数值代数大作业报告

黄天域 2300010720

2025.02.07

1 引言

1.1 问题背景介绍

大作业主要关心的问题是数值求解 MAC 格式的离散 Stokes 方程，Stokes 方程的具体内容如下。

$$\begin{cases} -\Delta \vec{u} + \nabla p = \vec{F}, & (x,y) \in (0,1) \times (0,1), \\ \operatorname{div} \vec{u} = 0, & (x,y) \in (0,1) \times (0,1), \end{cases} \tag{1}$$

其中边界条件为

$$\begin{aligned} &\frac{\partial u}{\partial \vec{n}} = b, \quad y = 0, \quad \frac{\partial u}{\partial \vec{n}} = t, \quad y = 1, \\ &\frac{\partial v}{\partial \vec{n}} = l, \quad x = 0, \quad \frac{\partial v}{\partial \vec{n}} = r, \quad x = 1, \\ &u = 0, \quad x = 0, 1, \quad v = 0, \quad y = 0, 1, \end{aligned}$$

其中 $\vec{u} = (u, v)$ 为速度, p 为压力, $\vec{F} = (f, g)$ 为外力, $\vec{n}$ 为外法向方向, 这里边界定向取逆时针方向.

利用交错网格上的 MAC 格式离散 Stokes 方程 (1), 可得到如下线性方程组

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B^T} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{U} \\ \mathbf{P} \end{pmatrix} = \begin{pmatrix} \mathbf{F} \\ \mathbf{0} \end{pmatrix}. \tag{2}$$

这里交错网格的 MAC 格式展开如下，其思路是在内部格点利用有限差分离散原方程, 在 Neumann 边界处利用边界条件离散, 在强制边界为 0 处不列方程。方程 (2) 中 (使用 matlab 的语

法)

$$\mathbf{U} = [u; v], \mathbf{F} = [f; g]$$

将 $(0,1) \times (0,1)$ 的方格分拆为 $N \times N$ 的方格表, 从左下角起, 第 k 列 (从左往右) 记为列 $i = k$, 第 j 行 (从下往上) 记为 $j = k$ $(1 \leq k \leq N+1)$。由此可将 (x, y) 坐标系转换为 (i, j) 坐标系，那么 (x, y) 对应 (i, j) 坐标系中的 $(1 + \frac{x}{h}, 1 + \frac{y}{h})$, u, v, f, g 分别为 Stokes 方程 (1)中的 $\vec{u} = (u, v), \vec{F} = (f, g)$ 分量在网格的指定位置的值，下标取在 (i, j) 坐标系中代表位置。其具体可见下面的示意图。

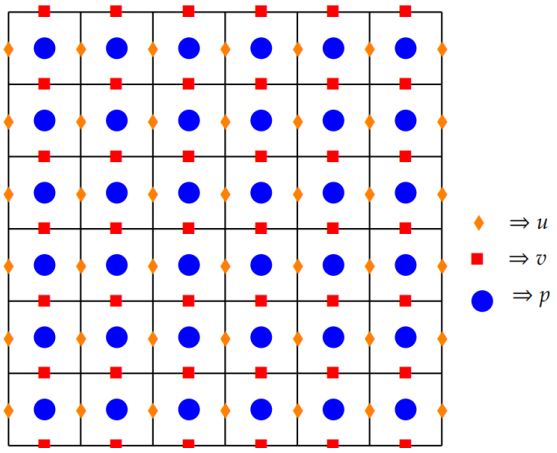

图 1: MAC 交错网格

关于 u 的方程 $(h = 1/N)$:

(1) 当 $2 \leq i \leq N, 2 \leq j \leq N-1$,

$$-\frac{u_{i+1,j+\frac{1}{2}} - 2u_{i,j+\frac{1}{2}} + u_{i-1,j+\frac{1}{2}}}{h^2} - \frac{u_{i,j+\frac{3}{2}} - 2u_{i,j+\frac{1}{2}} + u_{i,j-\frac{1}{2}}}{h^2} + \frac{p_{i+\frac{1}{2},j+\frac{1}{2}} - p_{i-\frac{1}{2},j+\frac{1}{2}}}{h} = f_{i,j+\frac{1}{2}};$$

(2) 当 $2 \leq i \leq N (j = 1)$,

$$-\frac{u_{i,\frac{5}{2}} - u_{i,\frac{3}{2}}}{h^2} - \frac{b_{i,1}}{h} - \frac{u_{i+1,\frac{3}{2}} - 2u_{i,\frac{3}{2}} + u_{i-1,\frac{3}{2}}}{h^2} + \frac{p_{i+\frac{1}{2},\frac{3}{2}} - p_{i-\frac{1}{2},\frac{3}{2}}}{h} = f_{i,\frac{3}{2}};$$

(3) 当 $2 \leq i \leq N (j = N)$,

$$\frac{u_{i,N+\frac{1}{2}} - u_{i,N-\frac{1}{2}}}{h^2} - \frac{t_{i,N+1}}{h} - \frac{u_{i+1,N+\frac{1}{2}} - 2u_{i,N+\frac{1}{2}} + u_{i-1,N+\frac{1}{2}}}{h^2} + \frac{p_{i+\frac{1}{2},N+\frac{1}{2}} - p_{i-\frac{1}{2},N+\frac{1}{2}}}{h} = f_{i,N+\frac{1}{2}};$$

(4) 当 $i = 1, N+1, 1 \leq j \leq N, u_{i,j+\frac{1}{2}} = 0.$

关于 v 的方程 $(h = 1/N)$：

(1) 当 $2 \le j \le N, 2 \le i \le N-1$,

$$-\frac{v_{i+\frac{3}{2},j} - 2v_{i+\frac{1}{2},j} + v_{i-\frac{1}{2},j}}{h^2} - \frac{v_{i+\frac{1}{2},j+1} - 2v_{i+\frac{1}{2},j} + v_{i+\frac{1}{2},j-1}}{h^2} + \frac{p_{i+\frac{1}{2},j+\frac{1}{2}} - p_{i+\frac{1}{2},j-\frac{1}{2}}}{h} = g_{i+\frac{1}{2},j};$$

(2) 当 $2 \le j \le N(i = 1)$,

$$-\frac{v_{\frac{5}{2},j} - v_{\frac{3}{2},j}}{h^2} - \frac{l_{1,j}}{h} - \frac{v_{\frac{3}{2},j+1} - 2v_{\frac{3}{2},j} + v_{\frac{3}{2},j-1}}{h^2} + \frac{p_{\frac{3}{2},j+\frac{1}{2}} - p_{\frac{3}{2},j-\frac{1}{2}}}{h} = g_{\frac{3}{2},j};$$

(3) 当 $2 \le j \le N(i = N)$,

$$\frac{v_{N+\frac{1}{2},j} - v_{N-\frac{1}{2},j}}{h^2} - \frac{r_{N+1,j}}{h} - \frac{v_{N+\frac{1}{2},j+1} - 2v_{N+\frac{1}{2},j} + v_{N+\frac{1}{2},j-1}}{h^2} + \frac{p_{N+\frac{1}{2},j+\frac{1}{2}} - p_{N+\frac{1}{2},j-\frac{1}{2}}}{h} = g_{N+\frac{1}{2},j};$$

