*h))-1;
        A(j,j) = 10/(12*epsilon(k*h,(j-1)*h))+2;
        A(j,j+1) = 1/(12*epsilon(k*h,j*h))-1;
    end
    % solve
    u_final = [u_final A\c];
```

```
    end
    % Our final solution
    u_now = [b_0(3:end-2) u_final b_pi(3:end-2)];
    % store the solution
    U{end+1} = u_now;
    F{end+1} = fu(u_now);
end
tEnd = cputime - tStart;

%% Real solution
u_real = zeros(N+1,N+1);
for j = 1:N+1  % y coordinate
    for k = 1:N+1  % x coordinate
        u_real(j,k) = u(T,(k-1)*h, (j-1)*h);
    end
end
Er(end+1) = max(max(abs(u_real-U{end})));
Er(end+1) = norm(reshape(abs(u_real-U{end}),1,[]))/(N-1);
Time(end+1) = tEnd;
```

For **Example 4** (Table 4.8, 4.9). The numerical example in Section 4.6 used the same algorithm, however with different coefficients, source function, initial conditions and boundary conditions.

```
%% Coupled Equations
%% Fourth-order ADI
%% General diffusion coefficients
%% u_t = a(x,y)*u_xx+b(x,y)*u_yy-v+s(x,y,t) on [0,1]*[0,pi]*[0,pi]
%% v_t = c(x,y)*u_xx+d(x,y)*u_yy+u+s(x,y,t) on [0,1]*[0,pi]*[0,pi]
% N>=5
global N h tau alpha beta gamma epsilon u v;
N = 50; h = pi/N;
T = 1; M = 1000; tau = T/M;
% Save numerical solution at each step here
U = {}; V = {};
% Error Tolerance
tol = 10^(-8); total_Itr = 0;

%% Coeffcients
% alpha = x+y+1/2; beta = 2/sqrt(x^2+y^2+1)
% gamma = 1/(y^2+1); epsilon = (x^2+1)/3
% Consume the (x,y) and produces the coefficient value at that point
alpha = @(x,y) tau/(2*h^2)*(x+y+1)/2;
beta = @(x,y) tau/(2*h^2)*2/sqrt(x^2+y^2+1);
gamma = @(x,y) tau/(2*h^2)*1/(y^2+1);
epsilon = @(x,y) tau/(2*h^2)*(x^2+1)/3;

%% Initial Conditions and contructing the first matrix
% u(0,x,y) = f1(x,y)
f1 = @(x,y) cos(x)*sin(2*y);
% v(0,x,y) = f2(x,y)
f2 = @(x,y) cos(x)*cos(y);
u_now = zeros(N+1,N+1);
v_now = zeros(N+1,N+1);
for j = 1:N+1  % y coordinate
    for k = 1:N+1  % x coordinate
        u_now(j,k) = f1((k-1)*h, (j-1)*h);
    end
end
for j = 1:N+1  % y coordinate
    for k = 1:N+1  % x coordinate
        v_now(j,k) = f2((k-1)*h, (j-1)*h);
    end
end
U{end+1} = u_now;
V{end+1} = v_now;

% Analytic solution
u = @(t,x,y) exp(-t)*cos(x)*sin(2*y);
v = @(t,x,y) exp(-t)*cos(x)*cos(y);

tStart = cputime;
```

```
% Solve for u and v iteratively
for i = 1:M
    u_present = U{end};
    u_last = U{end};
    v_present = V{end};
    v_last = V{end};
    % Max Iterations allowed
    k = 15;
    % Iteratively solve the system
    while k == 15 || (max(max(max(abs(u_present-u_last))),max(max(abs(
   v_present-v_last))))>tol && k>=1)
        total_Itr = total_Itr + 1;
        k = k-1;
        u_last = u_present;
        v_last = v_present;
        u_present = solve_u(U{end},i,V{end},v_present);
        v_present = solve_v(V{end},i,U{end},u_present);
    end
    % Store the solution
    U{end+1} = u_present;
    V{end+1} = v_present;
end
tEnd = cputime - tStart;

%% Real solution
u_real = zeros(N+1,N+1);
for j = 1:N+1  % y coordinate
    for k = 1:N+1  % x coordinate
        u_real(j,k) = u(T,(k-1)*h, (j-1)*h);
    end
end
%Er(end+1) = max(max(abs(u_real-U{end})));
Er(end+1) = norm(reshape(abs(u_real-U{end}),1,[]))/(N-1);
%Time(end+1) = tEnd;

