重庆大学本科学生毕业设计（论文）

对流扩散方程新型差分格式的交替方向迭代法

学　　生：杨眉剑

学　　号：20181927

指导教师：王坤　副教授

专　　业：信息与计算数学

重庆大学数学与统计学院

2022 年 6 月

Graduation Design(Thesis) of Chongqing University

Alternating Direction Iteration Method of a New Difference Scheme for Convection Diffusion Equation

Undergraduate: Meijian Yang

Supervisor: Associate Prof. Kun Wang

Major: Information and Computing Sciences

College of Mathematics and Statistics

Chongqing University

June 2022

摘 要

本文研究了对流扩散方程基于新型差分格式的交替方向迭代方法。对文献[1]中提出的求解奇异摄动线性对流扩散方程的高精度新型差分格式，本文探讨了其交替方向迭代求解方法以进一步提高该方法的计算速度。不同于经典的差分格式，基于新型差分格式的交替方向迭代法中的最优迭代参数不能用特征值分析得到，对此，本文提出了一种近似估计法。进一步的，本文将新型差分格式和交替方向迭代法推广到了非线性标量与向量形式的 Burgers 方程的计算中，并将新型差分的交替方向迭代法、新型差分的超松弛迭代法、经典差分的交替方向迭代法进行了对比，数值计算表明，基于新型差分格式的交替方向迭代法在求解对流扩散方程时具有良好的计算性能。

关键词：对流扩散方程，Burgers 方程，新型差分格式，交替方向迭代方法

ABSTRACT

In this paper, the alternating direction iteration method for convection diffusion equation based on a new difference scheme is studied. For the new high-precision difference scheme for solving singularly perturbed linear convection diffusion equations proposed in literature[1], this paper discusses its alternating direction iterative solution method to further improve the calculation speed of this method. Different from the classical difference scheme, the optimal iteration parameters in the alternating direction iteration method based on the new difference scheme can not be obtained by eigenvalue analysis. Therefore, an approximate estimation method is proposed in this paper. Furthermore, in this paper, the new difference scheme and the alternating direction iteration method are extended to the calculation of nonlinear scalar and vector burgers equations, and the alternating direction iteration method of the new difference, the over relaxation iteration method of the new difference and the alternating direction iteration method of the classical difference are compared. The numerical results show that, The alternating direction iteration method based on the new difference scheme has good computational performance in solving convection diffusion equations.

Key words: Convection Diffusion Equation, Burgers Equation，New Difference Scheme, Alternating Direction Iteration

目 录

1 绪论

1.1 课题背景

微分方程，是指含有未知函数及其导数的关系式。解微分方程就是找出未知函数，是伴随着微积分学一起发展起来的。是描述事物变化的一个很好的工具，在物理、计算机、化学等科学中有着极其广泛的应用。大多数情况下我们都是关心微分方程的解，可惜的是只有极少数简单的微分方程可以求得解析解，只能用数值方法求解。

偏微分方程是关于未知函数的偏导数的微分方程，在求解物理方程方面有着广泛应用，其对求解更加复杂。偏微分方程可以分为线性偏微分方程和非线性偏微分方程。如果在偏微分方程，未知函数及它的所有偏导数都是线性的，且方程中的系数都仅依赖于自变量（或者是常数），那么这样的偏微分方程就称为线性偏微分方程，特别的，如果方程中的系数都是常数，则称为常系数偏微分方程。显然，如果方程中的系数是自变量的函数，则称为变系数偏微分方程。方程中出现未知函数及偏导数不是线性的，则称为非线性偏微分方程。

本文研究的对流扩散方程（convection diffusion equation ），是流体力学中的一类非常基本的运动方程，其研究结果在流体力学领域具有重要意义。流体力学是力学的一个重要分支，主要研究的是在各种力的作用下流体的运动与作用，在许多工程如模型仿真、图形解算上都有广泛的应用前景。流体力学和其他学科一样，是通过理论分析和实验研究两种手段发展起来的。很早就已有理论流体力学和实验液体力学两大分支。理论分析是用数学方法求出问题的定量结果。但能用这种方法求出结果的问题毕竟是少数，计算流体力学正是为弥补分析方法的不足而发展起来的，本文就是对计算流体力学领域的一个基本问题，偏微分方程快速解算器的探讨。

1.2 研究现状

对流扩散方程同样也是是偏微分方程一个很重要的分支，在众多领域都有着广泛的应用，如流体力学，气体动力学等。由于对流扩散方程很难通过解析的方法得到解析解，所以通过各种数值方法来求解对流扩散方程在数值分析中占有很

重要的地位。本文中我们仅考虑具有如下形式的对流扩散方程：

$$-\varepsilon\Delta u+\alpha u_x+\beta u_y=f,(x,y)\in\Omega=(0,1)\times(0,1), \tag{1.1}$$

其中 $-\varepsilon\Delta u$ 是扩散项，$\alpha u_x+\beta u_y$ 是对流项，右边的 f 描述的流体的源。我们总假设对流系数 、 和源项 f 均为足够光滑的函数。众所周知，该问题在计算流体力学中起着重要作用。当 ε 比较大，扩散项在物理过程中起主导作用，则用标准有限差分方法以及有限元方法求解就可以得到很好的数值结果，但是若 ε 较小，对流项就占主导地位，即对流的影响远大于扩散的影响，该方程变为具有边界层的奇摄动问题，变得难以模拟。

为了克服这一困难，人们提出了许多方法。一种流行的方法是迎风格式方案，该方案在[2] 中进行了考虑。遗憾的是，由于误差污染的影响，简单的迎风格式不能准确地捕捉边界层的特征。并且由于方程存在着边界，导致解决这个问题非常困难，为此许多研究人员专注于构造一些特殊的网格来划分计算区域（见[3]）以增加精度。其中最著名的网格是 20 世纪 90 年代提出的 Shishkin 网格。文献[4] 介绍了多种基于层自适应网格的稳健数值方法，可以提高数值格式的精度和稳定性。基于 Richardson 外推技术，Sun[5] 提出了一种可以提高计算精度的算子插值方案。另外作为著名的数值方法之一的高阶有限差分格式在过去的几十年中也被发展用于求解奇异摄动问题。为了增强对流扩散方程的对流稳定性，Chiu 和 Sheu[6] 构造了一个保持色散关系的对偶紧致迎风格式。Chu 和 Fan[7] 提出了一种具有六阶精度的通用组合紧致差分格式。另一种高阶有限差分方法是指数有限差分法。Pillai[8] 开发了对流扩散问题的四阶指数有限差分法。Tian 和 Dai[9] 也进行了类似的研究，其中获得了常系数和变系数对流扩散方程的四阶格式。

众所周知，传统的基于泰勒展开的有限差分法在推导时只截取有限项。并且在构造该格式时如果考虑更多的项便可以得到更高的收敛阶和更精确的结果。然而，具有奇摄动系数的对流扩散方程的解满足 $\left|u^{(k)}\right|\leq 1/\varepsilon^k$。因此如果应用经典有限差分法，当 ε 非常小时，局部截断误差将非常大。将[10,11] 中的思想推广到奇异摄动问题，文献[1] 研究了一种新的有限差分方法，该方法通过计算泰勒展开式中的无穷项来实现。无论奇异摄动系数如何，所提出的格式都能在均匀网格上达到预测的收敛阶，并且与其他已知格式相比具有更高的计算精度。

当方程 (1.1) 中的 α、β 为 u 的函数时，方程就变为非线性的对流扩散方程。特别的，当 $\alpha=u$、$\beta=u$ 时，就得到了经典的 Burgers 方程。Burgers 方程可以描述非常多的现象。如湍流的数学模型[12]，粘性流体中冲击波的传播[13]。由于非线性对流项的形式和粘性项的出现，可以将其视为经典流体力学模型 Navier-Stokes 方程的简化形式[14]。因此，研究 Burgers 方程有助于理解对流扩散现象。此外，该研究

涉及大量物理问题，如重力作用下两种颗粒在流体悬浮液中的沉降[15]，声传输[16]，交通和翼型流动理论[17]，以及 Navier-Stokes 方程的先决条件[18]。由于其广泛的应用，Burgers 方程的数值方法引起了人们的广泛关注。如 Arminjon 和 Beauchamp[19] 利用空间变量中的双线性函数构造了二维耦合 Burgers 方程的一种数值方法，而 Jain 和 Holla[14] 提出了两种基于三次 B 样条的数值方法；El Sayed 和 Kaya[20] 提出了一种基于 Adomian 分解的方法；而 Khater 等人[13] 描述了一种基于差分切比雪夫多项式的谱配置方法，以获得二维 Burgers 方程的单形式和耦合形式的数值解。

但研究对流项占优的 Burgers 方程的离散方法相对较少，常规的差分方法在该情况下误差会变得很大，因此有必要研究一种在对流的影响远大于扩散的影响下的 Burgers 方程的求解方法，本文通过推广文献[1] 中新的有限差分方法来尝试解决这个问题。

1.3 本文结构

本文研究了文献[1] 提出的求解线性对流扩散方程的新型差分方法的交替方向迭代法（ADI），并将该方法推广到了 Burgers 方程。

第二章介绍了新方法与交替方向迭代法，给出了数值算例，收敛阶进行了验证，也将交替方向迭代法与超松弛迭代法进行了对比。

第三章首先介绍了线性对流扩散方程的新型差分方法的交替方向迭代求解方法，并对其进行改进，得到了一种计算性能较好的格式，并将其推广到非线性的标量以及向量形式的 Burgers 方程，将新型差分的交替方向迭代法、新型差分的超松弛迭代法、经典差分的交替方向迭代法进行的对比。

第四章对文章进行了总结，简单回顾了本文的研究内容，总结了本文的研究成果，并指出了可以进一步讨论的地方。

2 新差分格式与交替方向迭代法

本章将讨论一种新的对流扩散方程的差分方法与交替方向迭代，并对其进行计算模拟实现，这些是本文后面讨论的基础。

2.1 新差分格式

本节将简要回顾论文[1] 中提出的新型差分方法。在一维线性情况下对流扩散方程：

$$-\varepsilon\frac{\partial^2 u}{\partial x^2}+\alpha(x)\frac{\partial u}{\partial x}=f(x),\quad x\in\Omega=(0,1), \tag{2.1}$$

其中 $\alpha(x)\neq 0$。

新型差分方法的本质是使用泰勒展开与原微分方程构造新的格式。我们下面将给出一维对流扩散方程的新型差分方法。

2.1.1 常系数问题

先考虑函数一次项系数为常数的简单情况，即 $\alpha(x)$ 为常数。

设剖分网格均匀，且 $h=1/N$（N 为正整数）为网格尺寸，根据泰勒公式，可以得到如下两个式子：

$$u_{i+1}-u_i=hu_i^{(1)}+\frac{h^2}{2!}u_i^{(2)}+\cdots+\frac{h^n}{n!}u_i^{(n)}+\cdots, \tag{2.2}$$

$$u_{i-1}-u_i=(-h)u_i^{(1)}+\frac{(-h)^2}{2!}u_i^{(2)}+\cdots+\frac{(-h)^n}{n!}u_i^{(n)}+\cdots, \tag{2.3}$$

其中 $x_i=ih, u_i=u(x_i), f_i=f\left(x_i\right)$。

另外一方面，观察原微分方程，在该方程 (2.1) 的两边同时求 $n-2$ 次导数，可以得到：

$$u^{(n)}=\left(\frac{\alpha}{\varepsilon}\right)^{n-1}u^{(1)}-\frac{1}{\varepsilon}\sum_{k=0}^{n-2}\left(\frac{\alpha}{\varepsilon}\right)^{n-2-k}f^{(k)}, \tag{2.4}$$

将上式 (2.4) 代入泰勒展开的第一个式子 (2.2)，有：

$$\begin{aligned} u_{i+1}-u_i &= hu_i^{(1)}+\sum_{n=2}^{+\infty}\frac{h^n}{n!}\left[\left(\frac{\alpha}{\varepsilon}\right)^{n-1}u_i^{(1)}-\frac{1}{\varepsilon}\sum_{k=0}^{n-2}\left(\frac{\alpha}{\varepsilon}\right)^{n-2-k}f_i^{(k)}\right] \\ &= \sum_{n=1}^{+\infty}\frac{h^n}{n!}\left(\frac{\alpha}{\varepsilon}\right)^{n-1}u_i^{(1)}-\frac{1}{\varepsilon}\sum_{n=2}^{+\infty}\frac{h^n}{n!}\sum_{k=0}^{n-2}\left(\frac{\alpha}{\varepsilon}\right)^{n-2-k}f_i^{(k)} \\ &= \frac{\varepsilon}{\alpha}\left(e^r-1\right)u_i^{(1)}-R^+, \end{aligned} \tag{2.5}$$

类似的，我们也可以将 (2.4) 到代入泰勒展开的另外一个式子 (2.3) 得到：

$$u_{i-1}-u_i=\frac{\varepsilon}{\alpha}\left(e^{-r}-1\right)u_i^{(1)}-R^-, \tag{2.6}$$