(4) 当 $j = 1$ 和 $j = N+1, 1 \le i \le N, v_{i+\frac{1}{2},j} = 0.$

有了这些准备，我们可以来解释离散 Stokes 方程中的变量 $\mathbf{U}, \mathbf{P}$, 这里 $\mathbf{U}$ 是由 $u_{i,j+\frac{1}{2}}(2 \le i \le N, 1 \le j \le N)$ 和 $v_{i+\frac{1}{2},j}(1 \le i \le N, 2 \le j \le N)$ 构成的, 而 $\mathbf{P}$ 由 f, g 构成。这里我们排除了 u, v 在边界处被强制设置为 0 的部分，这将为我们的理论推导部分带来一定帮助。

1.2 记号约定与测试设备说明

在开始具体讨论我的算法前，我需要约定一些记号, 它们与我在代码中使用的记号是一致的, 我使用 matlab 语言编程, 下面讨论中一些矩阵的切片操作等表达含义与相应的 matlab 代码一致。

我使用 $(N+1) \times N$ 的矩阵 U, F_U 分别表示速度分量 u 和外力分量 f。

$N \times (N+1)$ 的矩阵 V, F_V 表示速度分量 v 和外力分量 g。

$N \times N$ 的矩阵 P(是否为粗体表示的都是离散压力矩阵) 表示压力 p。

$$U(i,j) = u_{i+\frac{1}{2},j}, F_U(i,j) = f_{i+\frac{1}{2},j}(1 \le i \le N+1, 1 \le j \le N)$$
$$V(i,j) = u_{i,j+\frac{1}{2}}, F_V(i,j) = f_{i,j+\frac{1}{2}}(1 \le i \le N, 1 \le j \le N+1)$$
$$P(i,j) = p_{i+\frac{1}{2},j+\frac{1}{2}}(1 \le i \le N, 1 \le j \le N)$$

注意到这里的矩阵 U, V 和矩阵 $\mathbf{A}$ 的尺寸并不符合，这是因为我保留了 u, v 中在边界强制为 0 的部分, 但这主要是为了编写代码中能够减少分类, 简化代码。本质上并不会影响矩阵 $\mathbf{A}, \mathbf{B}, \mathbf{B^T}$ 等矩阵的作用。(换言之，我们将是否保留强制为 0 的边界的两个矩阵等同起来)

本大作业使用的测试解如下：在区域 $\Omega = (0,1) \times (0,1)$ 上，外力为

$$f(x,y) = -4\pi^2(2\cos(2\pi x) - 1)\sin(2\pi y) + x^2$$

$$g(x,y) = 4\pi^2(2\cos(2\pi y) - 1)\sin(2\pi x).$$

此时 Stokes 方程的真解为

$$u(x,y) = (1 - \cos(2\pi x))\sin(2\pi y),$$

$$v(x,y) = -(1 - \cos(2\pi y))\sin(2\pi x),$$

$$p(x,y) = \frac{x^3}{3} - \frac{1}{12}.$$

此外，以下代码运行所使用的 CPU 为 I5-1335U, 基本频率 (最大频率) 为 1.30 GHz(4.60 GHz), GPU 为 RTX 2050, 用于并行加速计算。

1.3 算法思路说明

在实现要求的算法同时，我注重提升算法的效率，并尝试利用 GPU 加速运算。为此我需要避免使用 for 循环，利用矢量化操作或者 matlab 提供的兼容 GPU 的函数代替 for 循环操作, 例如矩阵卷积。同时我还修改了第一题中 DGS 迭代变量的遍历次序，使用使用红黑 GS 迭代，并在后续更新速度、压力分量时也采取红黑遍历顺序。除此以外，我的算法与课堂以及 ppt 上所述一致。

此外，我还针对使用是否使用 GPU 分别进行了针对优化, 在使用 GPU 加速的前提下，我第一问的算法在 $N = 2048$ 的情况下只需要几秒钟便可以收敛到指定精度。

2 准备工作

在详细介绍我对三个问题的算法前，我先进行一些准备工作的说明。它们构成了我的三个算法的基石。

2.1 提升限制算子

提升限制算子我参考图片2中给出的提升限制算子完成。

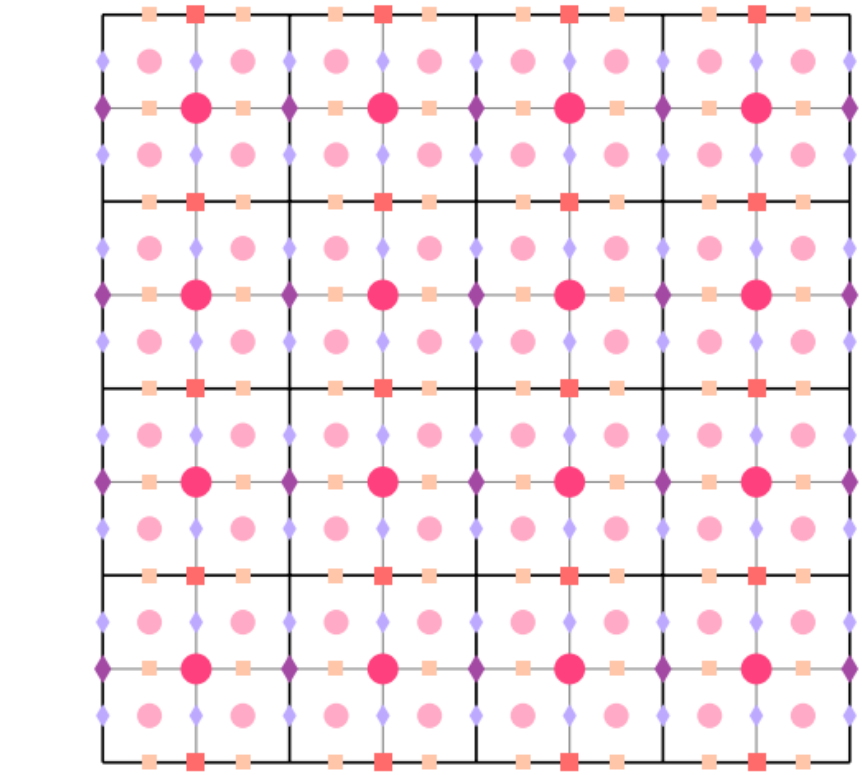

图 2: 提升限制算子的实现

2.1.1 提升算子

注意到切片操作在 CPU 和 GPU 上都能够比较高效的运行，故我使用切片操作实现提升算子如下 (仅展示核心计算代码)。

Listing 1: 提升算子

```
function [U_lift, V_lift, P_lift] = lift(U, V, P, N, device)
    P_lift(1:2:end, 1:2:end) = P;
    P_lift(1:2:end, 2:2:end) = P;
    P_lift(2:2:end, 1:2:end) = P;
    P_lift(2:2:end, 2:2:end) = P;
    U_lift(1:2:end, 1:2:(N*2)) = U;
    U_lift(1:2:end, 2:2:(N*2)) = U;
    U_lift(2:2:N*2, 1:2:N*2) = (U(1:N, 1:N) + U(2:N+1, 1:N)) / 2;
    U_lift(2:2:N*2, 2:2:N*2) = U_lift(2:2:N*2, 1:2:N*2);
    V_lift(1:2:N*2, 1:2:end) = V;
    V_lift(2:2:N*2, 1:2:end) = V;
    V_lift(2:2:N*2, 2:2:N*2) = (V(:, 1:N) + V(:, 2:N+1)) / 2;
    V_lift(1:2:N*2, 2:2:N*2) = V_lift(2:2:N*2, 2:2:N*2);
end
```