% Solve u Function
% U list so far, i-th step, latest v -> new U list
function u_output = solve_u(u_now, i, v_now, v_present)
    global N h tau alpha beta u v;

    %Boundary Conditions
    % u(t,x,0) = g1(t,x)
    % u(t,x,pi) = g2(t,x)
    % u(t,0,y) = h1(t,y)
    % u(t,pi,y) = h2(t,y)
    g1 = @(t,x) 0;
    g2 = @(t,x) 0;
    h1 = @(t,y) exp(-t)*sin(2*y);
    h2 = @(t,y) -exp(-t)*sin(2*y);
    % Source term
    s = @(t,x,y) -u(t,x,y)+(x+y+1)/2*u(t,x,y)+8/sqrt(x^2+y^2+1)*u(t,x,y)+v
   (t,x,y);

    % Add four extra rows on u_now by interpolation
```

```
u_modified = [zeros(2,N+1);u_now;zeros(2,N+1)];
for j = 1:N+1
    p1 = polyfit(0:h:4*h,u_now(1:5,j),4);
    u_modified(1,j) = polyval(p1,-2*h);
    u_modified(2,j) = polyval(p1,-h);
    p2 = polyfit((N-4)*h:h:N*h,u_now(N-3:N+1,j),4);
    u_modified(N+4,j) = polyval(p2,(N+1)*h);
    u_modified(N+5,j) = polyval(p2,(N+2)*h);
end

% Calculate the inner part (x neq 0, pi) of u_prime
u_prime = zeros(N+1, N+1);
for j = 1:N+1 % y coordinate
    for k = 2:N % x coordinate
        u_prime(j,k) = 1/12*u_modified(j+2,k-1)/alpha((k-2)*h,(j-1)*h)
+10/12*u_modified(j+2,k)/alpha((k-1)*h,(j-1)*h)+...
            1/12*u_modified(j+2,k+1)/alpha(k*h,(j-1)*h)+(u_modified(j
+2,k-1)-2*u_modified(j+2,k)+u_modified(j+2,k+1))...
            +1/12*(-1/12*beta((k-2)*h,(j-1)*h)/alpha((k-2)*h,(j-1)*h)*
u_modified(j,k-1)+...
            4/3*beta((k-2)*h,(j-1)*h)/alpha((k-2)*h,(j-1)*h)*
u_modified(j+1,k-1)-...
            5/2*beta((k-2)*h,(j-1)*h)/alpha((k-2)*h,(j-1)*h)*
u_modified(j+2,k-1)+...
            4/3*beta((k-2)*h,(j-1)*h)/alpha((k-2)*h,(j-1)*h)*
u_modified(j+3,k-1)-...
            1/12*beta((k-2)*h,(j-1)*h)/alpha((k-2)*h,(j-1)*h)*
u_modified(j+4,k-1))...
            +10/12*(-1/12*beta((k-1)*h,(j-1)*h)/alpha((k-1)*h,(j-1)*h)
*u_modified(j,k)+...
            4/3*beta((k-1)*h,(j-1)*h)/alpha((k-1)*h,(j-1)*h)*
u_modified(j+1,k)-...
            5/2*beta((k-1)*h,(j-1)*h)/alpha((k-1)*h,(j-1)*h)*
u_modified(j+2,k)+...
            4/3*beta((k-1)*h,(j-1)*h)/alpha((k-1)*h,(j-1)*h)*
u_modified(j+3,k)-...
            1/12*beta((k-1)*h,(j-1)*h)/alpha((k-1)*h,(j-1)*h)*
u_modified(j+4,k))...
            +1/12*(-1/12*beta(k*h,(j-1)*h)/alpha(k*h,(j-1)*h)*
u_modified(j,k+1)+...
            4/3*beta(k*h,(j-1)*h)/alpha(k*h,(j-1)*h)*u_modified(j+1,k
+1)-...
            5/2*beta(k*h,(j-1)*h)/alpha(k*h,(j-1)*h)*u_modified(j+2,k
+1)+...
            4/3*beta(k*h,(j-1)*h)/alpha(k*h,(j-1)*h)*u_modified(j+3,k
+1)-...
            1/12*beta(k*h,(j-1)*h)/alpha(k*h,(j-1)*h)*u_modified(j+4,k
+1))...
            +(-1/12*u_modified(j,k-1)*beta((k-2)*h,(j-1)*h)+4/3*
u_modified(j+1,k-1)*beta((k-2)*h,(j-1)*h)-...
            5/2*u_modified(j+2,k-1)*beta((k-2)*h,(j-1)*h)+4/3*
u_modified(j+3,k-1)*beta((k-2)*h,(j-1)*h)-...
            1/12*u_modified(j+4,k-1)*beta((k-2)*h,(j-1)*h))...
            -2*(-1/12*u_modified(j,k)*beta((k-1)*h,(j-1)*h)+4/3*
```

