计算流体力学作业 5

郑恒 2200011086

2025 年 5 月 23 日

1 问题介绍

在单位正方形内，求解不可压缩流动。仅上边界为水平运动边界，其余边界均为固定壁面。上边界速度分布设为

$$u(x) = \sin^2(\pi x)$$

该分布在角点处函数值和导数均为零，保证了速度场的连续性。通过数值方法计算流场，考察不同位置的速度剖面、主涡涡心位置和流函数数值等。使用 Python 语言，设运动粘度 $\nu = 0.001$，画出流线图，并对主要物理量进行讨论。

2 算法原理

对于不可压缩流体，流场满足的方程为 Navier-Stokes 方程。对于二维不可压缩流体，Navier-Stokes 方程可以写成如下形式：

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{1}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{1}{\rho}\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{2}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{3}$$

其中，u 和 v 分别是流体在 x 和 y 方向的速度分量，ρ 是流体密度，p 是压力，ν 是运动粘度。在本问题中，我们需要求解流体的速度场和压力场。我们使用有限差分法来离散化 Navier-Stokes 方程，并使用迭代方法来求解离散方程。我们将流场离散化为一个网格，并在每个网格点上计算速度和压力。我们可以使用显式时间步进方法来更新速度和压力场。具体步骤如下：

1. 初始化网格和边界条件
2. 计算速度场和压力场
3. 更新速度场和压力场
4. 重复步骤 2 和 3，直到收敛
5. 计算主涡涡心位置和流函数值
6. 绘制流线图和速度剖面
7. 输出结果

在本问题中，密度 ρ 被视为常数，不随空间和时间变化。由于 ρ 为常数，实际计算时常常将其归一化处理，即令 $\rho = 1$，以简化方程和计算。

在迭代计算速度场合压力场时，我们采用投影法，以下是投影法的步骤公式：

1. 预测步：先不考虑压力项，计算临时速度场 $\mathbf{u}^*$：

$$\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -(\mathbf{u}^n \cdot \nabla)\mathbf{u}^n + \nu \nabla^2 \mathbf{u}^n \tag{4}$$

   其中，$\mathbf{u}^n$ 为当前时刻速度，$\mathbf{u}^*$ 为预测速度。

2. 压力泊松方程：为保证不可压缩性，对预测速度场求解压力修正：

$$\nabla^2 p^{n+1} = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^* \tag{5}$$

3. 修正步：用新压力修正速度场，得到下一个时刻的速度：

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \Delta t \nabla p^{n+1} \tag{6}$$

   这样可以保证 $\nabla \cdot \mathbf{u}^{n+1} = 0$，满足不可压缩条件。

在实际算法中，我们需要对投影法的每一步进行离散化。对于上述投影法的三个步骤，采用中心差分离散化，差分格式如下：

1. 预测步：

$$\begin{aligned}\frac{u^*_{i,j}-u^n_{i,j}}{\Delta t}=&-u^n_{i,j}\frac{u^n_{i+1,j}-u^n_{i-1,j}}{2\Delta x}-v^n_{i,j}\frac{u^n_{i,j+1}-u^n_{i,j-1}}{2\Delta y}\\&+\nu\left(\frac{u^n_{i+1,j}-2u^n_{i,j}+u^n_{i-1,j}}{\Delta x^2}+\frac{u^n_{i,j+1}-2u^n_{i,j}+u^n_{i,j-1}}{\Delta y^2}\right)\end{aligned} \tag{7}$$

对 v 分量同理。

2. 压力泊松方程：

$$\frac{p^{n+1}_{i+1,j}-2p^{n+1}_{i,j}+p^{n+1}_{i-1,j}}{\Delta x^2}+\frac{p^{n+1}_{i,j+1}-2p^{n+1}_{i,j}+p^{n+1}_{i,j-1}}{\Delta y^2}=\frac{1}{\Delta t}\left(\frac{u^*_{i+1,j}-u^*_{i-1,j}}{2\Delta x}+\frac{v^*_{i,j+1}-v^*_{i,j-1}}{2\Delta y}\right) \tag{8}$$

3. 速度修正步：

$$u^{n+1}_{i,j}=u^*_{i,j}-\Delta t\frac{p^{n+1}_{i+1,j}-p^{n+1}_{i-1,j}}{2\Delta x} \tag{9}$$

$$v^{n+1}_{i,j}=v^*_{i,j}-\Delta t\frac{p^{n+1}_{i,j+1}-p^{n+1}_{i,j-1}}{2\Delta y} \tag{10}$$