2.1.2 限制算子

完全类似地，我们可以只使用矩阵的切片等高效操作实现限制算子，其思路与提升算子的实现完全一致，只需注意边界处需要额外处理即可。具体地讲，u, v 在边界处的限制 u_{res}, v_{res} 为最近的两个 u, v 节点的平均值。经过实验这个限制算子是高效的，为了使报告简洁，我没有贴这部分的代码，您可以在提交的代码文件”restrict.m” 中查看其具体实现。

2.2 矩阵 $\mathbf{A}, \mathbf{B}, \mathbf{B}^{\mathbf{T}}$ 的作用

通过观察离散 Stokes 方程的展开格式，我们便可得到矩阵 $\mathbf{A}, \mathbf{B}, \mathbf{B}^{\mathbf{T}}$ 作用的具体公式。具体地讲 $\mathbf{A}$ 在 U, V 上的作用会分别得到 $(N+1) \times N$ 和 $N \times (N+1)$ 维的向量 A_U, A_V(本质是只作用其中除去强制为 0 边界的部分，但是作用可以开拓到整个矩阵 U, V 上)。其具体公式即为离散 Stokes 方程展开式左边由 u, v 分量构成的部分。类似地我们可以得到 $\mathbf{B}, \mathbf{B}^{\mathbf{T}}$ 的作用。

2.2.1 矩阵 A 的作用

其核心部分可以利用卷积操作快速得到。

$$kernel = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$A_U(2:N, 2:N-1) = \frac{1}{h^2}\text{conv2}(U, kernel, \text{'valid'});$$

$$A_V(2:N-1, 2:N) = \frac{1}{h^2}\text{conv2}(V, kernel, \text{'valid'});$$

这里 conv2 为 matlab 的内置卷积函数。对于 $C = \text{conv2}(A, B)$ 有

$$C(j,k) = \sum_p \sum_q A(p,q)B(j-p+1, k-q+1)$$

其中 (默认模式下)，p 和 q 取遍所有使得 $A(p,q)$ 和 $B(j-p+1, k-q+1)$ 的下标合法的值。若设置模式为 'valid' (默认模式为'full') 则 j,k 需要满足 p,q 可以分别取遍 $1,\ldots,j$ 和 $1,\ldots,k$。因而 C 的尺寸需要相应缩小。

类似地, A_U, A_V 的边界 $A_U[2:N,[1,N]], A_V[[1,N],2:N]$ 也可以由 U,V 相应部分与某些 kernel 卷积得到。例如

$$kernel = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -3 & 1 \end{pmatrix}$$

$$A_V(1, 2:N) = \frac{1}{h^2}\text{conv2}(V(1:2, 1:N+1), kernel, \text{'valid'});$$

其余的情况是类似可得。

此后我们约定记号 $*_f$ 为模式为 full 模式的矩阵卷积,$*_v$ 为模式为 valid 的矩阵卷积。

2.2.2 矩阵 B 的作用

B 作用在压力矩阵 P 上得到 B_{Pu}, B_{Pv} 分别为 P 在 U,V 分量上的投影。则我们有

$$B_{Pu}(2:N, :) = \frac{1}{h}P *_v \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$B_{Pv}(:, 2:N) = \frac{1}{h}P *_v \begin{pmatrix} 1 & -1 \end{pmatrix}$$

2.2.3 矩阵 B^T 的作用

$\mathbf{B^T}$ 作用在速度 (U,V) 上得到 $N\times N$ 的矩阵 P 有

$$P=-\frac{1}{h}\left(U*_v\begin{pmatrix}1\\-1\end{pmatrix}+V*_v\begin{pmatrix}1 & -1\end{pmatrix}\right)$$

3 第一题

3.1 问题描述

分别取 $N=64,128,256,512,1024,2048$，以 DGS 为磨光子，用基于 V-cycle 的多重网格方法求解离散问题 2，停机标准为 $\frac{\|r_h\|_2}{\|r_0\|_2}\le 10^{-8}$，对不同的 ν_1,ν_2,L，比较 V-cycle 的次数和 CPU 时间，并计算误差

$$e_N=h\left(\sum_{j=1}^{N}\sum_{i=2}^{N}\left|u_{i,j+\frac{1}{2}}-u((i-1)h,(j-\frac{1}{2})h)\right|^2+\sum_{j=2}^{N}\sum_{i=1}^{N}\left|v_{i+\frac{1}{2},j}-v((i-\frac{1}{2})h,(j-1)h)\right|^2\right)^{\frac{1}{2}}.$$

3.2 红黑 GS 迭代以及红黑 DGS 迭代

为了并行加速，我没有使用顺序的 GS 迭代或 DGS 迭代，而使用了红黑 GS 迭代和红黑 DGS 迭代。下面我将介绍它们的细节。

注意到 GS 迭代 (或 DGS 迭代) 即按照某种顺序依次更新解，但是新更新的解分量需要利用上这一轮已经更新过的解分量。因此更新顺序对与能否并行运算有很大影响。基于 Stokes 方程的特点，红黑遍历的方式恰好能够满足我们的需求。

具体地讲红黑 GS 迭代在更新 U, V 时采取如下的更新顺序, 更新分为两轮, 第一轮更新

$$U(i,j),\ V(i,j)(i+j\equiv 1\ mod\ 2)$$

当然对于 U, V 来讲 (i,j) 的定义域是不同的，它们在这里分别取遍使得它们有定义且不被强制为 0 的区域。再更新

$$U(i,j),\ V(i,j)(i+j\equiv 0\ mod\ 2)$$

注意到 **A** 的特殊性，离散 Stokes 方程中每个 $U(i,j),V(i,j)$ 都分别只会和与之相邻的 U, V 分量同时出现在一个式子中。这意味着在两轮更新中，需要被更新的分量都是互相不会影响的，因此在每一轮中，可以并行地同时处理所有需要更新的分量。

对于红黑 DGS 迭代，处理是类似地。将 $N \times N$ 个格子以左下角顶点编号为 (i, j) $1 \le i, j \le N$，则第一轮更新 $i + j \equiv 1 \ mod \ 2$ 的格子，第二轮更新 $i + j \equiv 1 \ mod \ 2$ 的格子。与前面相同的，其中的每一轮更新都是可以并行处理的。

3.2.1 算法实现

接下来我先给出伪代码以展示算法的整体框架，再讨论如何进行并行计算。

Algorithm 1 以 DGS 迭代法为磨光子的 V-cycle 多重网格算法

while $\|r_h\|_2/\|r_0\|_2 > 10^{-8}$ **do**
    repeat
        使用红黑 DGS 迭代 ν_1 次，计算残量并限制到下一层.
    until 到达底层网格，网格尺寸为 $bottom \times bottom$.
    repeat
        使用红黑 DGS 迭代 ν_2 次，计算残量并提升到上一层.
    until 到达顶层
end while

红黑 GS 迭代的并行实现利用了 matlab 的广播机制，通过创建一个与 U, V 相同尺寸的 true false 掩码矩阵，筛选出每一轮需要更新的解分量，再依次加上他们的残差。这里我对使用设备 device 为 GPU 还是 CPU 分别进行处理，在使用设备为 CPU 时，因为并行计算算法在设计时只能利用 matlab 的广播机制或者兼容 GPU 的函数，灵活性受限，因此设计出的并行算法相比 for 循环的时间复杂度会更高一些，经过实验我最终决定采取 for 循环的方式完成红黑 GS 迭代。下面我主要展示 GPU 版本的核心实现代码。