```
u_modified(j+1,k)*beta((k-1)*h,(j-1)*h)-...
              5/2*u_modified(j+2,k)*beta((k-1)*h,(j-1)*h)+4/3*u_modified
(j+3,k)*beta((k-1)*h,(j-1)*h)-...
              1/12*u_modified(j+4,k)*beta((k-1)*h,(j-1)*h))...
              +(-1/12*u_modified(j,k+1)*beta(k*h,(j-1)*h)+4/3*u_modified
(j+1,k+1)*beta(k*h,(j-1)*h)-...
              5/2*u_modified(j+2,k+1)*beta(k*h,(j-1)*h)+4/3*u_modified(j
+3,k+1)*beta(k*h,(j-1)*h)-...
              1/12*u_modified(j+4,k+1)*beta(k*h,(j-1)*h))...
              +tau/2*(1/12*(s((i-1)*tau,(k-2)*h,(j-1)*h)-v_now(j,k-1))/
alpha((k-2)*h,(j-1)*h)+...
              10/12*(s((i-1)*tau,(k-1)*h,(j-1)*h)-v_now(j,k))/alpha((k
-1)*h,(j-1)*h)+...
              1/12*(s((i-1)*tau,k*h,(j-1)*h)-v_now(j,k+1))/alpha(k*h,(j
-1)*h))...
              +tau/2*(1/12*(s(i*tau,(k-2)*h,(j-1)*h)-v_present(j,k-1))/
alpha((k-2)*h,(j-1)*h)+...
              10/12*(s(i*tau,(k-1)*h,(j-1)*h)-v_present(j,k))/alpha((k
-1)*h,(j-1)*h)+...
              1/12*(s(i*tau,k*h,(j-1)*h)-v_present(j,k+1))/alpha(k*h,(j
-1)*h));
      end
 end
 % u_prime at boundary
 % x = 0
 b_0 = zeros(N+5,1);
 for j = 1:N+1
     b_0(j+2) = h1(i*tau,(j-1)*h);
 end
 p1 = polyfit(0:h:4*h,b_0(3:7),4);
 b_0(1) = polyval(p1,-2*h);
 b_0(2) = polyval(p1,-h);
 p2 = polyfit((N-4)*h:h:N*h,b_0(N-1:N+3),4);
 b_0(N+4) = polyval(p2,(N+1)*h);
 b_0(N+5) = polyval(p2,(N+2)*h);
 for j = 1:N+1
     u_prime(j,1)= b_0(j+2)-beta(0,(j-1)*h)*(-1/12*b_0(j)+4/3*b_0(j+1)
-...
         5/2*b_0(j+2)+4/3*b_0(j+3)-1/12*b_0(j+4));
 end

 % x = pi
 b_pi = zeros(N+5,1);
 for j = 1:N+1
     b_pi(j+2) = h2(i*tau,(j-1)*h);
 end
 p1 = polyfit(0:h:4*h,b_pi(3:7),4);
 b_pi(1) = polyval(p1,-2*h);
 b_pi(2) = polyval(p1,-h);
 p2 = polyfit((N-4)*h:h:N*h,b_pi(N-1:N+3),4);
 b_pi(N+4) = polyval(p2,(N+1)*h);
 b_pi(N+5) = polyval(p2,(N+2)*h);
 for j = 1:N+1
     u_prime(j,N+1)= b_pi(j+2)-beta(pi,(j-1)*h)*(-1/12*b_pi(j)+4/3*b_pi
```

```
    (j+1)-...
             5/2*b_pi(j+2)+4/3*b_pi(j+3)-1/12*b_pi(j+4));
    end

    % Solve in x direction
    u_star = [];
    for j = 1:N+1
        b = u_prime(j,:)';
        % Construct the matrix A
        A = eye(N+1);
        for k = 2:N
            A(k,k-1) = 1/(12*alpha((k-2)*h,(j-1)*h))-1;
            A(k,k) = 10/(12*alpha((k-1)*h,(j-1)*h))+2;
            A(k,k+1) = 1/(12*alpha(k*h,(j-1)*h))-1;
        end