其中：

$$\begin{aligned} r &= \frac{\alpha h}{\varepsilon}, \\ R^+ &= \frac{1}{\varepsilon}\sum_{n=2}^{+\infty}\frac{h^n}{n!}\sum_{k=0}^{n-2}\left(\frac{\alpha}{\varepsilon}\right)^{n-2-k}f_i^{(k)}=\sum_{n=0}^{+\infty}\frac{\varepsilon^{n+1}}{\alpha^{n+2}}\left[e^r-\sum_{l=0}^{n+1}\frac{r^l}{l!}\right]f_i^{(n)}, \\ R^- &= \frac{1}{\varepsilon}\sum_{n=2}^{+\infty}\frac{(-h)^n}{n!}\sum_{k=0}^{n-2}\left(\frac{\alpha}{\varepsilon}\right)^{n-2-k}f_i^{(k)}=\sum_{n=0}^{+\infty}\frac{\varepsilon^{n+1}}{\alpha^{n+2}}\left[e^{-r}-\sum_{l=0}^{n+1}\frac{(-r)^l}{l!}\right]f_i^{(n)}. \end{aligned} \tag{2.7}$$

运用公式 (2.7) 的两个式子消去 $u^{(1)}$ 可以得到：

$$u_{i-1}-\left(e^{-r}+1\right)u_i+e^{-r}u_{i+1}=\sum_{m=0}^{+\infty}Z_m, \tag{2.8}$$

其中：

$$Z_m=\frac{\varepsilon^{m+1}}{\alpha^{m+2}}\left[\sum_{l=1}^{m+1}\frac{r^l}{l!}\left[e^{-r}+(-1)^l\right]\right]f_i^{(m)}.$$

其对应的离散格式即为：

$$U_{i-1}-\left(e^{-r}+1\right)U_i+e^{-r}U_{i+1}=\sum_{m=0}^{n}Z_m. \tag{2.9}$$

这样我们便得到了在系数为常数的情况下新型差分方法求解线性对流扩散方程的公式 (2.9)。

2.1.2 变系数问题

下面将考虑原微分方程的系数不为常数的情况。仿照系数为常数情况的做法，在该方程的两边同时求 $n-2$ 次导数，有：

$$\begin{aligned} u^{(n)} &= \frac{1}{\varepsilon}\left[\alpha(x)u^{(1)}-f\right]^{(n-2)} \\ &= \frac{1}{\varepsilon}\left[C_{n-2}^{n-2}\alpha u^{(n-1)}+\cdots+C_{n-2}^{k}\alpha^{(n-2-k)}u^{(k+1)}+\cdots+C_{n-2}^{0}\alpha^{(n-2)}u^{(1)}-f^{(n-2)}\right], \end{aligned} \tag{2.10}$$

其中 $\alpha^{(k)}$ 以及 $u^{(k)}$ 表示 α 和 u 的 k 阶导数。

将上式 (2.10) 右边 u 高于二阶的导数再用原式 (2.1) 代入，如此反复，直到右边所有的关于 u 的导数均小于二阶，使得方程 (2.10) 右边仅剩下 $u^{(1)}$，这样我们就可以得到：

$$u^{(n)} = \sum_{k=1}^{n-1} D^k u^{(1)} + \sum_{j=0}^{n-3} \left[\sum_{k=2}^{n-j-1} E_j^k \right] f^{(j)} + \frac{1}{\varepsilon} f^{(n-2)}, \tag{2.11}$$

其中系数 D^k 以及 E_j^k 为关于 $1/\varepsilon^k$ 的常数。可以证明，所有含 $1/\varepsilon^{k+1}$ 的项之和是所有含 $1/\varepsilon^k$ 之和的高阶项。基于公式 (2.11) 忽略右边的高阶小量我们便可以得到新型差分方法的公式。比如仅保留右边的 $1/\varepsilon^{n-1}$ 次量，便得到：

$$u^{(n)} \approx \frac{\alpha^{n-1}}{\varepsilon^{n-1}} u^{(1)} - \frac{\alpha^{n-2}}{\varepsilon^{n-1}} f, \tag{2.12}$$

仿照系数为常数的做法代入泰勒展开公式 (2.2)、(2.3) 中：

$$e^{-p_i} \left(u_{i+1} - u_i \right) \approx \widehat{B}_i^1 u_i^{(1)} + \widehat{C}_{0,i}^1 f_i, \tag{2.13}$$

$$e^{-p_i} \left(u_{i-1} - u_i \right) \approx \bar{B}_i^1 u_i^{(1)} + \bar{C}_{0,i}^1 f_i, \tag{2.14}$$

其中：

$$\begin{aligned}
p_i &= \frac{h\alpha_i}{\varepsilon}, \\
\widehat{B}_i^1 &= \frac{\varepsilon}{\alpha_i} \left(1 - e^{-p_i} \right), \qquad \widehat{C}_{0,i}^1 = -\frac{\varepsilon}{\alpha_i^2} \left(1 - e^{-p_i} - p_i e^{-p_i} \right), \\
\bar{B}_i^1 &= \frac{\varepsilon}{\alpha_i} \left(e^{-2p_i} - e^{-p_i} \right), \quad \bar{C}_{0,i}^1 = -\frac{\varepsilon}{\alpha_i^2} \left(e^{-2p_i} - e^{-p_i} + p_i e^{-p_i} \right).
\end{aligned}$$

根据以上式子 (2.13)、(2.14)，代入消去 $u^{(1)}$ 得到：

$$A_1 U_{i-1} + A_2 U_i + A_3 U_{i+1} = Z_0 f_i, \tag{2.15}$$

其中：

$$\begin{aligned}
A_1 &= -e^{-p_i} \widehat{B}_i^1, \quad A_2 = e^{-p_i} \left(\widehat{B}_i^1 - \bar{B}_i^1 \right), \\
A_3 &= e^{-p_i} \bar{B}_i^1, \quad Z_0 = \bar{B}_i^1 \widehat{C}_{0,i}^1 - \widehat{B}_i^1 \bar{C}_{0,i}^1.
\end{aligned}$$

如上便是新型差分方法在变系数情况下的一阶公式。

观察以上式子，我们可以发现这就是一个三对角矩阵，因此可以用追赶法对该方程进行求解。追赶法是将一个三对角矩阵进行 LU 分解，然后求两个简单方程组，每次求解的复杂度仅为 $O(n)$，可以大大提供求解速度。

同样的，如果在上式中保留 $1/\varepsilon^{n-2}$ 次的量，即有：

$$\begin{aligned} u^{(n)} \approx & \frac{\alpha^{n-1}}{\varepsilon^{n-1}}u^{(1)} - \frac{\alpha^{n-2}}{\varepsilon^{n-1}}f + \frac{(n-1)(n-2)}{2}\frac{\alpha^{n-3}\alpha^{(1)}}{\varepsilon^{n-2}}u^{(1)} \\ & - \frac{n(n-3)}{2}\frac{\alpha^{n-4}\alpha^{(1)}}{\varepsilon^{n-2}}f - \frac{\alpha^{n-3}}{\varepsilon^{n-2}}f^{(1)}, \end{aligned} \tag{2.16}$$

代入原式泰勒展开中，有：

$$\begin{aligned} e^{-p_i}\left(u_{i+1}-u_i\right) &\approx \left(\widehat{B}_i^1+\widehat{B}_i^2\right)u_i^{(1)} + \left(\widehat{C}_{0,i}^1+\widehat{C}_{0,i}^2\right)f_i + \widehat{C}_{1,i}^2 f_i^{(1)}, \\ e^{-p_i}\left(u_{i-1}-u_i\right) &\approx \left(\bar{B}_i^1+\bar{B}_i^2\right)u_i^{(1)} + \left(\bar{C}_{0,i}^1+\bar{C}_{0,i}^2\right)f_i + \bar{C}_{1,i}^2 f_i^{(1)}, \end{aligned} \tag{2.17}$$

其中：

$$\begin{aligned} \widehat{B}_i^2 &= \frac{\alpha_i^{(1)}\varepsilon^2}{2\alpha_i^3}\left[\left(p_i^2-2p_i+2\right)-2e^{-p_i}\right], \\ \bar{B}_i^2 &= \frac{\alpha_i^{(1)}\varepsilon^2}{2\alpha_i^3}\left[e^{-2p_i}\left(p_i^2+2p_i+2\right)-2e^{-p_i}\right], \\ \widehat{C}_{0,i}^2 &= -\frac{\alpha_i^{(1)}\varepsilon^2}{2\alpha_i^4}\left[\left(1-e^{-p_i}-p_ie^{-p_i}\right)\left(p_i^2-2p_i\right)+p_i^3e^{-p_i}\right], \\ \bar{C}_{0,i}^2 &= -\frac{\alpha_i^{(1)}\varepsilon^2}{2\alpha_i^4}\left[\left(e^{-2p_i}-e^{-p_i}+p_ie^{-p_i}\right)\left(p_i^2+2p_i\right)-p_i^3e^{-p_i}\right], \\ \widehat{C}_{1,i}^2 &= -\frac{\varepsilon^2}{\alpha_i^3}\left(1-e^{-p_i}-p_ie^{-p_i}-\frac{p_i^2}{2!}e^{-p_i}\right), \\ \bar{C}_{1,i}^2 &= -\frac{\varepsilon^2}{\alpha_i^3}\left(e^{-2p_i}-e^{-p_i}+p_ie^{-p_i}-\frac{p_i^2}{2!}e^{-p_i}\right). \end{aligned}$$

同样根据以上式子，代入消去 $u^{(1)}$：

$$A_1U_{i-1}+A_2U_i+A_3U_{i+1} = Z_0f_i+Z_1f_i^{(1)}, \tag{2.18}$$

其中：

$$\begin{aligned} A_1 &= -e^{-p_i}\left(\widehat{B}_i^1+\widehat{B}_i^2\right), \\ A_2 &= e^{-p_i}\left(\widehat{B}_i^1+\widehat{B}_i^2\right)-e^{-p_i}\left(\bar{B}_i^1+\bar{B}_i^2\right), \\ A_3 &= e^{-p_i}\left(\bar{B}_i^1+\bar{B}_i^2\right), \\ Z_0 &= \left(\bar{B}_i^1+\bar{B}_i^2\right)\left(\widehat{C}_{0,i}^1+\widehat{C}_{0,i}^2\right)-\left(\widehat{B}_i^1+\widehat{B}_i^2\right)\left(\bar{C}_{0,i}^1+\bar{C}_{0,i}^2\right), \\ Z_1 &= \left(\bar{B}_i^1+\bar{B}_i^2\right)\widehat{C}_{1,i}^2-\left(\widehat{B}_i^1+\widehat{B}_i^2\right)\bar{C}_{1,i}^2. \end{aligned}$$

如上便是新型差分方法在保留二阶项下的公式，可以发现其同样是解决一个系数为三对角矩阵的方程，因此同样可以用追赶法求解。

2.1.3　数值算例

在方程 (2.1) 中，设 $\varepsilon = 10^{-2}, \alpha(x) = 1/(1+x)$，将 $u(x) = e^x + 2^{-10^{-2}}(1+x)^{1+10^{-2}}$ 代入方程得到 $f(x)$，使用上述讨论的新型差分方法 (2.15) 和 (2.18) 对其进行求解。其中 (2.15) 结果见图2.1与图2.2。可以发现误差在 10^{-4} 量级，由于真解在 $x = 1$ 附

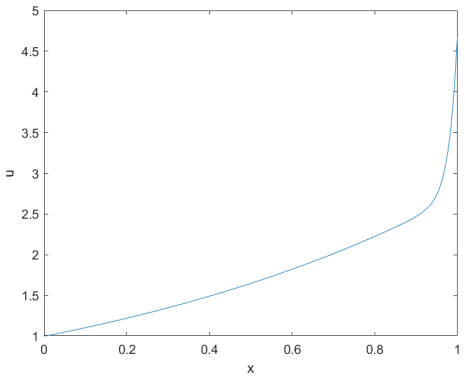

图 2.1　格式 (2.15) 的数值解

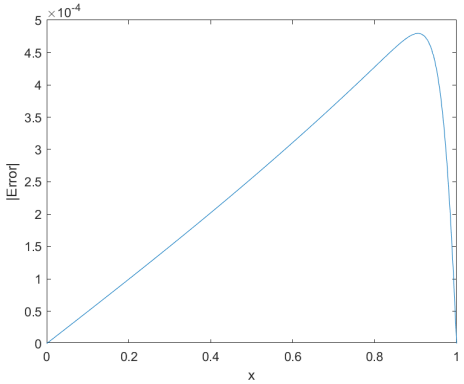

图 2.2　格式 (2.15) 的误差

近有较大跨度，因此其在 $x = 1$ 附近的误差最大。为衡量误差，本文使用无穷范数，对 $U = (U_1, U_2, \cdots, U_N)^T$，其无穷范数为：

$$\|U\|_\infty = \max_{0 \le i \le N} |U_i| . \tag{2.19}$$

不妨设网格划分数量 $N = 2^k$，则根据误差计算误差阶数的公式如下：

$$o_i = \log\left(\frac{E_{i-1}}{E_i}\right) / \log(2), \tag{2.20}$$

其中 E_i 为 $N = 2^i$ 时的误差。具体误差与阶数见表2.1。由此可见，无论是保留

表 2.1　(2.15) 格式、(2.18) 格式误差表

网格数	格式 (2.15) 的误差	收敛阶	格式 (2.18) 的误差	收敛阶
8	6.560175e-02	–	9.307574e-04	–
16	2.364026e-02	1.472488	3.867108e-05	4.589078
32	7.150644e-03	1.725101	5.399598e-05	-0.4815971
64	1.889477e-03	1.920086	1.495016e-05	1.852691
128	4.795532e-04	1.978224	3.850474e-06	1.957053
256	1.203523e-04	1.994427	9.700895e-07	1.988846
512	3.011731e-05	1.998598	2.430016e-07	1.997152