Listing 2: 红黑 GS 迭代

```
function [U_ite, V_ite, P_ite] = RBGS_iteration(U_ite, V_ite, P_ite, F_U, F_V,
    D, N, device) %该函数负责进行一次红黑DGS迭代，这里取出其中红黑GS的部分
    N = gpuArray(N);
    h = 1/N;
    h_sq = h^2;
    [red_mask_U, black_mask_U] = create_red_mask_U(N);%创建U矩阵的红黑掩码
    [red_mask_V, black_mask_V] = create_red_mask_V(N);%创建V矩阵的红黑掩码
```

```
    [A_U, A_V] = apply_A(U_ite, V_ite, N, device); %计算A作用在U, V上的结果
    [B_Pu, B_Pv] = apply_B(P_ite, N, device); %计算B作用在P上的结果
    residual_U = F_U - (A_U + B_Pu); %F_U为外力的U分量, 这里是U分量对应的残差
    residual_V = F_V - (A_V + B_Pv); %F_V为外力的V分量, 这里是V分量对应的残差
    h_temp = h_sq / 4;
    % 更新内部红色节点(i + j mod 2 == 1)
    U_ite(red_mask_U) = U_ite(red_mask_U) + h_temp * residual_U(red_mask_U);
    V_ite(red_mask_V) = V_ite(red_mask_V) + h_temp * residual_V(red_mask_V);
    h_temp = h_sq / 3;
    %边界处理
    U_ite(2:2:end, 1) = U_ite(2:2:end, 1) + h_temp * residual_U(2:2:end, 1);
    U_ite(end-2:-2:3, N) = U_ite(end-2:-2:3, N) + h_temp * residual_U(end
        -2:-2:3, N);
    V_ite(1, 2:2:end) = V_ite(1, 2:2:end) + h_temp * residual_V(1, 2:2:end);
    V_ite(N, end-2:-2:3) = V_ite(N, end-2:-2:3) + h_temp * residual_V(N, end
        -2:-2:3);
    %黑色节点更新逻辑与红色节点完全一致, 这里为从简便省略了。
end
```

红黑 DGS 后续的压力更新以及速度更新与红黑 GS 的思路一致。注意到 DGS 更新压力与速度的方式如下：(u, v) 是已经使用红黑 GS 迭代更新得到的速度分量，再设散度方程为 (实际过程中由于提升限制散度方程不一定与原始离散 Stokes 方程一致)

$$\mathbf{B}^{\mathrm{T}}\mathbf{U} = D =: d_{i,j} (1 \le i, j \le N)$$

并记 $U = (u_{i,j})$, $V = (v_{i,j})$, 则 DGS 更新的公式为：

1. 对内部单元 (i, j) (四个顶点 $((i-1)h, (j-1)h), (ih, (j-1)h), ((i-1)h, jh), ((i-1)h, (j-1)h)$), $2 \le i, j \le N-1$，计算散度方程的残量

$$r_{i,j} = -\frac{u_{i+1,j} - u_{i,j}}{h} - \frac{v_{i,j+1} - v_{i,j}}{h} - d_{i,j},$$

并令 $\delta = r_{i,j}h/4$。

更新内部单元速度：

$$u_{i,j} = u_{i,j} - \delta, \quad u_{i+1,j} = u_{i+1,j} + \delta,$$

$$v_{i,j} = v_{i,j} - \delta, \quad v_{i,j+1} = v_{i,j+1} + \delta$$

更新内部单元的压力：

$$
\begin{aligned}
p_{i,j} &= p_{i,j} + r_{i,j},\\
p_{i+1,j} &= p_{i+1,j} - r_{i,j}/4,\\
p_{i,j+1} &= p_{i,j+1} - r_{i,j}/4,\\
p_{i-1,j} &= p_{i-1,j} - r_{i,j}/4,\\
p_{i,j-1} &= p_{i,j-1} - r_{i,j}/4
\end{aligned}
$$

2. 对边界单元 (i,N) (四个顶点 $((i-1)h,(N-1)h),((i-1)h,Nh),(ih,(N-1)h),(ih,Nh)$)，$2 \le i \le N-1$，先计算散度的残量 $r_{i,N}$，计算公式如前，并令 $\delta = r_{i,N}h/3$。

   更新速度如下：

$$
\begin{aligned}
u_{i,N} = u_{i,N} - \delta,\ u_{i+1,N} &= u_{i+1,N} + \delta,\\
v_{i,N} &= v_{i,N} - \delta.
\end{aligned}
$$

   更新边界单元 (i,N) 的压力：

$$
\begin{aligned}
p_{i,N} &= p_{i,N}^{k} + r_{i,N},\\
p_{i+1,N} &= p_{i+1,N} - r_{i,N}/3,\\
p_{i-1,N} &= p_{i-1,N} - r_{i,N}/3,\\
p_{i,N-1} &= p_{i,N-1} - r_{i,N}/3
\end{aligned}
$$

   对于其余三个边界更新是类似地。

3. 对顶点单元 $(1,1)$，计算散度残量 $r_{1,1}$，计算公式如前，令 $\delta = r_{1,1}h/2$。更新速度如下：

$$
u_{2,1} = u_{2,1} + \delta, \quad v_{1,2} = v_{1,2} + \delta.
$$

   更新压力如下：

$$
\begin{aligned}
p_{1,1} &= p_{1,1} + r_{1,1},\\
p_{1,2} &= p_{1,2} - \frac{r_{1,1}}{2},\\
p_{2,1} &= p_{2,1} - \frac{r_{1,1}}{2}.
\end{aligned}
$$

   对其他顶点单元 $(1,N),(N,1)$ 和 (N,N)，类似更新速度和压力。

类似于矩阵作用的部分，这里仍然可以通过卷积加速最主要的内部格子更新, 对于第一轮红色格点的更新：

$$
\begin{aligned}
mask &= \begin{cases} 1 & i+j \equiv 1 \ mod\ 2, \text{且 } (i-1)(i-N)(j-1)(j-N) \neq 0 \\ 0 & else \end{cases} \\
R &= (\mathbf{B}^{\mathbf{T}}\mathbf{U} - D) \odot mask \\
R &= R(2:N-1,\ 2:N-1) \\
P &= P + R *_f \begin{pmatrix} 0 & -\frac{1}{4} & 0 \\ -\frac{1}{4} & 1 & -\frac{1}{4} \\ 0 & -\frac{1}{4} & 0 \end{pmatrix} \\
U &= U + R *_f \begin{pmatrix} -1 \\ 1 \end{pmatrix} \\
V &= V + R *_f \begin{pmatrix} -1 & 1 \end{pmatrix}
\end{aligned}
$$

这里 $\odot$ 指 Hadmard 乘积。对于边界以及角部格子的更新则完全类似提升算子的实现，可以直接通过矩阵切片的方式进行更新。至此我们完成了第一问算法的 GPU 版本的实现，CPU 版本则是完全将其中的更新部分转换为 for 循环并适当减少不必要的计算得到的。

3.2.2 GPU 并行加速的补充说明

由于在规模较小的计算问题中，GPU 版本的代码因为灵活性受限导致时间复杂度更高、还有 CPU 以及 GPU 间数据传输等问题会导致 GPU 加速效果并不明显，有使用 GPU 计算时间甚至超过纯 CPU 计算的情况出现。对此，在设置 device = 'gpu' 时，我提供了参数 tol 以决定从哪 Vcycle 一层开始进行 GPU 加速。经过实验 $N \leq 1024$ 时，只加速最顶层是比较合适的，当 $N = 2048, 4096$ 时，加速最上面两层是比较合适的。此外由于 CUDA 启动等原因，使用 GPU 计算一系列任务时，最开始的几个任务可能会稍慢。