        % Solve
        u_star = [u_star;(A\b)'];
    end
    u_star = u_star(:,2:end-1);

    u_str = zeros(N+1,N-1);
    for k = 1:N-1
        for j = 2:N
            u_str(j,k) = 1/12*u_star(j-1,k)/beta(k*h,(j-2)*h)+10/12*u_star
    (j,k)/beta(k*h,(j-1)*h)...
                +1/12*u_star(j+1,k)/beta(k*h,j*h);
        end
        u_str(1,k) = g1(i*tau,k*h);
        u_str(N+1,k) = g2(i*tau,k*h);
    end

    % Solve in y direction
    u_final = [];
    for k = 1:N-1
        c = u_str(:,k);
        % Construct the big matrix A
        A = eye(N+1);
        for j = 2:N
            A(j,j-1) = 1/(12*beta(k*h,(j-2)*h))-1;
            A(j,j) = 10/(12*beta(k*h,(j-1)*h))+2;
            A(j,j+1) = 1/(12*beta(k*h,j*h))-1;
        end
        % solve
        u_final = [u_final A\c];
    end
    % Our final solution
    u_output = [b_0(3:end-2) u_final b_pi(3:end-2)];
end

% Solve v Function
% V list so far, i-th step, latest u -> new V list
function v_output = solve_v(v_now, i, u_now, u_present)
    global N h tau gamma epsilon u v;
```

```
%Boundary Conditions
% v(t,x,0) = g1(t,x)
% v(t,x,pi) = g2(t,x)
% v(t,0,y) = h1(t,y)
% v(t,pi,y) = h2(t,y)
g1 = @(t,x) exp(-t)*cos(x);
g2 = @(t,x) -exp(-t)*cos(x);
h1 = @(t,y) exp(-t)*cos(y);
h2 = @(t,y) -exp(-t)*cos(y);
% Source term
s = @(t,x,y) -v(t,x,y)+v(t,x,y)/(y^2+1)+(x^2+1)/3*v(t,x,y)-u(t,x,y);

% Add four extra rows on v_now by interpolation
v_modified = [zeros(2,N+1);v_now;zeros(2,N+1)];
for j = 1:N+1
    p1 = polyfit(0:h:4*h,v_now(1:5,j),4);
    v_modified(1,j) = polyval(p1,-2*h);
    v_modified(2,j) = polyval(p1,-h);
    p2 = polyfit((N-4)*h:h:N*h,v_now(N-3:N+1,j),4);
    v_modified(N+4,j) = polyval(p2,(N+1)*h);
    v_modified(N+5,j) = polyval(p2,(N+2)*h);
end

% Calculate the inner part (x neq 0, pi) of v_prime
v_prime = zeros(N+1, N+1);
for j = 1:N+1 % y coordinate
    for k = 2:N % x coordinate
        v_prime(j,k) = 1/12*v_modified(j+2,k-1)/gamma((k-2)*h,(j-1)*h)
+10/12*v_modified(j+2,k)/gamma((k-1)*h,(j-1)*h)+...
            1/12*v_modified(j+2,k+1)/gamma(k*h,(j-1)*h)+(v_modified(j
+2,k-1)-2*v_modified(j+2,k)+v_modified(j+2,k+1))...
            +1/12*(-1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*
h)*v_modified(j,k-1)+...
            4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
v_modified(j+1,k-1)-...
            5/2*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
v_modified(j+2,k-1)+...
            4/3*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
v_modified(j+3,k-1)-...
            1/12*epsilon((k-2)*h,(j-1)*h)/gamma((k-2)*h,(j-1)*h)*
v_modified(j+4,k-1))...
            +10/12*(-1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)
*h)*v_modified(j,k)+...
            4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
v_modified(j+1,k)-...
            5/2*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
v_modified(j+2,k)+...
            4/3*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
v_modified(j+3,k)-...
            1/12*epsilon((k-1)*h,(j-1)*h)/gamma((k-1)*h,(j-1)*h)*
v_modified(j+4,k))...
            +1/12*(-1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*
v_modified(j,k+1)+...
```