$1/\varepsilon^{n-1}$ 还是保留 $1/\varepsilon^{n-2}$ 项，算法阶数都是二阶，但保留 $1/\varepsilon^{n-2}$ 项算法精度会高一些。

2.2 交替方向迭代法

下面我们介绍传统的交替方向迭代法（ADI）[21]。

2.2.1 方法介绍

交替方向迭代法（ADI）顾名思义，是交替 x 与 y 方向迭代来对方程求解的算法。以下列泊松方程为例：

$$-\Delta u = f(x,y), \quad 0 < x, y < 1, \tag{2.21}$$

其五点差分格式为：

$$-\Delta_h u_{ij} = -\left(\frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{h^2}\right) = f_{ij}. \tag{2.22}$$

定义三个算子 L_1, L_2, L_h 分别为：

$$\begin{aligned} (L_1 u)_{ij} &= \left(-u_{i-1,j} + 2u_{ij} - u_{i+1,j}\right)/h^2, \\ (L_2 u)_{ij} &= \left(-u_{i,j-1} + 2u_{ij} - u_{i,j+1}\right)/h^2, \\ (L_h u)_{ij} &= (L_1 u)_{ij} + (L_2 u)_{ij}. \end{aligned} \tag{2.23}$$

则可以将原五点差分格式写成：

$$L_h u = L_1 u + L_2 u = f, \tag{2.24}$$

给定初值 $u^{(0)}$，对 $k = 0, 1, \cdots$, 依次求解如下方程组：

$$\begin{aligned} u^{(k+\frac{1}{2})} &= u^{(k)} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k)} - f\right), \\ u^{(k+1)} &= u^{(k+\frac{1}{2})} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k+1)} - f\right), \end{aligned} \tag{2.25}$$

其中 τ_k 为迭代步长。将上面公式 (2.25) 按层合并，即：

$$\begin{aligned} \left(I + \tau_k L_1\right) u^{(k+\frac{1}{2})} &= \left(I - \tau_k L_2\right) u^{(k)} + \tau_k f, \\ \left(I + \tau_k L_2\right) u^{(k+1)} &= \left(I - \tau_k L_1\right) u^{(k+\frac{1}{2})} + \tau_k f. \end{aligned} \tag{2.26}$$

可以发现，上面两个式子 (2.26) 的右边都是已知的，左边也都是一个三对角矩阵，即同样可以用追赶法求解。但与之前的一维情况不一样的是，这是一个求解二维方程离散后得到的线性方程组的迭代法，可以证明选取适当迭代步长后该迭代是收敛的，在迭代中评价是否收敛需要使用范数，在本文中二维情况下使用的范数是：

$$\|\mathrm{U}\| = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} |U_{ij}|^2, \tag{2.27}$$

在比较计算结果与真解的时候使用的同样是这个范数。另外，τ_k 迭代步长会影响 ADI 的收敛情况，设置得不好就不会收敛，设置得好可以收敛得更快，有不少的设置策略，可以每次迭代设置不同的值，但本文为了简单起见将其设为一个定值。经过推导可以得出结论，用以下公式确定 τ_k 的值可以收敛得最快。

$$\tau_{opt} = \frac{1}{2}h^2(\sin \pi h)^{-1}. \tag{2.28}$$

传统的交替方向迭代法是基于五点差分格式求解泊松方程的，但我们可以对其略加修改使其可以求解对流扩散方程。首先二维对流扩散方程是如下形式的：

$$-\varepsilon\Delta u + \alpha u_x + \beta u_y = f, \tag{2.29}$$

类似泊松方程那样，构造 x 与 y 方向的算子：

$$\begin{aligned}
(L_1 u)_{ij} &= \left(-u_{i-1,j} + 2u_{ij} - u_{i+1,j}\right)\varepsilon/h^2 + \alpha_{ij}(u_{i+1,j} - u_{i-1,j})/2h, \\
(L_2 u)_{ij} &= \left(-u_{i,j-1} + 2u_{ij} - u_{i,j+1}\right)\varepsilon/h^2 + \beta ij(u_{i,j+1} - u_{i,j-1})/2h, \\
(L_h u)_{ij} &= (L_1 u)_{ij} + (L_2 u)_{ij}\,.
\end{aligned} \tag{2.30}$$

然后就可以与上面讨论的求解方式完全一样了。但需要注意的是这个时候步长 τ_k 的取值与泊松方程时的取值是不一样的，与 α、β 有关系。

2.2.2　数值算例

为了验证对算法理解的正确性，接下来我们给出一个简单的数值算例。在 (2.21) 中将真解 $u = e^{pi(x+y)}\sin(2\pi y)\sin(\pi x)$ 代入方程所得到 $f(x,y)$，使用交替方向迭代法对该方程进行求解。求解结果见图2.1与图2.2。求解的具体误差与阶数

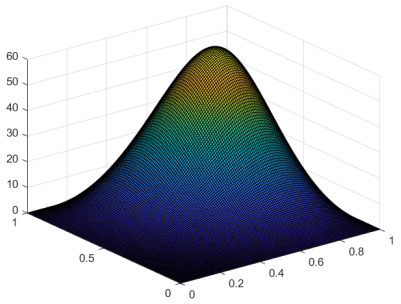

图 2.3　ADI 的数值解

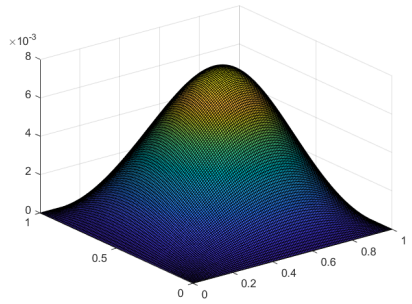

图 2.4　ADI 的误差

发现见表2.2。可以发现交替方向迭代法是一种二阶方法。对比经典的超松弛迭代法（SOR），即：

$$\begin{aligned}
u_{ij}^{(k+1)} =& u_{ij}^{(k)} - \frac{\omega}{a_{ij}}\Big(a_{ij}u_{ij}^{(k)} - a_{i-1,j}u_{i-1,j}^{(k+1)} - a_{i,j-1}u_{i,j-1}^{(k+1)} \\
& - a_{i+1,j}u_{i+1,j}^{(k)} - a_{i,j+1}u_{i,j+1}^{(k)} - f_{ij}\Big), \quad i,j,k = 0,1,\cdots,
\end{aligned} \tag{2.31}$$

表 2.2 ADI 误差表

网格数 N	ADI 格式	收敛阶	迭代次数
8	8.663578e-01	–	21
16	2.151274e-01	2.009772	41
32	5.368677e-02	2.002553	77
64	1.342337e-02	1.999820	146
128	3.372004e-03	1.993068	277

可以证明[21]，当

$$\omega_{opt} = 2/\left(1+\sqrt{1-\sin^2 \pi h}\right)$$

时其收敛速度最快。利用超松弛迭代法求解可以得到误差见表2.3。可见，超松弛

表 2.3 SOR 误差表

网格数 N	误差	收敛阶	迭代次数
8	8.663575e-01	–	28
16	2.151273e-01	2.009772	53
32	5.368567e-02	2.002581	102
64	1.341964e-02	2.000191	198
128	3.366027e-03	1.995227	377

迭代法是与交替方向同阶的方法，但迭代次数会多一些。且交替方向迭代法会从 x 方向和 y 方向分别迭代，部分情况下可以分别对其特殊处理来提高收敛速度，这是超松弛迭代法做不到的。

2.2.3 交替方向迭代法的推广

可以发现交替方向迭代法是在五点差分格式的基础上求解二维微分方程的方法，其核心思想是构造两个算子，一个在 x 方向，另一个是 y 方向，利用两个算子交替进行迭代，从而使得方程收敛到一个较好的结果。原格式的算子基于五点差分格式，不太容易推广。为了在后面章节将该方法用于新型差分格式，我们需要将其进行变形。注意到两个算子分别处理 x 方向和 y 方向，且相加就与原方程等效，因此我们将原方程按 x 与 y 方向拆开：

$$\begin{aligned} -\varepsilon u_{xx} &= cf, \\ -\varepsilon u_{yy} &= (1-c)f. \end{aligned} \tag{2.32}$$

然后分别构造两个算子：

$$\begin{aligned}
\left(L_1^{'}u\right)_{ij} &= \left(-u_{i-1,j}+2u_{ij}-u_{i+1,j}\right)/h^2-cf_{ij},\\
\left(L_2^{'}u\right)_{ij} &= \left(-u_{i,j-1}+2u_{ij}-u_{i,j+1}\right)/h^2-(1-c)f_{ij}/2,\\
\left(L_h^{'}u\right)_{ij} &= \left(L_1^{'}u\right)_{ij}+\left(L_2^{'}u\right)_{ij}=0.
\end{aligned} \tag{2.33}$$

最后的迭代计算方式为：

$$\begin{aligned}
u^{(k+\frac{1}{2})} &= u^{(k)}-\tau_k\left(L_1^{'}u^{(k+\frac{1}{2})}+L_2^{'}u^{(k)}\right),\\
u^{(k+1)} &= u^{(k+\frac{1}{2})}-\tau_k\left(L_1^{'}u^{(k+\frac{1}{2})}+L_2^{'}u^{(k+1)}\right),\quad k=0,1,\cdots.
\end{aligned} \tag{2.34}$$

可以发现虽然我们在构造的时候加入了一个参数 c，但其在后面的计算中被消去了，且该格式与传统的交替方向迭代法是一样的，但更容易进行推广。

3 在对流扩散方程中的应用

下面便是本文的重点，将之前介绍的一维新型差分方法与交替方向迭代法结合来求解二维对流扩散方程。

3.1 线性问题

首先讨论相对简单的线性情况，要将交替方向迭代法与新型差分方法相结合，有许多种方式，在本节将对其一一讨论。为了方便起见，介绍思路时仅采用新型差分方法的一阶情况，其余二阶或更高阶情况同理。

3.1.1 数值格式

① 格式一

从上一章中我们可以看到在交替方向迭代法中，我们需要将一个二维方程拆分成两个一维方程，然后构造两个算子，一个算子出来 x 方向的，另外一个处理 y 方向的，最后将两个算子结合起来求解。结合 2.2.3 节中的内容，(1.1) 可以拆分为如下形式：

$$\begin{aligned} -\varepsilon u_{xx} + \alpha u_x &= cf, \\ -\varepsilon u_{yy} + \beta u_y &= (1-c)f. \end{aligned} \tag{3.1}$$

在上述 (2.15) 的两个式中分别使用如上公式，我们便可以得到对流扩散方程新型差分格式的交替方向迭代法，给定初值 $u^{(0)}$，对 $k = 0, 1, \cdots$, 依此求

$$\begin{aligned} u^{(k+\frac{1}{2})} &= u^{(k)} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k)} \right), \\ u^{(k+1)} &= u^{(k+\frac{1}{2})} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k+1)} \right), \end{aligned} \tag{3.2}$$

其中：

$$\begin{aligned} (L_1 u)_{ij} &= A_{\alpha 1} U_{i-1,j} + A_{\alpha 2} U_{i,j} + A_{\alpha 3} U_{i+1,j} - c Y_{\alpha 0} f_{i,j}, \\ (L_2 u)_{ij} &= A_{\beta 1} U_{i,j-1} + A_{\beta 2} U_{i,j} + A_{\beta 3} U_{i,j+1} - (1-c) Y_{\beta 0} f_{i,j}, \\ (L_h u)_{ij} &= (L_1 u)_{ij} + (L_2 u)_{ij} = 0. \end{aligned} \tag{3.3}$$

按迭代层进行合并后：

$$\begin{aligned} \tau_k A_{\alpha 1} U_{i-1,j}^{(k+\frac{1}{2})} + (1 + \tau_k A_{\alpha 2}) U_{i,j}^{(k+\frac{1}{2})} + \tau_k A_{\alpha 3} U_{i+1,j}^{(k+\frac{1}{2})} &= F_{\alpha 1} + G_{\alpha 1} \\ \tau_k A_{\beta 1} U_{i,j-1}^{(k+1)} + (1 + \tau_k A_{\beta 2}) U_{i,j}^{(k+1)} + \tau_k A_{\beta 3} U_{i,j+1}^{(k+1)} &= F_{\beta 1} + G_{\beta 1}. \end{aligned} \tag{3.4}$$