对第 2 步压力泊松方程，我们采用 JOCIBI 迭代法进行求解：

1. 初始化压力场 p^{n+1} 为零场

2. 迭代计算压力场，直到收敛：

$$p^{n+1}_{i,j}=\frac{1}{4}\left(p^{n+1}_{i+1,j}+p^{n+1}_{i-1,j}+p^{n+1}_{i,j+1}+p^{n+1}_{i,j-1}\right)-\frac{\Delta t}{4}\left(\frac{u^*_{i+1,j}-u^*_{i-1,j}}{2\Delta x}+\frac{v^*_{i,j+1}-v^*_{i,j-1}}{2\Delta y}\right) \tag{11}$$

3. 迭代停止条件：当压力场的变化量小于设定的阈值时，停止迭代

4. 更新压力场

5. 返回第 2 步

3 代码实现

模拟流场的代码如下：

首先是基本参数设定和边界条件设定，对本题正方形区域，我们不妨设置 $\delta x = \delta y = 0.01$，即网格数为 100*100。速度边界条件满足除上边界外其他边界为 0，压力边界条件为法向梯度 0。

```
import numpy as np
import matplotlib.pyplot as plt

# 模拟参数
N = 101  # 网格点数（包括边界）
h = 1.0 / (N - 1)  # 网格间距
nu = 0.001  # 运动粘度
dt = 0.0001  # 时间步长
max_iter = 1000000  # 最大迭代次数
max_p_iter = 10000      # 最大压力迭代次数
p_tol = 1e-5         # 压力残差容差
p_residual = 1.0     # 初始化残差
velocity_tol = 1e-7   # 速度收敛容差
check_interval = 100    # 收敛检查间隔
# 初始化场变量
u = np.zeros((N, N))
v = np.zeros((N, N))
p = np.zeros((N, N))

# 设置上边界的水平速度（sin²(x)）
x = np.linspace(0, 1, N)
u_top = np.sin(np.pi * x) ** 2
u[:, -1] = u_top  # 上边界条件


# 边界掩模（用于强制固定边界条件）
def apply_boundary_conditions(u, v):
```

```
    # 上边界
    u[:, -1] = u_top  # 水平速度
    v[:, -1] = 0  # 垂直速度

    # 下边界
    u[:, 0] = 0
    v[:, 0] = 0

    # 左边界
    u[0, :] = 0
    v[0, :] = 0

    # 右边界
    u[-1, :] = 0
    v[-1, :] = 0
    return u, v

# 压力边界条件函数
def apply_pressure_bc(p):
    # Neumann边界条件（法向梯度为零）
    p[0, :] = p[1, :]      # 左边界
    p[-1, :] = p[-2, :]    # 右边界
    p[:, 0] = p[:, 1]      # 下边界
    p[:, -1] = p[:, -2]    # 上边界
    return p
```