```
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*v_modified(j
+1,k+1)-...
                5/2*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*v_modified(j
+2,k+1)+...
                4/3*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*v_modified(j
+3,k+1)-...
                1/12*epsilon(k*h,(j-1)*h)/gamma(k*h,(j-1)*h)*v_modified(j
+4,k+1))...
                +(-1/12*v_modified(j,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
v_modified(j+1,k-1)*epsilon((k-2)*h,(j-1)*h)-...
                5/2*v_modified(j+2,k-1)*epsilon((k-2)*h,(j-1)*h)+4/3*
v_modified(j+3,k-1)*epsilon((k-2)*h,(j-1)*h)-...
                1/12*v_modified(j+4,k-1)*epsilon((k-2)*h,(j-1)*h))...
                -2*(-1/12*v_modified(j,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
v_modified(j+1,k)*epsilon((k-1)*h,(j-1)*h)-...
                5/2*v_modified(j+2,k)*epsilon((k-1)*h,(j-1)*h)+4/3*
v_modified(j+3,k)*epsilon((k-1)*h,(j-1)*h)-...
                1/12*v_modified(j+4,k)*epsilon((k-1)*h,(j-1)*h))...
                +(-1/12*v_modified(j,k+1)*epsilon(k*h,(j-1)*h)+4/3*
v_modified(j+1,k+1)*epsilon(k*h,(j-1)*h)-...
                5/2*v_modified(j+2,k+1)*epsilon(k*h,(j-1)*h)+4/3*
v_modified(j+3,k+1)*epsilon(k*h,(j-1)*h)-...
                1/12*v_modified(j+4,k+1)*epsilon(k*h,(j-1)*h))...
                +tau/2*(1/12*(s((i-1)*tau,(k-2)*h,(j-1)*h)+u_now(j,k-1))/
gamma((k-2)*h,(j-1)*h)+...
                10/12*(s((i-1)*tau,(k-1)*h,(j-1)*h)+u_now(j,k))/gamma((k
-1)*h,(j-1)*h)+...
                1/12*(s((i-1)*tau,k*h,(j-1)*h)+u_now(j,k+1))/gamma(k*h,(j
-1)*h))...
                +tau/2*(1/12*(s(i*tau,(k-2)*h,(j-1)*h)+u_present(j,k-1))/
gamma((k-2)*h,(j-1)*h)+...
                10/12*(s(i*tau,(k-1)*h,(j-1)*h)+u_present(j,k))/gamma((k
-1)*h,(j-1)*h)+...
                1/12*(s(i*tau,k*h,(j-1)*h)+u_present(j,k+1))/gamma(k*h,(j
-1)*h));
        end
    end
    % v_prime at boundary
    % x = 0
    b_0 = zeros(N+5,1);
    for j = 1:N+1
        b_0(j+2) = h1(i*tau,(j-1)*h);
    end
    p1 = polyfit(0:h:4*h,b_0(3:7),4);
    b_0(1) = polyval(p1,-2*h);
    b_0(2) = polyval(p1,-h);
    p2 = polyfit((N-4)*h:h:N*h,b_0(N-1:N+3),4);
    b_0(N+4) = polyval(p2,(N+1)*h);
    b_0(N+5) = polyval(p2,(N+2)*h);
    for j = 1:N+1
        v_prime(j,1)= b_0(j+2)-epsilon(0,(j-1)*h)*(-1/12*b_0(j)+4/3*b_0(j
+1)-...
            5/2*b_0(j+2)+4/3*b_0(j+3)-1/12*b_0(j+4));
    end
```

```
% x = pi
b_pi = zeros(N+5,1);
for j = 1:N+1
    b_pi(j+2) = h2(i*tau,(j-1)*h);
end
p1 = polyfit(0:h:4*h,b_pi(3:7),4);
b_pi(1) = polyval(p1,-2*h);
b_pi(2) = polyval(p1,-h);
p2 = polyfit((N-4)*h:h:N*h,b_pi(N-1:N+3),4);
b_pi(N+4) = polyval(p2,(N+1)*h);
b_pi(N+5) = polyval(p2,(N+2)*h);
for j = 1:N+1
    v_prime(j,N+1)= b_pi(j+2)-epsilon(pi,(j-1)*h)*(-1/12*b_pi(j)+4/3*
b_pi(j+1)-...
        5/2*b_pi(j+2)+4/3*b_pi(j+3)-1/12*b_pi(j+4));
end

% Solve in x direction
v_star = [];
for j = 1:N+1
    b = v_prime(j,:)';
    % Construct the matrix A
    A = eye(N+1);
    for k = 2:N
        A(k,k-1) = 1/(12*gamma((k-2)*h,(j-1)*h))-1;
        A(k,k) = 10/(12*gamma((k-1)*h,(j-1)*h))+2;
        A(k,k+1) = 1/(12*gamma(k*h,(j-1)*h))-1;
    end

    % Solve
    v_star = [v_star;(A\b)'];
end
v_star = v_star(:,2:end-1);

v_str = zeros(N+1,N-1);
for k = 1:N-1
    for j = 2:N
        v_str(j,k) = 1/12*v_star(j-1,k)/epsilon(k*h,(j-2)*h)+10/12*
v_star(j,k)/epsilon(k*h,(j-1)*h)...
            +1/12*v_star(j+1,k)/epsilon(k*h,j*h);
    end
    v_str(1,k) = g1(i*tau,k*h);
    v_str(N+1,k) = g2(i*tau,k*h);
end