其中：

$$\begin{aligned}
F_{\alpha 1} &= -\tau_k A_{\beta 1} U_{i,j-1}^{(k)} + (1-\tau_k A_{\beta 2}) U_{i,j}^{(k)} - \tau_k A_{\beta 3} U_{i,j+1}^{(k)}, \\
G_{\alpha 1} &= \tau_k (c Y_{\alpha 0} f_{i,j} + (1-c) Y_{\beta 0} f_{i,j}), \\
F_{\beta 1} &= -\tau_k A_{\alpha 1} U_{i-1,j}^{(k+\frac{1}{2})} + (1-\tau_k A_{\alpha 2}) U_{i,j}^{(k+\frac{1}{2})} - \tau_k A_{\alpha 3} U_{i+1,j}^{(k+\frac{1}{2})}, \\
G_{\beta 1} &= \tau_k (c Y_{\alpha 0} f_{i,j} + (1-c) Y_{\beta 0} f_{i,j}).
\end{aligned}$$

可以发现这样我们便在原格式中添加了一个参数 c，且该参数一般不能约去，只有在方程 α 等于 β 的时候才可以消去。在实际测试中可以发现这种方法只有当求解的方程中 α 等于 β 的时才会有不错的表现，其余的时候效果都比较差，改变参数 c 的值也没有太大作用。

② 格式二

如上所述，因为在实际测试中发现格式一有不足之处，我们需要进行改进。考虑到格式一中右边太过简单，而且

$$-\varepsilon u_{xx} + \alpha u_x = cf,$$

对于二维方程其实是不成立的（忽略了一些项）。采用如下形式：

$$-\varepsilon u_{xx} + \alpha u_x = f + \varepsilon u_{yy} - \beta u_y,$$

其实更加合理，基于以上思路我们也可以构造出一种格式：

$$\begin{aligned}
-\varepsilon u_{xx} + \alpha u_x &= f + \varepsilon u_{yy} - \beta u_y, \\
-\varepsilon u_{yy} + \beta u_y &= f + \varepsilon u_{xx} - \alpha u_x.
\end{aligned} \tag{3.5}$$

同样，类似前面所述，定义算子：

$$\begin{aligned}
(L_1 u)_{i,j} &= A_{\alpha 1} U_{i-1,j} + A_{\alpha 2} U_{i,j} + A_{\alpha 3} U_{i+1,j} - \widetilde{f}, \\
(L_2 u)_{i,j} &= A_{\beta 1} U_{i,j-1} + A_{\beta 2} U_{i,j} + A_{\beta 3} U_{i,j+1} - \widehat{f}, \\
(L_h u)_{i,j} &= (L_1 u)_{i,j} + (L_2 u)_{i,j} = 0,
\end{aligned} \tag{3.6}$$

其中：

$$\begin{aligned}
\widetilde{f} &= Y_{\beta 0}(f_{i,j} + \varepsilon \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} - \beta_{i,j} \frac{U_{i,j+1} - U_{i,j-1}}{2h}), \\
\widehat{f} &= Y_{\alpha 0}(f_{i,j} + \varepsilon \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} - \alpha_{i,j} \frac{U_{i+1,j} - U_{i-1,j}}{2h}).
\end{aligned}$$

一样的代入如下迭代式子中：

$$\begin{aligned}
u^{(k+\frac{1}{2})} &= u^{(k)} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k)} \right), \\
u^{(k+1)} &= u^{(k+\frac{1}{2})} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k+1)} \right), \quad k = 0, 1, \cdots.
\end{aligned}$$

然后按层合并：

$$\begin{aligned}\tau_k A_{\alpha1}U_{i-1,j}^{(k+\frac{1}{2})}+(1+\tau_k A_{\alpha2})U_{i,j}^{(k+\frac{1}{2})}+\tau_k A_{\alpha3}U_{i+1,j}^{(k+\frac{1}{2})}&=F_{\alpha2}+G_{\alpha2}+H_{\alpha2}\\ \tau_k A_{\beta1}U_{i,j-1}^{(k+1)}+(1+\tau_k A_{\beta2})U_{i,j}^{(k+1)}+\tau_k A_{\beta3}U_{i,j+1}^{(k+1)}&=F_{\beta2}+G_{\alpha2}+H_{\alpha2}.\end{aligned}\tag{3.7}$$

其中：

$$\begin{aligned}F_{\alpha2}&=-\tau_k A_{\beta1}U_{i,j-1}^{(k)}+(1-\tau_k A_{\beta2})U_{i,j}^{(k)}-\tau_k A_{\beta3}U_{i,j+1}^{(k)},\\ G_{\alpha2}&=\tau_k Y_{\beta0}(f_{i,j}+\varepsilon\frac{U_{i,j+1}^{(k)}-2U_{i,j}^{(k)}+U_{i,j-1}^{(k)}}{h^2}-\beta_{i,j}\frac{U_{i,j+1}^{(k)}-U_{i,j-1}^{(k)}}{2h}),\\ H_{\alpha2}&=\tau_k Y_{\alpha0}(f_{i,j}+\varepsilon\frac{U_{i,j+1}^{(k)}-2U_{i,j}^{(k)}+U_{i,j-1}^{(k)}}{h^2}-\alpha_{i,j}\frac{U_{i,j+1}-U_{i,j-1}^{(k)}}{2h}),\\ F_{\beta2}&=-\tau_k A_{\alpha1}U_{i-1,j}^{(k+\frac{1}{2})}+(1-\tau_k A_{\alpha2})U_{i,j}^{(k+\frac{1}{2})}-\tau_k A_{\alpha3}U_{i+1,j}^{(k+\frac{1}{2})},\\ G_{\beta2}&=\tau_k Y_{\beta0}(f_{i,j}+\varepsilon\frac{U_{i,j+1}^{(k+\frac{1}{2})}-2U_{i,j}^{(k+\frac{1}{2})}+U_{i,j-1}^{(k+\frac{1}{2})}}{h^2}-\beta_{i,j}\frac{U_{i,j+1}^{(k+\frac{1}{2})}-U_{i,j-1}^{(k+\frac{1}{2})}}{2h}),\\ H_{\beta2}&=\tau_k Y_{\alpha0}(f_{i,j}+\varepsilon\frac{U_{i+1,j}-2U_{i,j}^{(k+\frac{1}{2})}+U_{i-1,j}^{(k+\frac{1}{2})}}{h^2}-\alpha_{i,j}\frac{U_{i+1,j}^{(k+\frac{1}{2})}-U_{i-1,j}^{(k+\frac{1}{2})}}{2h}).\end{aligned}$$

以上便是第二种格式 (3.7)。

可以发现这种格式相比格式一 (3.4) 更加复杂，考虑的因素也更多，实际情况中也确实相比格式一 (3.4) 算法误差小不少。但可以发现该算法收敛得太慢了，即使设定合适的迭代步长 τ_k 也要迭代几千次才可以收敛到不错的结果，因此需要在该算法的基础上加以改进。

③ 格式三

由于迭代步长的估计较为麻烦，因此有必要尝试将格式二 (3.7) 中的 τ_k 去掉。因此构造了一种不带参数的格式，即在上式的基础上，改变迭代格式：

$$\begin{aligned}L_1u^{(k+\frac{1}{2})}&=-L_2u^{(k)},\\ L_2u^{(k+1)}&=-L_1u^{(k+\frac{1}{2})},\quad k=0,1,\cdots.\end{aligned}\tag{3.8}$$

展开后按层合并，即为：

$$\begin{aligned}A_{\alpha1}U_{i-1,j}^{(k+\frac{1}{2})}+A_{\alpha2}U_{i,j}^{(k+\frac{1}{2})}+A_{\alpha3}U_{i+1,j}^{(k+\frac{1}{2})}&=F_{\alpha3}+G_{\alpha3}+H_{\alpha3},\\ A_{\beta1}U_{i,j-1}^{(k+1)}+A_{\beta2}U_{i,j}^{(k+1)}+A_{\beta3}U_{i,j+1}^{(k+1)}&=F_{\beta3}+G_{\beta3}+H_{\beta3}.\end{aligned}\tag{3.9}$$

其中：

$$
\begin{aligned}
F_{\alpha 3} &= -A_{\beta 1}U_{i,j-1}^{(k)} - A_{\beta 2}U_{i,j}^{(k)} - A_{\beta 3}U_{i,j+1}^{(k)},\\
G_{\alpha 3} &= Y_{\beta 0}(f_{i,j} + \varepsilon\frac{U_{i,j+1}^{(k)} - 2U_{i,j}^{(k)} + U_{i,j-1}^{(k)}}{h^2} - \beta_{i,j}\frac{U_{i,j+1}^{(k)} - U_{i,j-1}^{(k)}}{2h}),\\
H_{\alpha 3} &= Y_{\alpha 0}(f_{i,j} + \varepsilon\frac{U_{i,+1j}^{(k)} - 2U_{i,j}^{(k)} + U_{i,j-1}^{(k)}}{h^2} - \alpha_{i,j}\frac{U_{i+1,j} - U_{i-1,j}^{(k)}}{2h}),\\
F_{\beta 3} &= -A_{\alpha 1}U_{i-1,j}^{(k+\frac{1}{2})} + A_{\alpha 2}U_{i,j}^{(k+\frac{1}{2})} - A_{\alpha 3}U_{i+1,j}^{(k+\frac{1}{2})},\\
G_{\beta 3} &= Y_{\beta 0}(f_{i,j} + \varepsilon\frac{U_{i+1,j}^{(k+\frac{1}{2})} - 2U_{i,j}^{(k+\frac{1}{2})} + U_{i-1,j}^{(k+\frac{1}{2})}}{h^2} - \beta_{i,j}\frac{U_{i+1,j}^{(k+\frac{1}{2})} - U_{i-1,j}^{(k+\frac{1}{2})}}{2h}),\\
H_{\beta 3} &= Y_{\alpha 0}(f_{i,j} + \varepsilon\frac{U_{i+1,j} - 2U_{i,j}^{(k+\frac{1}{2})} + U_{i-1,j}^{(k+\frac{1}{2})}}{h^2} - \alpha_{i,j}\frac{U_{i+1,j}^{(k+\frac{1}{2})} - U_{i-1,j}^{(k+\frac{1}{2})}}{2h}).
\end{aligned}
$$

以上便是第三种格式。但在实际中可以发现如果不加步长参数调节使用如上格式会导致算法发散了，所有必须加一个步长参数。

④ **格式四**

回忆其它传统迭代算法的改进方法，可以发现，类似超松弛迭代法（SOR）对雅可比简单迭代法的改进一样[22]，如果可以将右边的第 k 层的一些项变为 $k+\frac{1}{2}$ 层的，可能可以加速。

首先回忆格式二 (3.6)：

$$
\begin{aligned}
(L_1u)_{i,j} &= A_{\alpha 1}U_{i-1,j} + A_{\alpha 2}U_{i,j} + A_{\alpha 3}U_{i+1,j} - \widetilde{f},\\
(L_2u)_{i,j} &= A_{\beta 1}U_{i,j-1} + A_{\beta 2}U_{i,j} + A_{\beta 3}U_{i,j+1} - \widehat{f},\\
(L_hu)_{i,j} &= (L_1u)_{i,j} + (L_2u)_{i,j} = 0.
\end{aligned}
$$

其中：

$$
\begin{aligned}
\widetilde{f} &= Y_{\beta 0}(f_{i,j} + \varepsilon\frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} - \beta_{i,j}\frac{U_{i,j+1} - U_{i,j-1}}{2h}),\\
\widehat{f} &= Y_{\alpha 0}(f_{i,j} + \varepsilon\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} - \alpha_{i,j}\frac{U_{i+1,j} - U_{i-1,j}}{2h}).
\end{aligned}
$$

可以发现：

$$
\begin{aligned}
\widetilde{f} + \widehat{f} &= Y_{\beta 0}(f_{i,j} + \varepsilon\frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} - \beta_{i,j}\frac{U_{i,j+1} - U_{i,j-1}}{2h}) +\\
&\quad Y_{\alpha 0}(f_{i,j} + \varepsilon\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} - \alpha_{i,j}\frac{U_{i+1,j} - U_{i-1,j}}{2h})\\
&= Y_{\beta 0}f_{i,j} + (Y_{\beta 0} - Y_{\alpha 0})(\varepsilon\frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} - \beta_{i,j}\frac{U_{i,j+1} - U_{i,j-1}}{2h})\\
&= Y_{\alpha 0}f_{i,j} + (Y_{\alpha 0} - Y_{\beta 0})(\varepsilon\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} - \alpha_{i,j}\frac{U_{i+1,j} - U_{i-1,j}}{2h}),
\end{aligned}
\tag{3.10}
$$

利用如上公式可以对其进行变形，使得按层合并后：

$$\begin{aligned}(L_1u)_{i,j} &= A^{'}_{\alpha1}U_{i-1,j} + A^{'}_{\alpha2}U_{i,j} + A^{'}_{\alpha3}U_{i+1,j} - F^{'}_{\alpha},\\ (L_2u)_{i,j} &= A^{'}_{\beta1}U_{i,j-1} + A^{'}_{\beta2}U_{i,j} + A^{'}_{\beta3}U_{i,j+1} - F^{'}_{\beta},\\ (L_hu)_{i,j} &= (L_1u)_{i,j} + (L_2u)_{i,j} = 0.\end{aligned} \tag{3.11}$$