3.3 数值结果

由于我的算法优化较好，即使是 $N = 4096$ 也能较快地收敛，因此下面给出的数值结果均会包含 $N = 4096$ 的测试数据，同时每一组测试数据我会分别给出只使用 CPU 和使用 GPU 加

速的运行时间。

3.3.1 和真实解的误差

表 1: 误差

N	64	128	256	512	1024	2048	4096
误差	1.4951e-03	3.7363e-04	9.3399e-05	2.3349e-05	5.8372e-06	1.4593e-06	3.6483e-07

以上误差为参数 $\nu_1 = 1, \nu_2 = 1, L = \frac{N}{4}$ 下的结果，由于不同超参的选择对于误差的影响不大，便一这一组数据为参考进行分析。

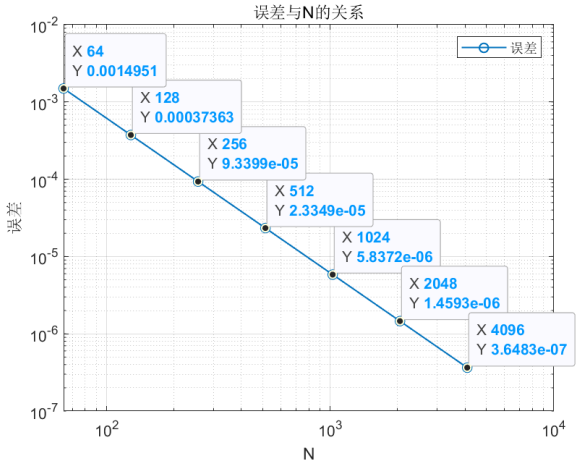

图 3: Enter Caption

在绘制的对数-对数图中，可以轻松看出其规律：误差与 $\frac{1}{N^2}$ 成正比, N 每乘 2, 误差变为原来的 $\frac{1}{4}$.(事实上是应当与某个 N^a 成正比，经过计算发现 $a = -2$)

3.3.2 不同超参 ν_1, ν_2, L 对于 Vcycle 迭代次数以及计算时间的影响

表 2: $v_1 = 1, v_2 = 1, N/L = 4$, 底层网格为 4×4.

N	64	128	256	512	1024	2048	4096
求解时间-GPU 加速 (s)	0.585156	0.666738	0.785708	0.890858	1.617643	4.870252	35.596113
求解时间-CPU(s)	0.007870	0.024075	0.097521	0.604895	3.884930	25.512464	161.447916
V-cycle 次数	10	10	11	11	11	11	12

表 3: $v_1 = 1, v_2 = 1, N/L = 2$, 底层网格为 2×2.

N	64	128	256	512	1024	2048	4096
求解时间-GPU 加速 (s)	0.594296	0.627040	0.796226	0.928459	1.633581	4.894184	38.740656
求解时间-CPU(s)	0.007531	0.023540	0.094621	0.605630	3.904031	25.693468	164.185407
V-cycle 次数	10	10	11	11	11	11	12

误差与 **3.3.1** 中的一致。此外当 $N/L = 8$ 时，所需迭代次数 (除了 $N = 4096$ 时迭代次数增加到 12) 与 $N/L = 4$ 的情形一致，求解时间相近，但是当 $N/L = 16$ 时，所需迭代次数将大大增加，详见下面 $v_1 = 2, v_2 = 2$ 的情形 (表 5)

表 4: $v_1 = 2, v_2 = 2, N/L = 4$, 底层网格为 4×4.

N	64	128	256	512	1024	2048	4096
求解时间-GPU 加速 (s)	0.886221	0.888649	0.900877	1.090027	1.778190	5.456692	36.195024
求解时间-CPU(s)	0.007691	0.022006	0.091151	0.586066	4.316558	30.263790	172.858025
V-cycle 次数	7	7	7	7	7	7	7
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3349e-5	5.8373e-6	1.4593e-6	3.6486e-7

表 5: $\nu_1 = 2, \nu_2 = 2, N/L = 16$, 底层网格为 16×16.

N	64	128	256	512	1024	2048	4096
求解时间-GPU 加速 (s)	2.590333	2.599041	2.604565	3.106615	5.051227	14.704554	116.321746
求解时间-CPU(s)	0.031651	0.064984	0.274295	1.698034	10.975851	82.930114	490.921569
V-cycle 次数	21	21	20	20	19	19	19
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3349e-5	5.8373e-6	1.4593e-6	3.6486e-7

表 6: $\nu_1 = 8, \nu_2 = 8, N/L = 4$, 底层网格为 4×4.

N	64	128	256	512	1024	2048	4096
求解时间-GPU 加速 (s)	1.817362	1.925761	1.971945	2.132085	3.629367	11.227309	88.413729
求解时间-CPU(s)	0.017133	0.038751	0.164940	1.071890	8.692657	67.754301	369.508251
V-cycle 次数	4	4	4	4	4	4	4
误差	1.4951e-3	3.7363e-4	9.3398e-5	2.3349e-5	5.8374e-6	1.4610e-6	3.7229e-7

表 7: $\nu_1 = 4, \nu_2 = 2, N/L = 4$, 底层网格为 4×4.

N	64	128	256	512	1024	2048	4096
求解时间-GPU 加速 (s)	0.981314	1.078697	1.164756	1.356919	2.251776	6.734809	58.382104
求解时间-CPU(s)	0.016822	0.034868	0.110544	0.712314	5.211267	39.760339	218.432401
V-cycle 次数	6	6	6	6	6	6	6
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3349e-5	5.8374e-6	1.4610e-6	3.6512e-7

根据上面的数据可以知道 GPU 加速在问题规模较大的时候能够相当程度上加速计算过程，但在规模较小时可能会适得其反，还有一些容易发现的规律，例如 ν_1, ν_2 与迭代次数成负相关关系等。最后，$\nu_1 = 1, \nu_2 = 1, L = \frac{N}{4}$ 是一组比较不错的超参数，其在各个 N 上均有相当不错的表现。

4 第二题

4.1 问题描述

分别取 $N = 64, 128, 256, 512$，以 Uzawa Iteration Method 求解上述离散问题，停机标准为 $\frac{\|r_h\|_2}{\|r_0\|_2} \leq 10^{-8}$，并计算误差

$$e_N = h\left(\sum_{j=1}^{N}\sum_{i=2}^{N}\left|u_{i,j+\frac{1}{2}} - u((i-1)h,(j-\frac{1}{2})h)\right|^2 + \sum_{j=2}^{N}\sum_{i=1}^{N}\left|v_{i+\frac{1}{2},j} - v((i-\frac{1}{2})h,(j-1)h)\right|^2\right)^{\frac{1}{2}}.$$

4.2 Uzawa Iteration Method

我使用共轭梯度法在 Uzawa 迭代中精确求解方程 $\mathbf{A}U = \mathbf{F} - \mathbf{B}P$，其为代码如下

Algorithm 2 Uzawa 迭代法

Require: $P_0, k = 0$

1: **while** $\|r_h\|_2/\|r_0\|_2 > 10^{-8}$ **do**

2: 利用共轭梯度法求解 $\mathbf{A}\mathbf{U}_{k+1} = \mathbf{F} - \mathbf{B}P_k$

3: 更新压力 $P_{k+1} = P_k + \alpha(\mathbf{B}^T\mathbf{U}_{k+1})$

4: **end while**

注意到共轭梯度法中只需要用到 $\mathbf{A}$ 在速度分量 U, V 上的作用，并不需要真正计算出 $\mathbf{A}$, 并且向量之间的点积事实上可以使用 hadamard 乘积与 matlab 的 sum 函数结合完成。从而 Uzawa 迭代法仍然能够并行地实现。这里我参照书上给出的共轭梯度法适当修改，在当前残差 2 范数小于迭代初始残差 2 范数的 $\frac{1}{10^{10}}$ 或迭代次数超过 $3 \times N$ 时跳出迭代。