% Solve in y direction
v_final = [];
for k = 1:N-1
    c = v_str(:,k);
    % Construct the big matrix A
    A = eye(N+1);
    for j = 2:N
        A(j,j-1) = 1/(12*epsilon(k*h,(j-2)*h))-1;
```

```
            A(j,j) = 10/(12*epsilon(k*h,(j-1)*h))+2;
            A(j,j+1) = 1/(12*epsilon(k*h,j*h))-1;
        end
        % solve
        v_final = [v_final A\c];
    end
    % Our final solution
    v_output = [b_0(3:end-2) v_final b_pi(3:end-2)];
end
```

For **Example 5** (Table 4.10, 4.11)

```
%% Fourth-order ADI
%% General diffusion coefficients
%% u_t = alpha(x,y)*u_xx+beta(x,y)*u_yy+s(x,y,t) on [0,1]*[0,pi]*[0,pi]
% N>=5
N = 500; h = pi/N;
T = 1; M = 40; tau = T/M;
% Save numerical solution at each step here
U = {};

%% Coeffcients
% alpha = x+y+t+1/2; beta = 2/sqrt(x^2+y^2+(t+1)^2)
% Consume the (x,y) and produces the coefficient value at that point
gamma = @(t,x,y) tau/(2*h^2)*(x+y+t+1)/2;
epsilon = @(t,x,y) tau/(2*h^2)*2/sqrt(x^2+y^2+(t+1)^2);
% Initial conditions and Boundary Conditions
% u(0,x,y) = f(x,y)
% u(t,x,0) = g1(t,x)
% u(t,x,pi) = g2(t,x)
% u(t,0,y) = h1(t,y)
% u(t,pi,y) = h2(t,y)
f = @(x,y) sin(x+pi/4)*cos(2*y);
g1 = @(t,x) exp(-t)*sin(x+pi/4);
g2 = @(t,x) exp(-t)*sin(x+pi/4);
h1 = @(t,y) sqrt(2)/2*exp(-t)*cos(2*y);
h2 = @(t,y) -sqrt(2)/2*exp(-t)*cos(2*y);

% Analytic solution
u = @(t,x,y) exp(-t)*sin(x+pi/4)*cos(2*y);
% Source term
s = @(t,x,y) -u(t,x,y)+(x+y+t+1)/2*u(t,x,y)+8/sqrt(x^2+y^2+(t+1)^2)*u(t,x,
    y);

tStart = cputime;
for i = 1:M
    % Implement the initial condition
    if i == 1
        % u_now is a (N+1)*(N+1) matrix
        u_now = zeros(N+1,N+1);
        for j = 1:N+1  % y coordinate
            for k = 1:N+1  % x coordinate
                u_now(j,k) = f((k-1)*h, (j-1)*h);
            end
        end
        U{end+1} = u_now;
    end

    % Add four extra rows on u_now by interpolation
    u_modified = [zeros(2,N+1);u_now;zeros(2,N+1)];
    for j = 1:N+1
        p1 = polyfit(0:h:4*h,u_now(1:5,j),4);
        u_modified(1,j) = polyval(p1,-2*h);
        u_modified(2,j) = polyval(p1,-h);
```

```
    p2 = polyfit((N-4)*h:h:N*h,u_now(N-3:N+1,j),4);
    u_modified(N+4,j) = polyval(p2,(N+1)*h);
    u_modified(N+5,j) = polyval(p2,(N+2)*h);
end