其中：

$$\begin{aligned}A^{'}_{\alpha1} &= \tau_k(A_{\alpha1} - (Y_{\alpha0} - Y_{\beta0})(\frac{\varepsilon}{h^2} - \frac{\alpha_{i,j}}{2h})),\\ A^{'}_{\alpha2} &= 1 + \tau_k(A_{\alpha2} + (Y_{\alpha0} - Y_{\beta0})(\frac{2\varepsilon}{h^2})),\\ A^{'}_{\alpha3} &= \tau_k(A_{\alpha3} - (Y_{\alpha0} - Y_{\beta0})(\frac{\varepsilon}{h^2} + \frac{\alpha_{i,j}}{2h})),\\ F^{'}_{\alpha} &= -\tau_kA_{\beta1}U^{(k)}_{i,j-1} + (1 - \tau_kA_{\beta2})U^{(k)}_{i,j} - \tau_kA_{\beta3}U^{(k)}_{i,j+1} + \tau_kY_{\alpha0}f_{i,j},\\ A^{'}_{\beta1} &= \tau_k(A_{\beta1} - (Y_{\beta0} - Y_{\alpha0})(\frac{\varepsilon}{h^2} - \frac{\alpha_{i,j}}{2h})),\\ A^{'}_{\beta2} &= 1 + \tau_k(A_{\beta2} + (Y_{\beta0} - Y_{\alpha0})(\frac{2\varepsilon}{h^2})),\\ A^{'}_{\beta3} &= \tau_k(A_{\beta3} - (Y_{\beta0} - Y_{\alpha0})(\frac{\varepsilon}{h^2} + \frac{\alpha_{i,j}}{2h})),\\ F^{'}_{\beta} &= -\tau_kA_{\alpha1}U^{(k+\frac{1}{2})}_{i-1,j} + (1 - \tau_kA_{\alpha2})U^{(k+\frac{1}{2})}_{i,j} - \tau_kA_{\alpha3}U^{(k+\frac{1}{2})}_{i+1,j} + \tau_kY_{\beta0}f_{i,j}.\end{aligned}$$

如上便是第四种格式。可以发现该方法的收敛速度最快而且精度也不错。

另外，基于以上格式我们也可以类似构造其它迭代求解公式，但实践可以发现用 Jacobi 迭代或 Gauss-Seidel 迭代求解均不收敛，但使用超松弛迭代法求解选用适当的迭代参数其可以做得收敛，其超松弛迭代法型的迭代公式如下：

$$U^{(k+1)}_{i,j} = U^{(k)}_{i,j} + \tau_k(F^{'}_{\alpha} + F^{'}_{\beta} + U^{(k)'}_{i,j})/(A^{'}_{\alpha2} + A^{'}_{\beta2}), \tag{3.12}$$

其中：

$$\begin{aligned}F^{'} &= F^{'}_{\alpha} + F^{'}_{\beta},\\ U^{(k)'}_{i,j} &= -A^{'}_{\alpha1}U^{(k+1)}_{i-1,j} - A^{'}_{\alpha2}U^{(k)}_{i,j} - A^{'}_{\alpha3}U^{(k)}_{i+1,j} - A^{'}_{\beta1}U^{(k+1)}_{i,j-1} - A^{'}_{\beta2}U^{(k)}_{i,j} - A^{'}_{\beta3}U^{(k)}_{i,j+1}.\end{aligned}$$

其系数与上述一致，取适当的 τ_k 同样可以收敛。

⑤ 二阶格式

讨论完一阶格式，下面便讨论在格式四 (3.11) 的基础上改为二阶格式。基于二阶新型差分方法 (2.18)，类似格式四处理方程可得：

$$\begin{aligned}(L_1u)_{i,j} &= A_{\alpha1}U_{i-1,j} + A_{\alpha2}U_{i,j} + A_{\alpha3}U_{i+1,j} - \widetilde{f},\\ (L_2u)_{i,j} &= A_{\beta1}U_{i,j-1} + A_{\beta2}U_{i,j} + A_{\beta3}U_{i,j+1} - \widehat{f},\\ (L_hu)_{i,j} &= (L_1u)_{i,j} + (L_2u)_{i,j} = 0.\end{aligned}$$

注意问题是 $\widetilde{f}$ 与 $\widehat{f}$ 的构造。首先尝试一种简单的构造，即：

$$\begin{aligned}(L_1u)_{i,j} &= A^{'}_{\alpha1}U_{i-1,j} + A^{'}_{\alpha2}U_{i,j} + A^{'}_{\alpha3}U_{i+1,j} - F^{'}_{\alpha} - G^{'}_{\alpha},\\ (L_2u)_{i,j} &= A^{'}_{\beta1}U_{i,j-1} + A^{'}_{\beta2}U_{i,j} + A^{'}_{\beta3}U_{i,j+1} - F^{'}_{\beta} - G^{'}_{\beta},\\ (L_hu)_{i,j} &= (L_1u)_{i,j} + (L_2u)_{i,j} = 0.\end{aligned} \tag{3.13}$$

其中：

$$\begin{aligned}A^{'}_{\alpha1} &= \tau_k(A_{\alpha1} - (Y_{\alpha0} - Y_{\beta0})(\frac{\varepsilon}{h^2} - \frac{\alpha_{i,j}}{2h})),\\ A^{'}_{\alpha2} &= 1 + \tau_k(A_{\alpha2} + (Y_{\alpha0} - Y_{\beta0})(\frac{2\varepsilon}{h^2})),\\ A^{'}_{\alpha3} &= \tau_k(A_{\alpha3} - (Y_{\alpha0} - Y_{\beta0})(\frac{\varepsilon}{h^2} + \frac{\alpha_{i,j}}{2h})),\\ F^{'}_{\alpha} &= -\tau_kA_{\beta1}U^{(k)}_{i,j-1} + (1 - \tau_kA_{\beta2})U^{(k)}_{i,j} - \tau_kA_{\beta3}U^{(k)}_{i,j+1} + \tau_kY_{\alpha0}f_{i,j},\\ G^{'}_{\alpha} &= Y_{\alpha1}\delta_x(f_{i,j} + \varepsilon\delta_{yy}(U_{i,j}) - \delta_y(\beta_{i,j}U_{i,j})),\\ A^{'}_{\beta1} &= \tau_k(A_{\beta1} - (Y_{\beta0} - Y_{\alpha0})(\frac{\varepsilon}{h^2} - \frac{\alpha_{i,j}}{2h})),\\ A^{'}_{\beta2} &= 1 + \tau_k(A_{\beta2} + (Y_{\beta0} - Y_{\alpha0})(\frac{2\varepsilon}{h^2})),\\ A^{'}_{\beta3} &= \tau_k(A_{\beta3} - (Y_{\beta0} - Y_{\alpha0})(\frac{\varepsilon}{h^2} + \frac{\alpha_{i,j}}{2h})),\\ F^{'}_{\beta} &= -\tau_kA_{\alpha1}U^{(k+\frac{1}{2})}_{i-1,j} + (1 - \tau_kA_{\alpha2})U^{(k+\frac{1}{2})}_{i,j} - \tau_kA_{\alpha3}U^{(k+\frac{1}{2})}_{i+1,j} + \tau_kY_{\beta0}f_{i,j},\\ G^{'}_{\beta} &= Y_{\beta1}\delta_x(f_{i,j} + \varepsilon\delta_{xx}(U_{i,j}) - \delta_x(\alpha_{i,j}U_{i,j})).\end{aligned}$$

其中的差分公式为：

$$\begin{aligned}\delta_y(U_{i,j}) &= \frac{U_{i,j+1} - U_{i,j-1}}{2h},\\ \delta_x(U_{i,j}) &= \frac{U_{i+1,j} - U_{i-1,j}}{2h},\\ \delta_y(U_{i,j})^2 &= \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2},\\ \delta_x(U_{i,j})^2 &= \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2}.\end{aligned}$$

需要注意的是其中的 $A_{\alpha1}, A_{\alpha2}, A_{\alpha3}, Y_{\alpha0}, Y_{\alpha1}, A_{\beta1}, A_{\beta2}, A_{\beta3}, Y_{\beta0}, Y_{\beta1}$ 均用二阶格式 (2.18) 中的计算方式，但最后发现效果与格式四几乎一样，还可能略逊于格式四。分析之后可以发现 Z_1 这一项是很小的，大概是 10^{-8}，可能是因为格式的构造方法导致高阶项被忽略，可能这种方案不适用于二阶格式。

3.1.2 数值算例

推导完公式后便需要编程实现具体的算法。首先本文的算法全是在 Matlab 上面写的，虽然从效率方面来说 C++ 速度会更快，但从可读性以及符号计算等方面考虑 Matlab 比 C++ 更容易上手。核心算法就是把公式写到程序里面，但有几个细节可以注意。

(1) Matlab 没有提供三对角矩阵的求解算法，这个需要自己写，简单的调用矩阵求逆或 Matlab 自带的线性方程组求解函数太慢了。

(2) 尽量不要动态改变矩阵大小，应该先把矩阵大小初始化，这样可以提高运行效率。

(3) 减少重复计算，合理规划运算结构可以提供运行效率。

另外，注意到交替方向迭代法的核心格式 (2.34)：

$$\begin{aligned} u^{(k+\frac{1}{2})} &= u^{(k)} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k)} - f \right), \\ u^{(k+1)} &= u^{(k+\frac{1}{2})} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k+1)} - f \right), \quad k = 0, 1, \cdots . \end{aligned}$$

其中有一个 τ_k，其会影响到算法的收敛速度，如果是经典的五点差分格式的交替方向迭代法我们可以通过分析矩阵特征根的形式得出最佳的数值，但如果是非经典的交替方向迭代法那这个值是不容易得出来的。特别是对于本文使用的这种新型差分方法，很难分析出 τ_k 的取值，因此我们需要找别的方法来求解 τ_k，下面便提出一种近似求解 τ_k 的偏工程的算法。为了估计 τ_k 的最优值，我们不妨将原算法看成一个关于 τ_k 和迭代次数 n 的函数，即：

$$F(\tau_k, n) = \|U_n - U_{n-1}\|, \tag{3.14}$$

其中 $U_n = LU_{n-1}$，L 为迭代算子。其函数值代表迭代参数为 τ_k、迭代 n 次的收敛情况。不妨假设存在 τ，使得对任意 $n \in N$、任意 $\tau_k \in R$，有 $F(\tau, n) <= F(\tau_k, n)$，即 τ 为使迭代收敛最快的参数。故我们可以先取一个较小的 n，在这种情况下估计出 τ，然后便可以在 n 较大的情况下依然可以以较快的速度收敛。固定较小 n 估计 τ 时，可以采用常用的二分法，将 $F(\tau_k, n+1) - F(\tau_k, n)$ 近似看成在点 $(\tau_k, n+1)$ 的导数，以此二分来查找最小值。

为验证之前思路的正确性，下面便求解一个具体算例。考虑如下方程：

$$-10^{-2}\Delta u + (x+1)u_x + (x+2y+1)u_y = f(x),$$

其中右边的 $f(x)$ 为将 $u = (e^x + 2^{-10^{-2}}(1+x)^{1+10^{-2}}) \cdot (e^y + 2^{-10^{-2}}(1+y)^{1+10^{-2}})$ 代入方程所得，可见其二阶导的系数为 10^{-2}，即为对流占优的方程。基于格式四 (3.11) 求解的结果见图3.1与图3.2。可以发现该函数在点 $(1,1)$ 处有较大突变，且在该点

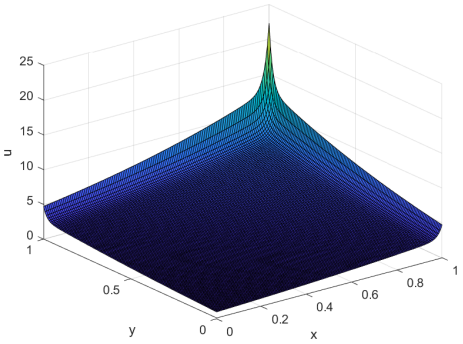

图 3.1 格式 (3.11) 的数值解

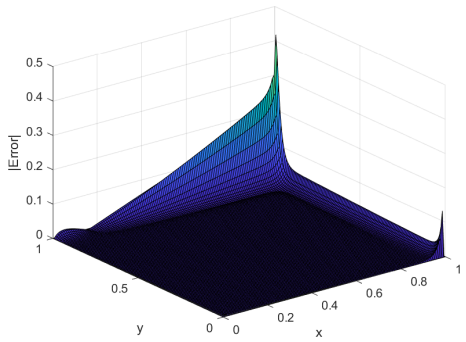

图 3.2 格式 (3.11) 的误差

附近也有较大误差。其计算的误差见表3.1。对比使用超松弛迭代法的公式 (3.12)