最后，我们需要确定参数 α 如何选取。

4.3 最优参数 α_* 的选取

注意到 $P_{k+1} = P_k + \alpha(\mathbf{B^T}\mathbf{U}_{k+1}) = P_k + \alpha\mathbf{B^T}\mathbf{A^{-1}}(\mathbf{F} - \mathbf{B}P_k) = (I - \alpha\mathbf{B^T}\mathbf{A^{-1}}\mathbf{B})P_k + \alpha\mathbf{B^T}\mathbf{A^{-1}}\mathbf{F}$。所以 Uzawa 迭代中更新 P 的部分可以视为松弛迭代。这要求我们研究矩阵 $C := \mathbf{B^T}\mathbf{A^{-1}}\mathbf{B}$

引理 4.1. *$rank(\mathbf{B}) = N^2 - 1$, 且 $\mathbf{B}$ 的零空间由所有元素均相等的 $N \times N$ 矩阵构成, 因而维数是 1。*

引理的证明: 注意到 $\mathbf{B}$ 作用在 $N \times N$ 矩阵 P 上给出 P 在列、行方向上的差分 (B_{Pu}, B_{Pv}), 根据 **2.2.2** 小节的讨论 $(B_{Pu}, B_{Pv}) = \mathbf{0}$ 当且仅当 P 的所有元素均相等，证毕。

引理 4.2. $\mathbf{AB} = \mathbf{BB^TB}$

注意到 $\mathbf{A}$ 是 $-\Delta$ 的离散化，$\mathbf{B}$ 是 ∇ 的离散化, $\mathbf{B^T}$ 是 $-\mathrm{div}$ 的离散化。对于 $(0,1)\times(0,1)$ 上的光滑函数 p,

$$-\Delta(\nabla p) = -(\frac{\partial^3 p}{\partial x^2 \partial x} + \frac{\partial^3 p}{\partial y^2 \partial x}, \frac{\partial^3 p}{\partial x^2 \partial y} + \frac{\partial^3 p}{\partial y^2 \partial y}) = \nabla((-\mathrm{div}) \cdot \nabla(p)))$$

通过这个转换思路，我们能够直接验证引理 4.2

引理 4.2 的证明: 对 $N \times N$ 矩阵 P, 记 $P_1 = \mathbf{AB}P, P_2 = \mathbf{BB^TB}P$, 由对称性，只需证明 P_1, P_2 在 U 分量的投影相等 (那么在 V 分量上的投影也同理相等), 根据算子 $\mathbf{A}, \mathbf{B}, \mathbf{B^T}$ 与其连续形式 $-\Delta, \nabla, -\mathrm{div}$ 的对应, 我们可以自然的记 (∂x 指的是沿列方向的差分例如 $P_{i+1,j} - P_{i,j}$, ∂y 则反之)

$$\begin{aligned}
\mathbf{A} &=: -(\frac{\partial_A^2}{\partial x^2} + \frac{\partial_A^2}{\partial y^2}) \\
\mathbf{B} &=: (\frac{\partial_B}{\partial x}, \frac{\partial_B}{\partial y}) \\
\mathbf{B^T} &=: -(\frac{\partial_{B^T}}{\partial x}, \frac{\partial_{B^T}}{\partial x}) \cdot \quad \text{(使用向量 } -(\frac{\partial_{B^T}}{\partial x}, \frac{\partial_{B^T}}{\partial x}) \text{ 与矩阵做 “点积”)}
\end{aligned}$$

那么 P_1 在 U 方向的投影即为 $-\frac{\partial_A^2}{\partial x^2}\frac{\partial_B}{\partial x}P - \frac{\partial_A^2}{\partial y^2}\frac{\partial_B}{\partial x}P$, P_2 在 U 方向投影即为 $-\frac{\partial_B}{\partial x}\frac{\partial_{B^T}}{\partial x}\frac{\partial_B}{\partial x}P - \frac{\partial_B}{\partial x}\frac{\partial_{B^T}}{\partial y}\frac{\partial_B}{\partial y}P$

从而只用分别证明

$$\frac{\partial_A^2}{\partial x^2}\frac{\partial_B}{\partial x} = \frac{\partial_B}{\partial x}\frac{\partial_{B^T}}{\partial x}\frac{\partial_B}{\partial x}, \frac{\partial_A^2}{\partial y^2}\frac{\partial_B}{\partial x} = \frac{\partial_B}{\partial x}\frac{\partial_{B^T}}{\partial y}\frac{\partial_B}{\partial y}$$

回顾我们对于速度分量 U, V 的约定，它们与 $\mathbf{A}$ 的尺寸不符合但是由于强制边界为 0, 我们仍然可以考虑 $\mathbf{A}, \mathbf{B^T}$ 等在其上的作用。

那么对于前者: 设 $\tilde{U} = \frac{\partial_B}{\partial x}P$ 是一个 $(N+1)\times N$ 的矩阵 (最上最下两行为 0), 则根据 **2.2** 小节, 我们有

$$\left(\frac{\partial_A^2}{\partial x^2}\tilde{U}\right)(2:N,:) = \frac{1}{h^2}\tilde{U} *_v \begin{pmatrix} -1 \\ 2 \\ -1 \end{pmatrix} = \frac{1}{h^2}(\tilde{U} *_v \begin{pmatrix} 1 \\ -1 \end{pmatrix}) *_v \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{\partial_B}{\partial x}\frac{\partial_{B^T}}{\partial x}\tilde{U}$$

得证

对于后者:

$$\hat{V} := \frac{\partial_B}{\partial y} P \quad \hat{V}(i,j) = \frac{1}{h}(P(i,j) - P(i,j-1)) \quad (i = 1,\dots,N, j = 2,\dots,N)$$
$$\hat{P} := \frac{\partial_{B^T}}{\partial y} \hat{V} \quad \hat{P}(i,j) = \frac{1}{h}(\hat{V}(i,j+1) - \hat{V}(i,j)) \quad (i = 1,\dots,N, j = 1,\dots,N)$$
$$\hat{U} := \frac{\partial_B}{\partial x} \hat{P} \quad \hat{U}(i,j) = \frac{1}{h}(\hat{P}(i,j) - \hat{P}(i-1,j)) \quad (i = 2,\dots,N, j = 1,\dots,N)$$

$$U_1 := \frac{\partial_B}{\partial x} \hat{P} \quad U_1(i,j) = \frac{1}{h}(P(i,j) - P(i-1,j)) \quad (i = 2,\dots,N, j = 1,\dots,N)$$

$$U := \frac{\partial_A^2}{\partial x^2} U_1 \quad U(i,j) = \begin{cases} \frac{1}{h}\left(U_1(i,j-1) + U_1(i,j+1) - 2U_1(i,j)\right) & (i = 2,\dots,N, j = 2,\dots,N-1)) \\ \frac{1}{h}\left(U_1(i,2) - U_1(i,1)\right) & j = 1 \\ \frac{1}{h}\left(-U_1(i,N) + U_1(i,N-1)\right) & j = N \end{cases}$$

则将 $U, \hat{U}$ 根据上面的式子展开，结合立刻有 $U = \hat{U}$。综上，**引理 4.2 得证**

定理 4.3. $\mathbf{B^T A^{-1} B}$ 的特征值只有 1 和 0, 且其中 0 特征值的重数为 1.