% Calculate the inner part (x neq 0, pi) of u_prime
u_prime = zeros(N+1, N+1);
for j = 1:N+1 % y coordinate
    for k = 2:N % x coordinate
        u_prime(j,k) = 1/12*u_modified(j+2,k-1)/gamma((i-1/2)*tau,(k
-2)*h,(j-1)*h)+10/12*u_modified(j+2,k)/gamma((i-1/2)*tau,(k-1)*h,(j-1)*
h)+...
            1/12*u_modified(j+2,k+1)/gamma((i-1/2)*tau,k*h,(j-1)*h)+(
u_modified(j+2,k-1)-2*u_modified(j+2,k)+u_modified(j+2,k+1))...
            +1/12*(-1/12*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*h)/gamma((i
-1/2)*tau,(k-2)*h,(j-1)*h)*u_modified(j,k-1)+...
            4/3*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*h)/gamma((i-1/2)*tau
,(k-2)*h,(j-1)*h)*u_modified(j+1,k-1)-...
            5/2*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*h)/gamma((i-1/2)*tau
,(k-2)*h,(j-1)*h)*u_modified(j+2,k-1)+...
            4/3*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*h)/gamma((i-1/2)*tau
,(k-2)*h,(j-1)*h)*u_modified(j+3,k-1)-...
            1/12*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*h)/gamma((i-1/2)*
tau,(k-2)*h,(j-1)*h)*u_modified(j+4,k-1))...
            +10/12*(-1/12*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)/gamma((
i-1/2)*tau,(k-1)*h,(j-1)*h)*u_modified(j,k)+...
            4/3*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)/gamma((i-1/2)*tau
,(k-1)*h,(j-1)*h)*u_modified(j+1,k)-...
            5/2*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)/gamma((i-1/2)*tau
,(k-1)*h,(j-1)*h)*u_modified(j+2,k)+...
            4/3*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)/gamma((i-1/2)*tau
,(k-1)*h,(j-1)*h)*u_modified(j+3,k)-...
            1/12*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)/gamma((i-1/2)*
tau,(k-1)*h,(j-1)*h)*u_modified(j+4,k))...
            +1/12*(-1/12*epsilon((i-1/2)*tau,k*h,(j-1)*h)/gamma((i
-1/2)*tau,k*h,(j-1)*h)*u_modified(j,k+1)+...
            4/3*epsilon((i-1/2)*tau,k*h,(j-1)*h)/gamma((i-1/2)*tau,k*h
,(j-1)*h)*u_modified(j+1,k+1)-...
            5/2*epsilon((i-1/2)*tau,k*h,(j-1)*h)/gamma((i-1/2)*tau,k*h
,(j-1)*h)*u_modified(j+2,k+1)+...
            4/3*epsilon((i-1/2)*tau,k*h,(j-1)*h)/gamma((i-1/2)*tau,k*h
,(j-1)*h)*u_modified(j+3,k+1)-...
            1/12*epsilon((i-1/2)*tau,k*h,(j-1)*h)/gamma((i-1/2)*tau,k*
h,(j-1)*h)*u_modified(j+4,k+1))...
            +(-1/12*u_modified(j,k-1)*epsilon((i-1/2)*tau,(k-2)*h,(j
-1)*h)+4/3*u_modified(j+1,k-1)*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*h)-...
            5/2*u_modified(j+2,k-1)*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*
h)+4/3*u_modified(j+3,k-1)*epsilon((i-1/2)*tau,(k-2)*h,(j-1)*h)-...
            1/12*u_modified(j+4,k-1)*epsilon((i-1/2)*tau,(k-2)*h,(j-1)
*h))...
            -2*(-1/12*u_modified(j,k)*epsilon((i-1/2)*tau,(k-1)*h,(j
-1)*h)+4/3*u_modified(j+1,k)*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)-...
            5/2*u_modified(j+2,k)*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)
+4/3*u_modified(j+3,k)*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h)-...
```

```
                1/12*u_modified(j+4,k)*epsilon((i-1/2)*tau,(k-1)*h,(j-1)*h
))...
                +(-1/12*u_modified(j,k+1)*epsilon((i-1/2)*tau,k*h,(j-1)*h)
+4/3*u_modified(j+1,k+1)*epsilon((i-1/2)*tau,k*h,(j-1)*h)-...
                5/2*u_modified(j+2,k+1)*epsilon((i-1/2)*tau,k*h,(j-1)*h)
+4/3*u_modified(j+3,k+1)*epsilon((i-1/2)*tau,k*h,(j-1)*h)-...
                1/12*u_modified(j+4,k+1)*epsilon((i-1/2)*tau,k*h,(j-1)*h))
...
                +tau/2*(1/12*s((i-1)*tau,(k-2)*h,(j-1)*h)/gamma((i-1/2)*
tau,(k-2)*h,(j-1)*h)+...
                10/12*s((i-1)*tau,(k-1)*h,(j-1)*h)/gamma((i-1/2)*tau,(k-1)
*h,(j-1)*h)+...
                1/12*s((i-1)*tau,k*h,(j-1)*h)/gamma((i-1/2)*tau,k*h,(j-1)*
h))...
                +tau/2*(1/12*s(i*tau,(k-2)*h,(j-1)*h)/gamma((i-1/2)*tau,(k
-2)*h,(j-1)*h)+...
                10/12*s(i*tau,(k-1)*h,(j-1)*h)/gamma((i-1/2)*tau,(k-1)*h,(
j-1)*h)+...
                1/12*s(i*tau,k*h,(j-1)*h)/gamma((i-1/2)*tau,k*h,(j-1)*h));
        end
    end
    % u_prime at boundary
    % x = 0
    b_0 = zeros(N+5,1);
    for j = 1:N+1
        b_0(j+2) = h1(i*tau,(j-1)*h);
    end
    p1 = polyfit(0:h:4*h,b_0(3:7),4);
    b_0(1) = polyval(p1,-2*h);
    b_0(2) = polyval(p1,-h);
    p2 = polyfit((N-4)*h:h:N*h,b_0(N-1:N+3),4);
    b_0(N+4) = polyval(p2,(N+1)*h);
    b_0(N+5) = polyval(p2,(N+2)*h);
    for j = 1:N+1
        u_prime(j,1)= b_0(j+2)-epsilon((i-1/2)*tau,0,(j-1)*h)*(-1/12*b_0(j
)+4/3*b_0(j+1)-...
            5/2*b_0(j+2)+4/3*b_0(j+3)-1/12*b_0(j+4));
    end