表 3.1 (3.11) 格式误差表

网格数	误差	收敛解	迭代参数 τ_k	迭代次数
32	2.586e-01	–	9.993e+3	164
64	5.213e-02	2.31	9.993e+3	61
128	5.392e-03	3.27	9.993e+3	60
256	2.398e-03	1.16	9.993e+3	54

所得的误差（见表3.2），可以发现对比超松弛迭代法迭代法，使用交替方向迭代法

表 3.2 (3.12) 格式误差表

网格数	误差	收敛阶数	迭代参数 τ_k	迭代次数
32	2.619e-01	–	0.1875	138
64	7.702e-02	1.76	0.1875	294
128	1.860e-02	2.04	0.4375	208
256	4.273e-03	2.12	0.4375	918

可以有更少的迭代次数以及较高的精度。我们再与经典五点差分下的交替方向迭

代法做对比，其误差见表3.3，可见，对于这一类 ε 较小的方程，经典五点差分下

表 3.3 ADI 求解误差表

网格数	误差	收敛阶数	迭代参数 τ_k	迭代次数
32	8.513e-01	–	17.00	29
64	8.395e-01	0.02	92.32	23
128	8.186e-01	0.04	315.3	27
256	8.437e-01	-0.04	315.3	70

的交替方向迭代法已经失效了。

由上述讨论可知，故新型差分方法相比于经典差分法对求解对流占优的对流扩散方程有优越性，且交替方向迭代法相比超松弛迭代法收敛得更快，故使用经典差分的交替方向迭代求解法是合理的。

3.2 非线性问题

在本节中，我们将前述方程推广到一类非线性对流扩散方程————Burgers 方程[23] 中。

3.2.1 一维问题

我们先考虑一维情况下的 Burgers 方程：

$$-\varepsilon\frac{\partial^2 u}{\partial x^2}+u(x)\frac{\partial u}{\partial x}=f(x),\quad x\in\Omega=(0,1). \tag{3.15}$$

对 (3.15) 做如下线性化处理：

$$-\varepsilon\frac{\partial^2 U^{k+1}}{\partial x^2}+U^k(x)\frac{\partial U^{k+1}}{\partial x}=f(x),$$

则可将格式 (3.11) 直接运用于上述方程，得到求解一维非线性方程的格式：

$$A_1(U^k)U_{i-1}^{k+1}+A_2(U^k)U_i^{k+1}+A_3(U^k)U_{i+1}^{k+1}=Z_0 f_i, \tag{3.16}$$

其中 $A_1(U^k),A_2(U^k),A_2(U^k)$ 由 $U^k=(U_1^k,U_2^k,\cdots,U_N^k)^T$ 来更新，在每个迭代步骤中求解 (3.16) 来计算新的 U^{k+1} 如此迭代，直到无穷范数收敛。所以其实求解非线性问题是与求解线性问题相似的，系数不断更新就可以了，不同之处在于原来求解一个方程组就可以了，现在需要求解很多个方程组，这样迭代下来直到收敛。

下面计算一个算例算例，考虑下面的一个非线性一维对流扩散方程：

$$-10^{-2}\frac{\partial^2 u}{\partial x^2}+u\frac{\partial u}{\partial x}=f(x),$$
$$x\in\Omega=(0,1).$$

其中 $f(x)$ 为将解 $u(x)=e^x+2^{-10^{-2}}(1+x)^{1+10^{-2}}$ 代入改方程所得到的值，下面便用上面讨论的非线性新型差分方法对其进行求解。求解误差结果见表3.4，可见非

表 3.4 (3.16) 格式误差表

网格数	误差	收敛阶	迭代次数
128	1.706669e-01	–	17
256	7.210036e-02	1.868634	17
512	1.974344e-02	1.947628	17
1024	5.118330e-03	1.985505	17

线性新型差分方法精度为二阶，且在网格划分较小时仍然有较高精度。

3.2.2 二维标量问题

接下来，我们来看二维的标量 Burgers 方程，即：

$$-\varepsilon\left(u_{xx}+u_{yy}\right)+uu_x+uu_y=f, \tag{3.17}$$

同样的用新型差分方法与 ADI 进行求解。先回忆新型差分方法与 ADI 结合的最简单的情况，也就是方法一：

$$\begin{aligned}
&(L_1u)_{ij}=A_{\alpha1}U_{i-1,j}+A_{\alpha2}U_{i,j}+A_{\alpha3}U_{i+1,j}-cY_{\alpha0}f_{i,j},\\
&(L_2u)_{ij}=A_{\beta1}U_{i,j-1}+A_{\beta2}U_{i,j}+A_{\beta3}U_{i,j+1}-(1-c)Y_{\beta0}f_{i,j},\\
&(L_hu)_{ij}=(L_1u)_{ij}+(L_2u)_{ij}=0.
\end{aligned}$$

迭代方法：

$$\begin{aligned}
&u^{(k+\frac{1}{2})}=u^{(k)}-\tau_k\left(L_1u^{(k+\frac{1}{2})}+L_2u^{(k)}\right),\\
&u^{(k+1)}=u^{(k+\frac{1}{2})}-\tau_k\left(L_1u^{(k+\frac{1}{2})}+L_2u^{(k+1)}\right),\quad k=0,1,\cdots.
\end{aligned}$$

仿照一维非线性的做法，每轮迭代更新系数，即 A_1^k,A_2^k,A_2^k 由 U^k 来更新，再用 A_1^k,A_2^k,A_2^k 计算出 U^{k+1}，具体算子为：

$$\begin{aligned}
&\left(L_1^ku\right)_{ij}=A_{\alpha1}^kU_{i-1,j}^{k+1}+A_{\alpha2}^kU_{i,j}^{k+1}+A_{\alpha3}^kU_{i+1,j}^{k+1}-cY_{\alpha0}^kf_{i,j},\\
&\left(L_2^ku\right)_{ij}=A_{\beta1}^kU_{i,j-1}^{k+1}+A_{\beta2}^kU_{i,j}^{k+1}+A_{\beta3}^kU_{i,j+1}^{k+1}-(1-c)Y_{\beta0}^kf_{i,j},\\
&\left(L_h^ku\right)_{ij}=\left(L_1^ku\right)_{ij}+\left(L_2^ku\right)_{ij}=0.
\end{aligned} \tag{3.18}$$

这样就可以实现非线性情况的求解。其余的结合方法也可以类似推广，在本文算例中使用的是格式四 (3.11) 的推广。

下面计算一个算例算例，考虑下面的一个非线性二维对流扩散方程：

$$-\varepsilon\Delta u + uu_x + uu_y = f(x)$$

其中右边的 $f(x)$ 为将 $u = \sin(3\pi x)\sin(\pi y)$ 代入所得。利用上述算法对其进行求解，求解结果见图3.3与图3.4。求解的具体误差与阶数发现见表3.5。可见其依然是

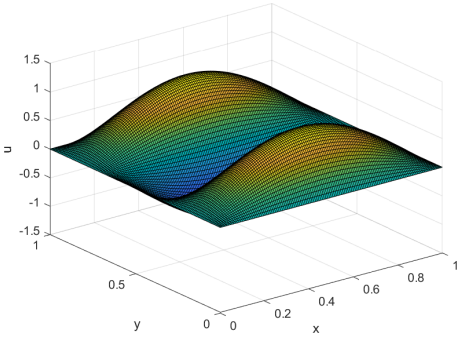

图 3.3 格式 (3.18) 的数值解

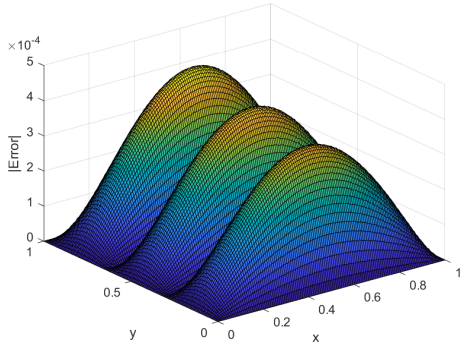

图 3.4 格式 (3.18) 的误差

表 3.5 (3.18) 格式误差表

网格数	误差	误差阶数	迭代参数 τ_k	迭代次数
8	5.5486e-02	–	1.4330	80
16	1.3276e-02	2.0633	163.8157	21
32	3.2832e-03	2.0156	315.3402	24
64	8.1875e-04	2.0036	315.3402	119
128	2.0475e-04	1.9996	3.1787e+04	31

二阶收敛的，且同样可以较为快速的收敛。

3.2.3 二维向量问题

最后我们还可以将非线性的情况推广到向量，也就是偏微分方程组的情况。求解的方程组如下：

$$\begin{aligned} -\varepsilon\left(u_{xx}+u_{yy}\right)+uu_x+vu_y &= f, \\ -\varepsilon\left(v_{xx}+v_{yy}\right)+uv_x+vv_y &= g. \end{aligned} \tag{3.19}$$

求解方法与上述的非线性方程的求解方法是类似的，一样考虑格式一的推广：

$$\begin{aligned}
\left(L_1^k u\right)_{ij} &= A_{\alpha 1}^k U_{i-1,j}^{k+1} + A_{\alpha 2}^k U_{i,j}^{k+1} + A_{\alpha 3}^k U_{i+1,j}^{k+1} - cY_{\alpha 0}^k f_{i,j}, \\
\left(L_2^k u\right)_{ij} &= A_{\beta 1}^k U_{i,j-1}^{k+1} + A_{\beta 2}^k U_{i,j}^{k+1} + A_{\beta 3}^k U_{i,j+1}^{k+1} - (1-c)Y_{\beta 0}^k f_{i,j}, \\
\left(L_h^k u\right)_{ij} &= \left(L_1^k u\right)_{ij} + \left(L_2^k u\right)_{ij} = 0.
\end{aligned} \tag{3.20}$$

$$\begin{aligned}
\left(L_1^k v\right)_{ij} &= A_{\alpha 1}^k V_{i-1,j}^{k+1} + A_{\alpha 2}^k V_{i,j}^{k+1} + A_{\alpha 3}^k V_{i+1,j}^{k+1} - cY_{\alpha 0}^k g_{i,j}, \\
\left(L_2^k v\right)_{ij} &= A_{\beta 1}^k V_{i,j-1}^{k+1} + A_{\beta 2}^k V_{i,j}^{k+1} + A_{\beta 3}^k V_{i,j+1}^{k+1} - (1-c)Y_{\beta 0}^k g_{i,j}, \\
\left(L_h^k v\right)_{ij} &= \left(L_1^k v\right)_{ij} + \left(L_2^k v\right)_{ij} = 0.
\end{aligned} \tag{3.21}$$

迭代方法：

$$\begin{aligned}
u^{(k+\frac{1}{2})} &= u^{(k)} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k)}\right), \\
v^{(k+\frac{1}{2})} &= v^{(k)} - \tau_k \left(L_1 v^{(k+\frac{1}{2})} + L_2 v^{(k)}\right), \\
u^{(k+1)} &= u^{(k+\frac{1}{2})} - \tau_k \left(L_1 u^{(k+\frac{1}{2})} + L_2 u^{(k+1)}\right), \\
v^{(k+1)} &= v^{(k+\frac{1}{2})} - \tau_k \left(L_1 v^{(k+\frac{1}{2})} + L_2 v^{(k+1)}\right), \quad k = 0, 1, \cdots.
\end{aligned} \tag{3.22}$$

只不过其中 A_1^k, A_2^k, A_2^k 由 U^k 与 V^k 来更新。另外迭代中使用的范数是：

$$\|U + V\| = \frac{\|U\| + \|V\|}{2}, \tag{3.23}$$

其中 $\|U\|$ 与 $\|V\|$ 为之前提到的二维下的范数。这样便可以实现向量非线性情况下的求解。其余的结合方法也可以类似推广，在本文算例中使用的是格式四 (3.11) 的推广。同样我们也可以采用超松弛迭代法求解，具体思路与 (3.12) 一样。

下面来具体求解一个向量非线性的方程，考虑下面这个偏微分方程：

$$\begin{aligned}
-\varepsilon\left(u_{xx} + u_{yy}\right) + uu_x + vu_y &= f(x), \\
-\varepsilon\left(v_{xx} + v_{yy}\right) + uv_x + vv_y &= g(x).
\end{aligned}$$

其中：

$$\begin{aligned}
\varepsilon &= 1, \\
f(x) &= \pi^2\cos(\pi x) + \pi^2\sin(\pi x) + (\cos(\pi x) + \sin(\pi x))(\pi\cos(\pi x) - \pi\sin(\pi x)), \\
g(x) &= x + y + \cos(\pi x) + \sin(\pi x).
\end{aligned}$$

其精确解为：$u = \cos(\pi x) + \sin(\pi x), v = x + y$。利用上面讨论的向量非线性方法对其进行求解，其中，$u(x, y)$ 的求解结果见图3.5与图3.6，$v(x, y)$ 的求解结果见图3.7与图3.8。 求解的具体误差见表3.6。可见即使是向量非线性方程该方法依然