定理 4.3 的证明: 由引理 4.2

$$\mathbf{AB} = \mathbf{BB^T B}$$

回忆 $C := \mathbf{B^T A^{-1} B}$, 利用上面的式子，我们有

$$\mathbf{B^T B} = (\mathbf{B^T A^{-1} B})\mathbf{B^T B} = C\mathbf{B^T B}$$

注意到线性代数中的熟知结论以及**引理 4.1**, 有

$$rank(\mathbf{B^T B}) = rank(\mathbf{B}) = N^2 - 1$$

这说明 C 的特征值为 1 的子空间维数至少为 $N^2 - 1$, 而由 $\mathbf{A}$ 正定对称，所以

$$rank(\mathbf{B^T A^{-1} B}) = rank(\mathbf{B}) = N - 1$$

所以 C 的特征值只有 0 或 1，且其中特征值为 0 的特征子空间维数为 1。**证毕!**

有了上面的准备工作，我们可以开始讨论最优 α 的选取问题。注意到原 Stokes 方程以及离散 Stokes 方程中压力 P(每个元素) 整体加上一个常数仍然是解，因此我们可以商掉常值矩阵 (函数) 张成的空间讨论收敛性问题，在这个商空间上，C 的特征值只有 1，则 $I - C$ 的谱半径相应为 0，达到最好的效果 (事实上 $I - \alpha\mathbf{B^T A^{-1} B}$ 的特征值为 $1 - \alpha, 1$, 在 $\alpha = 1$ 时变为秩 1 矩阵，此时收敛最快), 综上最优参数 $\alpha_* = 1$.

4.4 数值结果

根据上面三个小节的讨论，我们已经完成了 Uzawa 迭代法的复现，以下是我得到的数值结果，与第一问类似，我统计了完全使用 GPU 进行计算和使用 CPU 进行计算的运行时间，此外我还额外尝试了 $N = 1024, 2048$ 的情形，实验证明使用 GPU 加速在这样大规模的问题下求解时间仍在可接受范围内，而 CPU 版本则求解时间过长。

表 8: Uzawa 迭代数值结果

N	64	128	256	512	1024	2048
求解时间-GPU(s)	1.269186	1.822200	3.690446	7.068894	26.167055	177.178170
求解时间-CPU(s)	0.024223	0.139926	0.750014	11.267029	95.587917	751.479234
Uzawa 迭代次数	2	2	2	2	2	2
共轭梯度法迭代次数	(192, 129)	(384, 258)	(768, 517)	(1536, 1032)	(3072, 1992)	(6144, 3984)
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3349e-5	5.8372e-6	1.4593e-6

这里得到的误差与第一问得到的几乎完全一致，再次说明了误差应当与 $\frac{1}{N^2}$ 成正比.

5 第三题

5.1 问题描述

分别取 $N = 64, 128, 256, 512, 1024, 2048$，以 Inexact Uzawa Iteration Method 为迭代法求解上述离散问题，停机标准为 $\|r_h\|_2/\|r_0\|_2 \leq 10^{-8}$，其中以 V-cycle 多重网格方法为预条件子求解每一步的子问题 $\mathbf{A}\mathbf{U}_{k+1} = \mathbf{F} - \mathbf{B}P_k$，对不同的 $\alpha, \tau, \nu_1, \nu_2, L$，比较外循环的迭代次数和 CPU 时间，并计算误差

$$e_N = h\left(\sum_{j=1}^{N}\sum_{i=2}^{N}\left|u_{i,j+\frac{1}{2}} - u((i-1)h, (j-\frac{1}{2})h)\right|^2 + \sum_{j=2}^{N}\sum_{i=1}^{N}\left|v_{i+\frac{1}{2},j} - v((i-\frac{1}{2})h, (j-1)h)\right|^2\right)^{\frac{1}{2}}.$$

5.2 Inexact Uzawa Iteration Method

与第一问不同的是，这里 Vcycle 多重网格求解子问题 $\mathbf{A}\mathbf{U}_{k+1} = \mathbf{F} - \mathbf{B}P_k$ 时没用使用 (对称) 红黑 GS 迭代，而是使用了 (对称) 顺序 GS 迭代，因为通过实验，(对称) 红黑 GS 迭代在这个问

题中的收敛效率相当程度上慢于 (对称) 顺序 GS 迭代，即需要更多次迭代才能收敛。这里顺序 GS 迭代的遍历顺序 U 按照先列后行 (列标从小到大，固定列标时，行标从小到大遍历), V 按照先行后列的顺序遍历，遍历完第一遍后，按照第一轮遍历的顺序的逆序，反向更新 U, V. 我实现的 Inexact Uzawa 迭代的伪代码如下:

Algorithm 3 Inexact Uzawa 迭代法

Require: P_0, $k = 0$.

while $\|r_h\|_2/\|r_0\|_2 > 10^{-8}$ **do**

    以 V-cycle 预优共轭梯度法求解 $\mathbf{AU}_{k+1} = \mathbf{F} - \mathbf{B}P_k$，得到近似解 $\hat{\mathbf{U}}_{k+1}$.

    更新压力 $P_{k+1} = P_k + \alpha(\mathbf{B^T}\hat{\mathbf{U}}_{k+1})$.

end while

其中 V-cycle 预优共轭梯度法如下:

Algorithm 4 V-cycle 预优共轭梯度法

Require: x

$k = 0; r = b - \mathbf{A}x; \rho = r^T r$

for k = 1:PCG_ite_max **do**

    if $\left(\sqrt{r^T r} < \tau\|B^T\hat{U}_k\|_2\right)$ 或 $\left(\sqrt{r^T r} < \varepsilon\|b\|_2\right)$ **then**

        break

    end if

    以对称顺序 Gauss-Seidel 迭代法为磨光子，利用 Vcycle 多重网格方法求解 $Az = r$.

    if $k = 1$ **then**

        $p = z; \quad \rho = r^T z;$

    else

        $\tilde{\rho} = \rho; \quad \rho = r^T z; \quad \beta = \rho/\tilde{\rho}; \quad p = z + \beta p;$

    end if

    $w = \mathbf{A}p; \quad \alpha = \rho/p^T w; \quad x = x + \alpha p; \quad r = r - \alpha w;$

end for

上面伪代码中 τ 的含义见下面的解释：设 $\hat{\mathbf{U}}_{k+1}$ 是方程 $\mathbf{AU}_{k+1} = \mathbf{F} - \mathbf{B}P_k$ 的近似解。定义

$$\delta_k = \mathbf{A}\hat{\mathbf{U}}_{k+1} - \mathbf{F} + \mathbf{B}P_k$$

若总有

$$\|\delta_k\|_2 \leq \tau \|\mathbf{B}^{\mathbf{T}}\hat{\mathbf{U}}_k\|_2$$

根据课上所讲, 当 τ 充分小时, 上述迭代方法是收敛的。

对预优共轭梯度法的 Vcycle 多重网格，其至多求解 $Vcycle\_ite\_max$ 轮, 此外当多重网格当前残差 2 范数小于初始残差 2 范数乘 $Vcycle\_error$ 时，也跳出 Vcycle 迭代。

此外我还在 V-cycle 预优共轭梯度法中引入了参数 ε, 这是预优共轭梯度法中的收敛条件，经过实验，它的引入能够加速收敛 (有时 $\mathbf{B^T U_k}$ 已经足够小，再乘上 τ 导致算法难已收敛)。