    % x = pi
    b_pi = zeros(N+5,1);
    for j = 1:N+1
        b_pi(j+2) = h2(i*tau,(j-1)*h);
    end
    p1 = polyfit(0:h:4*h,b_pi(3:7),4);
    b_pi(1) = polyval(p1,-2*h);
    b_pi(2) = polyval(p1,-h);
    p2 = polyfit((N-4)*h:h:N*h,b_pi(N-1:N+3),4);
    b_pi(N+4) = polyval(p2,(N+1)*h);
    b_pi(N+5) = polyval(p2,(N+2)*h);
    for j = 1:N+1
        u_prime(j,N+1)= b_pi(j+2)-epsilon((i-1/2)*tau,pi,(j-1)*h)*(-1/12*
b_pi(j)+4/3*b_pi(j+1)-...
            5/2*b_pi(j+2)+4/3*b_pi(j+3)-1/12*b_pi(j+4));
```

```
    end

    % Solve in x direction
    u_star = [];
    for j = 1:N+1
        b = u_prime(j,:)';
        % Construct the matrix A
        A = eye(N+1);
        for k = 2:N
            A(k,k-1) = 1/(12*gamma((i-1/2)*tau,(k-2)*h,(j-1)*h))-1;
            A(k,k) = 10/(12*gamma((i-1/2)*tau,(k-1)*h,(j-1)*h))+2;
            A(k,k+1) = 1/(12*gamma((i-1/2)*tau,k*h,(j-1)*h))-1;
        end

        % Solve
        u_star = [u_star;(A\b)'];
    end
    u_star = u_star(:,2:end-1);

    u_str = zeros(N+1,N-1);
    for k = 1:N-1
        for j = 2:N
            u_str(j,k) = 1/12*u_star(j-1,k)/epsilon((i-1/2)*tau,k*h,(j-2)*
   h)+10/12*u_star(j,k)/epsilon((i-1/2)*tau,k*h,(j-1)*h)...
                +1/12*u_star(j+1,k)/epsilon((i-1/2)*tau,k*h,j*h);
        end
        u_str(1,k) = g1(i*tau,k*h);
        u_str(N+1,k) = g2(i*tau,k*h);
    end

    % Solve in y direction
    u_final = [];
    for k = 1:N-1
        c = u_str(:,k);
        % Construct the big matrix A
        A = eye(N+1);
        for j = 2:N
            A(j,j-1) = 1/(12*epsilon((i-1/2)*tau,k*h,(j-2)*h))-1;
            A(j,j) = 10/(12*epsilon((i-1/2)*tau,k*h,(j-1)*h))+2;
            A(j,j+1) = 1/(12*epsilon((i-1/2)*tau,k*h,j*h))-1;
        end
        % solve
        u_final = [u_final A\c];
    end
    % Our final solution
    u_now = [b_0(3:end-2) u_final b_pi(3:end-2)];
    % store the solution
    U{end+1} = u_now;
end
tEnd = cputime;

%% Real solution
u_real = zeros(N+1,N+1);
for j = 1:N+1  % y coordinate
```

```
    for k = 1:N+1  % x coordinate
        u_real(j,k) = u(T,(k-1)*h, (j-1)*h);
    end
end
Er(end+1) = max(max(abs(u_real-U{end})));
Er1(end+1) = norm(reshape(abs(u_real-U{end}),1,[]))/(N-1);
Time(end+1) = tEnd - tStart;
```