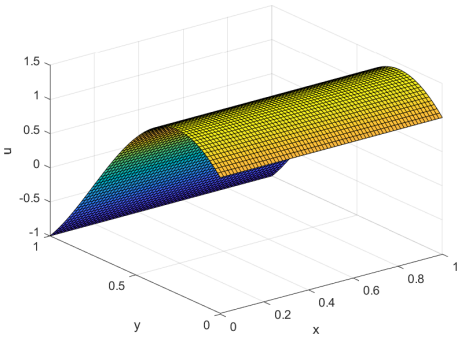

图 3.5　格式 (3.22) 中 u 的数值解

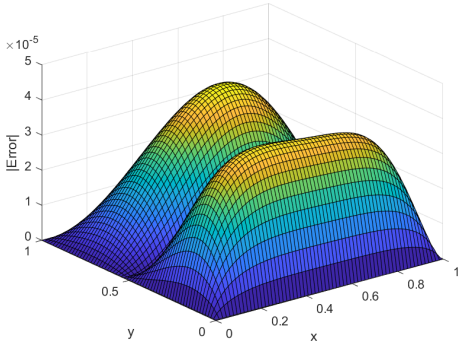

图 3.6　格式 (3.22) 中 u 的误差

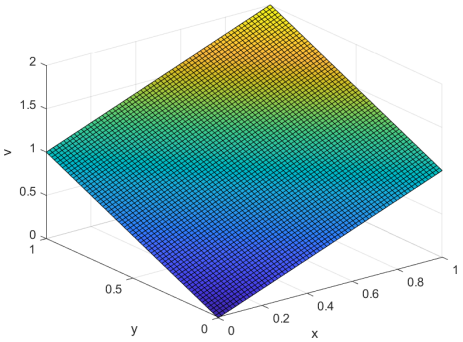

图 3.7　格式 (3.22) 中 v 的数值解

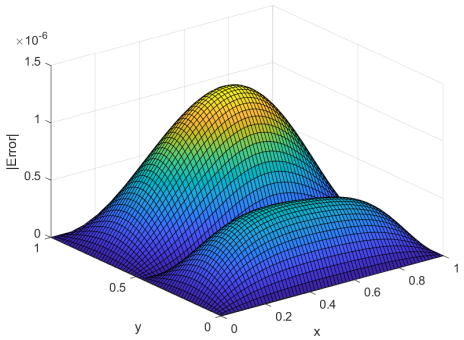

图 3.8　格式 (3.22) 中 v 的误差

是二阶收敛的。另外。同样的，我们可以用传统的交替方向迭代法做类似的处理使得其可以求解上述方程，其误差结果见表3.7。可见，确实可以说明一般情况下，新的差分方法比传统的交替方向迭代法一般情况下好上一个常数。另外如果用超松弛迭代法，部分误差结果可见表3.8，网格划分数 N 超过 32 后就会好非常长的时间才收敛，可见交替方向迭代法的收敛速度确实是优于超松弛迭代法。

表 3.6 (3.22) 格式误差表

网格数	误差	误差阶数	迭代参数 τ_k	迭代次数
8	1.4883e-03	–	26.1799	24
16	3.7944e-04	1.9717	129.3649	44
32	9.5148e-05	1.9956	315.3402	143
64	1.1035e-05	3.1081	315.3402	224

表 3.7 ADI 格式误差表

网格数	误差	误差阶数	迭代参数 τ_k	迭代次数
8	1.8575e-03	–	1.5399	19
16	4.7277e-04	1.9742	5.0481	31
32	1.1470e-04	2.0433	5.0481	63
64	1.4031e-05	3.0312	5.0481	214

表 3.8 SOR 格式误差表

网格数	误差	误差阶数	迭代参数 τ_k	迭代次数
8	1.4936e-03	–	0.09	778
16	3.5742e-04	2.0631	0.09	2679
32	5.1414e-05	2.7973	0.09	9222

最后我们讨论一个比较复杂的方程，在二维向量 Burgers 方程 (3.19) 中，令 $\varepsilon = 10^{-2}$，根据解 $u(x,y) = (e^x + 2^{-10^{-2}}(1+x)^{1+10^{-2}}) \cdot (e^y + 2^{-10^{-2}}(1+y)^{1+10^{-2}})$ 与 $v(x,y) = \exp(y-x) + 2^{-1/10^{-2}}(1+y)^{1+1/10^{-2}}$ 构造 $f(x,y)$ 与 $g(x,y)$，然后求解该方程组，该方程是对流占优的且其解含突变，在使用新型差分的交替方向迭代法求解中，$u(x,y)$ 的结果见图3.9与图3.10，$v(x,y)$ 的结果见图3.11与图3.12。

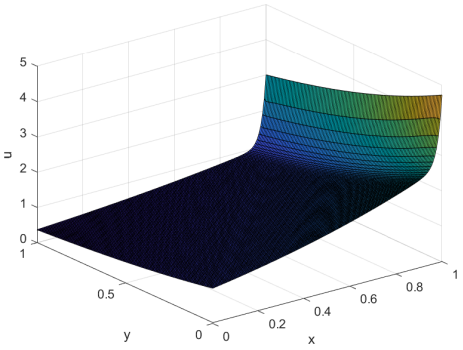

图 3.9 格式 (3.22) 中 u 的数值解

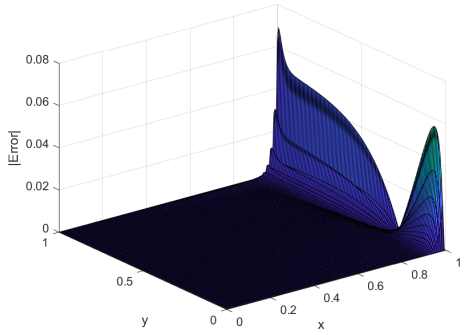

图 3.10 格式 (3.22) 中 u 的误差

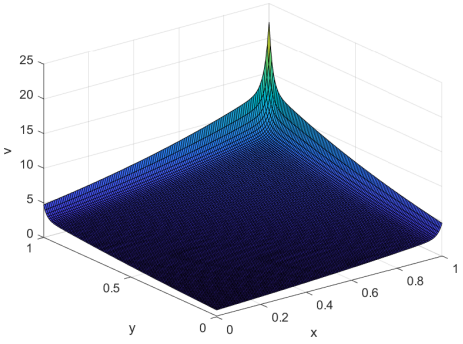

图 3.11 格式 (3.22) 中 v 的数值解

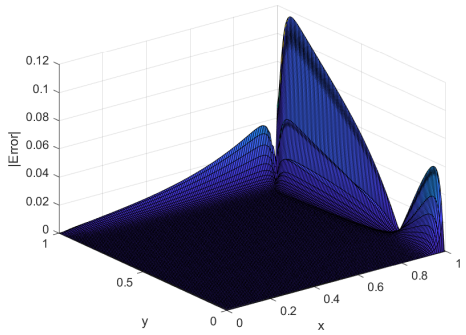

图 3.12 格式 (3.22) 中 v 的误差

比较在这种复杂情况下各个方法的优劣见表3.9。可见无论是在求解时间还是

表 3.9 不同方法的误差表

方法名	网格数	误差	迭代参数 τ_k	迭代次数
新型差分与 ADI	128	5.913e-03	120	553
新型差分与 SOR	128	5.950e-03	0.09	2431
经典差分与 ADI	128	9.777e-01	10	343

在精度上新型差分的交替方向迭代法的表现都是不错的。

4 结论与展望

4.1 研究总结

本文首先介绍了一类新型差分方法与交替方向迭代法，其次对传统的交替方向迭代法进行了推广，使得其可以与新方法结合求解对流扩散方程，然后尝试了几种结合方式，比较之后得到了一种结果比较不错的二阶收敛方案。另外，由于交替方向迭代法有一个参数 τ_k，在与新方法的结合中很难用传统的计算特征根等方法得到一个合理的值使其快速收敛，因此本文还提出了一种估计 τ_k 的算法，在求解上由于其是三对角矩阵因此采用追赶法使得可以在 $O(n)$ 时间内求解。本文也将该方法推广到了求解非线性的对流扩散方程，同样得到了不错的结果，最后，本文比较了新型差分的交替方向迭代法、新型差分的超松弛迭代法、经典差分的交替方向迭代法，证明了新型差分的交替方向迭代法在求解对流扩散方程上精度与速度的优越性。

4.2 需要进一步开展的工作

本文分析的是定常情况下的二维对流扩散方程，可以将该思想推广到其对应的非定常问题。在估计迭代参数 τ_k 上，本文的算法仅可做得大致估计，应该有更准确的估计方法。另外本文使用的新方法格式是一阶的，即仅保留了 $1/\varepsilon^{n-1}$ 项，而在新方法中保留 $1/\varepsilon^{n-2}$ 精度会更高，但直接简单的在新方法与交替方向迭代法中加上高阶项求解精度并不会增加，怎样构造一种新的格式才使得新方法具有更高的精度也是可以进一步讨论的问题。

致 谢

一转眼，又到了毕业季，四年的大学生活终于即将结束，已经放弃保研的我，甚至是学生生涯即将结束，多少有点感慨。即使是现在我也不敢肯定放弃保研直接工作是一个正确的决定，想当初高考结束不顾父母劝阻毅然选择数学系的我，那个在跟朋友讨论读计算机研究生的我，估计是无法想象我这时的选择。

人确实是会改变的，当初抱着试一试的想法去尝试投了一些简历，由于个人比较喜欢游戏，所以都是投的游戏公司，凭借自己还算扎实的算法基础收到了腾讯等大厂的 offer，薪水也还行，最后就选择了一份图形工程师的工作，便放弃保研了。

另外之前在补技术栈时有接触到图形渲染，感觉这个很有意思，特别是物理模拟方面，游戏中的物理模拟全是建立在物理模型的基础上的，并且游戏一般都是实时渲染因此对算法的时间性要求很高，其中不免需要用一些时间复杂度低的算法求解偏微分方程，特别是流体方面的模拟，牵扯到的知识更多，且全都需要解流体力学的相关方程。

因此当毕业选题的时候发现有解流体力学的对流扩散方程的选题时当场就选了，所以非常感谢王坤老师给的选题，另外王坤老师也确实是一位非常负责的老师，经常会与我们讨论，为我解答了很多疑惑以及求解方程的思路与改进方向，让我在写毕业论文的时候，确实多少有点做科研的感觉，也许读研就会是这种生活吧。

在写毕业论文的时候也用到了很多微分方程以及数值分析的东西，算是一次综合的学以致用，并且在之前找工作的时候，凭借扎实的高等代数基础与数学分析基础也获得过面试官的称赞，编程算法是基础，数学知识是加分项，正因如此我才能在激烈的就业环境中依然可以找到不错的工作，而这离不开重庆大学数统学院为我们上课的认真负责的老师，不得不说重庆大学还是有不少优秀的老师，让我们有更广阔的视野，以至于看数学结构时可以像看水晶一样清晰透彻。我一直都认为数学是非常有用且漂亮的，任何事物的底层都逃不过数学，即使以后离开学校了我也不会放弃数学相关的学习，算是一种爱好吧。

最后，我个人对这篇论文的结果还是挺满意的，整个论文的进程还算是一帆风顺，我自认也做出了一定的成果，在此再次感谢我的指导老师王坤老师。愿以此文，为大学生涯画上完美的句号。

参考文献

[1] HE X, WANG K. New finite difference methods for singularly perturbed convection-diffusion equations[J]. Taiwanese Journal of Mathematics, 2018, 22(4): 949–978.

[2] HUYAKORN S P. Solution of steady-state, convective transport equation using an upwind finite element scheme[J]. Applied Mathematical Modelling, 1977, 1(4): 187–195.

[3] AXELSSON O, NIKOLOVA M. Adaptive refinement for convection-diffusion problems based on a defect-correction technique and finite difference method[J]. Computing, 1997, 58(1): 1–30.

[4] ROOS H G. Robust numerical methods for singularly perturbed differential equations: a survey covering 2008–2012[J]. International Scholarly Research Notices, 2012, 2012: 30.

[5] SUN H, ZHANG J. A high-order finite difference discretization strategy based on extrapolation for convection diffusion equations[J]. Numerical Methods for Partial Differential Equations: An International Journal, 2004, 20(1): 18–32.

[6] CHIU P H, SHEU T W. On the development of a dispersion-relation-preserving dual-compact upwind scheme for convection–diffusion equation[J]. Journal of Computational Physics, 2009, 228(10): 3640–3655.

[7] CHU P C, FAN C. A three-point combined compact difference scheme[J]. Journal of Computational Physics, 1998, 140(2): 370–399.

[8] RADHAKRISHNA PILLAI A. Fourth-order exponential finite difference methods for boundary value problems of convective diffusion type[J]. International Journal for Numerical Methods in Fluids, 2001, 37(1): 87–106.

[9] TIAN Z, DAI S. High-order compact exponential finite difference methods for convection–diffusion type problems[J]. Journal of Computational Physics, 2007, 220(2): 952–974.

[10] WANG K, WONG Y S. Pollution-free finite difference schemes for non-homogeneous helmholtz equation.[J]. International Journal of Numerical Analysis & Modeling, 2014, 11(4): 787–815.

[11] WANG K, WONG Y S, DENG J. Efficient and accurate numerical solutions for helmholtz equation in polar and spherical coordinates[J]. Communications in Computational Physics, 2015, 17 (3): 779–807.