综上所述,我的算法能够调节的参数共有以下几个:$\nu_1, \nu_2, \alpha, L, \tau, \varepsilon, PCG\_ite\_max, Vcycle\_ite\_max$, $Vcycle\_error$。其中 PCG_ite_max 对收敛速度不起本质作用，其设置是为了防止预优共轭梯度算法未能收敛导致程序陷入死循环。

5.3 数值结果

表 9: $\nu_1 = 4, \nu_2 = 4, \alpha = 1, N/L = 4, \tau = 1e-3, \varepsilon = 1e-6,$
$PCG\_ite\_max = 2, Vcycle\_ite\_max = 2, Vcycle\_error = 1e-6$

N	64	128	256	512	1024	2048	4096
求解时间-CPU(s)	0.026898	0.034526	0.133297	0.666368	3.792414	22.239338	111.622652
Inexact Uzawa 迭代次数	2	2	2	2	2	2	2
PCG 迭代次数	2, 2	2, 2	2, 2	2, 2	2, 2	2, 2	2, 2
误差	1.4951e-3	3.7364e-4	9.3407e-5	2.3357e-5	5.8456e-6	1.4677e-6	3.7325e-7

表 10: $v_1 = 4, v_2 = 4, \alpha = 1, N/L = 8, \tau = 1e-3, \varepsilon = 1e-6,$
$PCG\_ite\_max = 2, Vcycle\_ite\_max = 3, Vcycle\_error = 1e-6$

N	64	128	256	512	1024	2048	4096
求解时间-CPU(s)	0.032328	0.085980	0.290237	1.443217	8.137539	49.195681	175.623284
Inexact Uzawa 迭代次数	3	3	3	3	3	3	2
PCG 迭代次数	2, 2, 2	2, 2, 2	2, 2, 2	2, 2, 2	2, 2, 2	2, 2, 2	2, 2
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3350e-5	5.8380e-6	1.4601e-6	5.8166

表 11: $v_1 = 2, v_2 = 2, \alpha = 1, N/L = 2, \tau = 1e-3, \varepsilon = 1e-6,$
$PCG\_ite\_max = 10, Vcycle\_ite\_max = 3, Vcycle\_error = 1e-6$

N	64	128	256	512	1024	2048	4096
求解时间-CPU(s)	0.024467	0.032684	0.116778	0.588672	3.187337	18.480399	91.595735
Inexact Uzawa 迭代次数	2	2	2	2	2	2	2
PCG 迭代次数	2, 2	2, 2	2, 2	2, 2	2, 2	2, 2	2, 2
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3350e-5	5.8377e-6	1.4597e-6	3.6524e-7

表 12: $v_1 = 2, v_2 = 2, \alpha = 0.95, N/L = 2, \tau = 1e-3, \varepsilon = 1e-8,$
$PCG\_ite\_max = 10, Vcycle\_ite\_max = 2, Vcycle\_error = 1e-6$

N	64	128	256	512	1024	2048	4096
求解时间-CPU(s)	0.061697	0.177659	0.463412	1.930474	10.646236	47.124055	256.556398
Inexact Uzawa 迭代次数	6	6	6	5	5	5	5
PCG 迭代次数	3, 5 个 4	3, 5 个 4	3, 5 个 4	3, 4 个 4	3, 4 个 4	5 个 3	5 个 3
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3350e-5	5.8398e-6	1.4696e-6	4.0405e-7

表 13: $v_1 = 2, v_2 = 2, \alpha = 1, N/L = 2, \tau = 1e-3, \varepsilon = 1e-8,$
$PCG\_ite\_max = 10, Vcycle\_ite\_max = 2, Vcycle\_error = 1e-6$

N	64	128	256	512	1024	2048	4096
求解时间-CPU(s)	0.041779	0.050313	0.143194	0.708133	3.808762	19.228862	117.304201
Inexact Uzawa 迭代次数	2	2	2	2	2	2	2
PCG 迭代次数	3, 4	3, 4	3, 4	3, 4	3, 4	3, 3	3,3
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3349e-5	5.8372e-6	1.4593e-6	3.5485e-7

表 14: $v_1 = 2, v_2 = 2, \alpha = 1.05, N/L = 2, \tau = 1e-3, \varepsilon = 1e-8,$
$PCG\_ite\_max = 10, Vcycle\_ite\_max = 2, Vcycle\_error = 1e-6$

N	64	128	256	512	1024	2048	4096
求解时间-CPU(s)	0.054376	0.142346	0.458875	1.906460	10.490952	59.588960	242.850367
Inexact Uzawa 迭代次数	6	6	6	5	5	5	5
PCG 迭代次数	3, 5 个 4	3, 5 个 4	3, 5 个 4	3, 4 个 4	3, 4 个 4	5 个 3	5 个 3
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3349e-5	5.8398e-6	1.4696e-6	4.0405e-7

表 15: $v_1 = 3, v_2 = 2, \alpha = 1, N/L = 4, \tau = 1e-1, \varepsilon = 1e-6,$
$PCG\_ite\_max = 10, Vcycle\_ite\_max = 3, Vcycle\_error = 1e-6$

N	64	128	256	512	1024	2048	4096
求解时间-CPU(s)	0.026522	0.047398	0.172523	0.824278	4.670473	28.225284	115.017383
Inexact Uzawa 迭代次数	3	3	3	3	3	3	2
PCG 迭代次数	1, 2, 2	1, 2, 2	1, 2, 2	1, 2, 2	1, 2, 2	1, 2, 2	2,2
误差	1.4951e-3	3.7363e-4	9.3399e-5	2.3349e-5	5.8373e-6	1.4594e-6	3.6492e-7

5.3.1 数值结果总结

从上面给出的几组数值结果可以给出几个参数的合理选择，首先通过比较表 12,13,14 容易发现 α 仍然是取 1 最好，稍微偏离 1 一点都会导致相当大的额外迭代次数，比较让我奇怪的是 $N = 4096$ 且 $\alpha \neq 1$ 时的计算误差相比其他结果的有较大的差异，这说明 α 的错误选择还会导致可能的误差放大。

而较小的误差与我们之前的得到的结果并无区别，这再一次验证了误差的收敛阶为 $\frac{1}{N^2}$

再比较表 9,10, 容易得到 N/L 最好取为 2, 4 而不要更大，同时观察其他数据发现 N/L 取 2 还是 4 对于迭代次数影响极小。最后提出上面说的 3 组参数选择有误的数据，从整体分析的角度看容易发现，Inexact Uzawa 迭代在正确的超参数下收敛速度其实相当快，如表 9,11 只需两次迭代便可收敛。此外还能定性分析迭代次数与超参数之间的关系。大致如下

1. τ, ε 的增加会导致 PCG 的迭代次数增加

2. 增加预优共轭梯度法的多重网格的迭代次数，能够使外循环的迭代次数以及 PCG 迭代次数减少。

3. 增大 ν_1, ν_2 在一定范围内能够减少迭代次数。

以及我发现了一个看起来比较奇怪的地方，似乎 N 越大需要的迭代次数越小，我没有想到比较合理的解释。

最后，我认为最好的一组超参数为表 11 的超参数 $\nu_1 = 2, \nu_2 = 2, \alpha = 1, N/L = 2, \tau = 1e-3, \varepsilon = 1e-6, PCG\_ite\_max = 10, Vcycle\_ite\_max = 3, Vcycle\_error = 1e-6$，在这一组超参下，Inexact Uzawa 迭代的速度已经较为显著地超过了 CPU 版本的 DGS 迭代，这体现出了 Inexact Uzawa 迭代地优越性。