[12] BURGERS J M. A mathematical model illustrating the theory of turbulence[J]. Advances in Applied Mechanics, 1948, 1: 171–199.

[13] KHATER A, TEMSAH R, HASSAN M. A chebyshev spectral collocation method for solving burgers'-type equations[J]. Journal of Computational and Applied Mathematics, 2008, 222(2): 333–350.

[14] JAIN P, HOLLA D. Numerical solutions of coupled burgers' equation[J]. International Journal of Non-Linear Mechanics, 1978, 13(4): 213–222.

[15] NEE J, DUAN J. Limit set of trajectories of the coupled viscous burgers' equations[J]. Applied Mathematics Letters, 1998, 11(1): 57–61.

[16] MITTAL R, JIWARI R. Differential quadrature method for two-dimensional burgers' equations [J]. International Journal for Computational Methods in Engineering Science and Mechanics, 2009, 10(6): 450–459.

[17] PERKO J, SARLER B, OTHERS. Weight function shape parameter optimization in meshless methods for non-uniform grids[J]. Computer Modeling in Engineering and Sciences, 2007, 19 (1): 55.

[18] ZHANG W, ZHANG C, XI G. An explicit chebyshev pseudospectral multigrid method for incompressible navier–stokes equations[J]. Computers & Fluids, 2010, 39(1): 178–188.

[19] ARMINJON P, BEAUCHAMP C. Numerical solution of burgers' equations in two space dimensions[J]. Computer Methods in Applied Mechanics and Engineering, 1979, 19(3): 351–365.

[20] EL-SAYED S M, KAYA D. On the numerical solution of the system of two-dimensional burgers' equations by the decomposition method[J]. Applied Mathematics and Computation, 2004, 158 (1): 101–109.

[21] 李荣华, 刘播. 微分方程数值解法[M]. 北京: 高等教育出版社, 2009.

[22] 杨大地, 王开荣. 数值分析[M]. 北京: 高等教育出版社, 2010.

[23] MITTAL R, TRIPATHI A. Numerical solutions of two-dimensional burgers' equations using modified bi-cubic b-spline finite elements[J]. Engineering Computations, 2015, 32: 1275–1306.

附 录

详细代码见 https://github.com/yangg1g/My-Convection-Diffusion-Equation。

A. 格式四核心函数代码

```
1
2     function [p,e,x,y,u,i,et]=Solve(n,ep,t,k)
3     div1=0.5;
4     h=1/n;
5     %t=1e+5;
6     %t=2e+3;
7     x=0:h:1;
8     y=0:h:1;
9     a=zeros(n+1,n+1);
10    b=zeros(n+1,n+1);
11    f=zeros(n+1,n+1);
12    u0=ones(n+1,n+1)*1e-5;
13    u_=zeros(n+1,n+1);
14    ux=zeros(n+1,n+1);
15    uxx=zeros(n+1,n+1);
16    uy=zeros(n+1,n+1);
17    uyy=zeros(n+1,n+1);
18    e=1;
19    for i=1:n+1
20        for j=1:n+1
21            a(i,j)=aexact(x(i),y(j));
22            b(i,j)=bexact(x(i),y(j));
23            f(i,j)=fexact(x(i),y(j));
24            p(i,j)=uexact(x(i),y(j));
25            ux(i,j)=uxexact(x(i),y(j));
```

```
            uxx(i,j)=uxxexact(x(i),y(j));
            uy(i,j)=uyexact(x(i),y(j));
            uyy(i,j)=uyyexact(x(i),y(j));
        end
    end
    for i=1:n+1
        u0(1,i)=p(1,i);
        u0(n+1,i)=p(n+1,i);
        u0(i,1)=p(i,1);
        u0(i,n+1)=p(i,n+1);
    end
    u=u0;
    A1=zeros(1,n-1);
    A2=zeros(1,n-1);
    A3=zeros(1,n-1);
    Y=zeros(1,n-1);
    for i=1:k
        for k=2:n
            for j=2:n
                aix=a(j,k);
                rix=h*aix/e;
                Yx=Y0_1(rix,aix,e);
                A1x=A1_1(rix,aix,e);
                A2x=A2_1(rix,aix,e);
                A3x=A3_1(rix,aix,e);

                aiy=b(j,k);
                riy=h*aiy/e;
                Yy=Y0_1(riy,aiy,e);
                A1y=A1_1(riy,aiy,e);
                A2y=A2_1(riy,aiy,e);
                A3y=A3_1(riy,aiy,e);

                fi=Yy*f(j,k);
```

```
60
61                  A1(j-1)=t*A1x-t*(Yy-Yx)*(e/h/h+aix/2/h);
62                  A2(j-1)=1+t*A2x+t*(Yy-Yx)*2*e/h/h;
63                  A3(j-1)=t*A3x-t*(Yy-Yx)*(e/h/h-aix/2/h);
64
65                  Y(j-1)=u0(j,k)-t*(A1y*u0(j,k-1)+A2y*u0(j,k)
    +A3y*u0(j,k+1))+t*fi;
66              end
67              Y(1)=Y(1)-A1(1)*u0(1,k);
68              Y(n-1)=Y(n-1)-A3(n-1)*u0(n+1,k);
69              u(2:n,k)=zhuiganfa(A1(2:n-1),A2,A3(1:n-2),Y);
70          end
71          for j=2:n
72              for k=2:n
73                  aix=a(j,k);
74                  rix=h*aix/e;
75                  Yx=Y0_1(rix,aix,e);
76                  A1x=A1_1(rix,aix,e);
77                  A2x=A2_1(rix,aix,e);
78                  A3x=A3_1(rix,aix,e);
79
80                  aiy=b(j,k);
81                  riy=h*aiy/e;
82                  Yy=Y0_1(riy,aiy,e);
83                  A1y=A1_1(riy,aiy,e);
84                  A2y=A2_1(riy,aiy,e);
85                  A3y=A3_1(riy,aiy,e);
86
87                  fi=Yx*f(j,k);
88
89                  A1(k-1)=t*A1y-t*(Yx-Yy)*(e/h/h+aiy/2/h);
90                  A2(k-1)=1+t*A2y+t*(Yx-Yy)*2*e/h/h;
91                  A3(k-1)=t*A3y-t*(Yx-Yy)*(e/h/h-aiy/2/h);
92
```

```
93                      Y(k-1)=u(j,k)-t*(A1x*u(j-1,k)+A2x*u(j,k)+
    A3x*u(j+1,k))+t*fi;
94                  end
95                  Y(1)=Y(1)-A1(1)*u(j,1);
96                  Y(n-1)=Y(n-1)-A3(n-1)*u(j,n+1);
97                  u0(j,2:n)=zhuiganfa(A1(2:n-1),A2,A3(1:n-2),Y);
98              end
99              et(i)=mynorm(u0,u_,h);
100             if(et(i)<ep)
101                 break;
102             end
103             if(isnan(et(i)))
104                 u0=u_;
105                 break;
106             end
107             u_=u0;
108     end
109     u=u0;
110     e=mynorm(u0,p,h);
111     end
```

B. 部分其它核心函数代码

```
1
2       function res=P_h_1_i(ri,ai,e)
3       res=e/ai*(1-exp(-ri));
4       end
5
6       function res=P_p_1_i(ri,ai,e)
7       res=e/ai*(exp(-2*ri)-exp(-ri));
8       end
9
10      function res=Q_h_1_0i(ri,ai,e)
```

```
11      res=-e/ai^2*(1-exp(-ri)-ri*exp(-ri));
12      end
13
14      function res=Q_p_1_0i(ri,ai,e)
15      res=-e/ai^2*(exp(-2*ri)-exp(-ri)+ri*exp(-ri));
16      end
17
18      function res=P_h_2_i(ri,ai,a1i,e)
19      res=a1i*e^2/2/ai^3*((ri^2-2*ri+2)-2*exp(-ri));
20      end
21
22      function res=P_p_2_i(ri,ai,a1i,e)
23      res=a1i*e^2/2/ai^3*(exp(-2*ri)*(ri^2+2*ri+2)-2*exp(-ri)
    );
24      end
25
26      function res=Q_h_2_0i(ri,ai,a1i,e)
27      res=-a1i*e^2/2/ai^4*((1-exp(-ri)-ri*exp(-ri))*(ri^2-2*
    ri)+ri^3*exp(-ri));
28      end
29
30      function res=Q_p_2_0i(ri,ai,a1i,e)
31      res=-a1i*e^2/2/ai^4*((exp(-2*ri)-exp(-ri)+ri*exp(-ri))
    *(ri^2+2*ri)-ri^3*exp(-ri));
32      end
33
34      function res=Q_h_2_1i(ri,ai,e)
35      res=-e^2/ai^3*(1-exp(-ri)-ri*exp(-ri)-ri^2/2*exp(-ri));
36      end
37
38      function res=Q_p_2_1i(ri,ai,e)
39      res=-e^2/ai^3*(exp(-2*ri)-exp(-ri)+ri*exp(-ri)-ri^2/2*
    exp(-ri));
40      end
```

```
41
42      function res=A1_1(ri,ai,e)
43      res=-exp(-ri)*(P_h_1_i(ri,ai,e));
44      end
45
46      function res=A2_1(ri,ai,e)
47      res=exp(-ri)*(P_h_1_i(ri,ai,e)-P_p_1_i(ri,ai,e));
48      end
49
50      function res=A3_1(ri,ai,e)
51      res=exp(-ri)*(P_p_1_i(ri,ai,e));
52      end
53
54      function res=Y0_1(ri,ai,e)
55      res=P_p_1_i(ri,ai,e)*Q_h_1_0i(ri,ai,e)-P_h_1_i(ri,ai,e)
    *Q_p_1_0i(ri,ai,e);
56      end
57
58      function res=A1_2(ri,ai,a1i,e)
59      res=-exp(-ri)*(P_h_1_i(ri,ai,e)+P_h_2_i(ri,ai,a1i,e));
60      end
61
62      function res=A2_2(ri,ai,a1i,e)
63      res=exp(-ri)*(P_h_1_i(ri,ai,e)+P_h_2_i(ri,ai,a1i,e))-
    exp(-ri)*(P_p_1_i(ri,ai,e)+P_p_2_i(ri,ai,a1i,e));
64      end
65
66      function res=A3_2(ri,ai,a1i,e)
67      res=exp(-ri)*(P_p_1_i(ri,ai,e)+P_p_2_i(ri,ai,a1i,e));
68      end
69
70      function res=Y0_2(ri,ai,a1i,e)
71      res=(P_p_1_i(ri,ai,e)+P_p_2_i(ri,ai,a1i,e))*(Q_h_1_0i(
    ri,ai,e)+Q_h_2_0i(ri,ai,a1i,e))-(P_h_1_i(ri,ai,e)+
```

```
P_h_2_i(ri,ai,a1i,e))*(Q_p_1_0i(ri,ai,e)+Q_p_2_0i(ri,ai,
a1i,e));
 end

 function res=Y1_2(ri,ai,a1i,e)
 res=(P_p_1_i(ri,ai,e)+P_p_2_i(ri,ai,a1i,e))*Q_h_2_1i(ri
,ai,e)-(P_h_1_i(ri,ai,e)+P_h_2_i(ri,ai,a1i,e))*Q_p_2_1i(
ri,ai,e);
 end

 function [x]=zhuiganfa(a,b,c,d)
 r=size(a);
 m=r(2);
 r=size(b);
 n=r(2);
 if size(a)~=size(c)|m~=n-1|size(b)~=size(d)
     error('');
 end
 l=zeros(1,n-1);
 u=zeros(1,n);
 y=zeros(1,n);
 x=zeros(1,n);
 u(1)=b(1);
 for i=2:n
     l(i-1)=a(i-1)/u(i-1);
     u(i)=b(i)-l(i-1)*c(i-1);
 end
 y(1)=d(1);
 for i=2:n
     y(i)=d(i)-l(i-1)*y(i-1);
 end
 x(n)=y(n)/u(n);
 for i=n-1:-1:1
     x(i)=y(i)/u(i);
```

```
        x(i)=(y(i)-c(i)*x(i+1))/u(i);
    end
    end

    function [res]=mynorm(A,B,h)
    T=(A-B).*(A-B);
    res=sqrt(sum(sum(T)))*h;
    end
```

C. 迭代参数估计代码

```
    n=32;
    vl=1;
    vr=1e+4;
    [p,el,x,y,u,i,et]=Solve(n,1e-6,vl,10);
    [p,er,x,y,u,i,et]=Solve(n,1e-6,vr,10);
    while(1)
        %vm=exp((log(vl)+log(vr))/2);
        vm=(vl+vr)/2;
        [p,em,x,y,u,i,et]=Solve(n,1e-6,vm,10);
        [p,em_,x,y,u,i,et]=Solve(n,1e-6,vm+1,10);
        if(em<em_)
            vr=vm;
            er=em;
        else
            vl=vm;
            el=em;
        end
        if(vr-vl<1)
            break;
        end
    end
```

（也可尝试使用其它的计算极值算法，如无导数法中的 Nelder-Mead 算法）