Listing 1: 基础参数设定和边界条件

然后是主程序求解流场速度分布：采用的差分算法就是我们在算法原理里提到的投影法。值得一提的是迭代收敛由两次，一次是由 u^n 到 u^{n+1} 的差值来判断，另一次是由压力场的变化量来判断。我们在代码中设置了一个阈值，当两次迭代都满足收敛条件时，才停止迭代。

```
# 主求解循环
prev_u = np.zeros_like(u)
```

```
prev_v = np.zeros_like(v)
converged = False
for iter in range(max_iter):
    # 保存前次速度场用于收敛判断
    if iter % check_interval == 0:
        prev_u[:] = u
        prev_v[:] = v
    # 临时速度场
    u_prev = u.copy()
    v_prev = v.copy()

    # 计算中间速度（扩散项 + 对流项）
    u[1:-1, 1:-1] += dt * (
            nu * (u_prev[2:, 1:-1] + u_prev[:-2, 1:-1] +
               u_prev[1:-1, 2:] + u_prev[1:-1, :-2] - 4 *
               u_prev[1:-1,1:-1]) / h ** 2
            - (u_prev[1:-1, 1:-1] * (u_prev[2:, 1:-1] -
               u_prev[:-2, 1:-1]) / (2 * h)
               + v_prev[1:-1, 1:-1] * (u_prev[1:-1, 2:] -
                  u_prev[1:-1, :-2]) / (2 * h))
    )

    v[1:-1, 1:-1] += dt * (
            nu * (v_prev[2:, 1:-1] + v_prev[:-2, 1:-1] +
               v_prev[1:-1, 2:] + v_prev[1:-1, :-2] - 4 *
               v_prev[1:-1,1:-1]) / h ** 2
            - (u_prev[1:-1, 1:-1] * (v_prev[2:, 1:-1] -
               v_prev[:-2, 1:-1]) / (2 * h)
               + v_prev[1:-1, 1:-1] * (v_prev[1:-1, 2:] -
                  v_prev[1:-1, :-2]) / (2 * h))
    )

    # 应用速度边界条件
```

```
u, v = apply_boundary_conditions(u, v)


for p_iter in range(max_p_iter):
    p_old = p.copy()

    p[1:-1, 1:-1] = (
                        (p[2:, 1:-1] + p[:-2, 1:-1] + p[1:-1, 2:] + p[1:-1, :-2])
                        - h ** 2 / (4 * dt) * (
                                (u[2:, 1:-1] - u[:-2, 1:-1]) / (2 * h) +
                                (v[1:-1, 2:] - v[1:-1, :-2]) / (2 * h)
                        )
                    ) / 4
    # 应用边界条件
    p = apply_pressure_bc(p)

    # 计算残差（仅内部点）
    p_residual = np.max(np.abs(p[1:-1, 1:-1] - p_old[1:-1, 1:-1]))

    # 收敛检查
    if p_residual < p_tol:
        print(f"压力场在 {p_iter+1} 次迭代后收敛，残差 = {p_residual:.3e}")
        break

# 最终收敛检查
if p_residual > p_tol:
    print(f"警告：压力场未在{max_p_iter}次迭代内收敛，最终残差{p_residual:.3e}")
```

```
    # 速度修正
    u[1:-1, 1:-1] -= dt * (p[2:, 1:-1] - p[:-2, 1:-1]) / (2 * h)
    v[1:-1, 1:-1] -= dt * (p[1:-1, 2:] - p[1:-1, :-2]) / (2 * h)

    # 最终应用边界条件
    u, v = apply_boundary_conditions(u, v)
    # 收敛性检查
    if iter % check_interval == 0 and iter > 0:
        # 计算速度场变化量
        du_max = np.max(np.abs(u - prev_u))
        dv_max = np.max(np.abs(v - prev_v))


        # 输出监控信息
        print(f"Iter {iter:04d}: Δu={du_max:.2e}, Δv={dv_max:.2e}")

        # 双重收敛标准
        if (du_max < velocity_tol and
                dv_max < velocity_tol):
            print(f"速度场在 {iter} 次迭代后收敛!")
            converged = True
            break
```

Listing 2: 主程序求解流场速度分布

最后是流线图的绘制和速度剖面的绘制。流线图的绘制采用了 matplotlib 库中的 streamplot 函数，速度剖面的绘制采用了 matplotlib 库中的 plot 函数。

```
def plot_analysis_results(u, v):
    #可视化分析结果
    x = np.linspace(0, 1, N)
```

```
y = np.linspace(0, 1, N)
X, Y = np.meshgrid(x, y)

# 流线图
plt.figure(figsize=(10, 8))
plt.streamplot(X, Y, u.T, v.T, density=2, color='lightgray')
plt.title('Vortex Core Locations')
plt.savefig('1_streamlines.pdf', bbox_inches='tight', dpi=300)
plt.close()

# 水平中线速度剖面
plt.figure(figsize=(10, 4))
mid_y = N // 2
absolute_velocityy = np.sqrt(u[mid_y, :] ** 2 + v[mid_y, :] ** 2)
plt.plot(x, absolute_velocityy, 'r-', linewidth=2)
plt.xlabel('x coordinate')
plt.ylabel('Velocity')
plt.title(f'Horizontal Velocity Profile @ y={y[mid_y]:.2f}')
plt.grid(True)
plt.savefig('2_horizontal_profile.pdf', bbox_inches='tight', dpi=300)
plt.close()

# 垂直中线速度剖面
plt.figure(figsize=(6, 8))
mid_x = N // 2
absolute_velocityx= np.sqrt(u[:, mid_x] ** 2 + v[:, mid_x] ** 2)
plt.plot(absolute_velocityx, y, 'b-', linewidth=2)
```

```
    plt.ylabel('y coordinate')
    plt.xlabel('Velocity')
    plt.title(f'Vertical Velocity Profile @ x={x[mid_x]:.2f
        }')
    plt.grid(True)
    plt.savefig('3_vertical_profile.pdf', bbox_inches='
        tight', dpi=300)
    plt.close()
```

Listing 3: 流线图和速度剖面的绘制

4 结果分析

通过以上代码生成的流线图和速度剖面图如下：

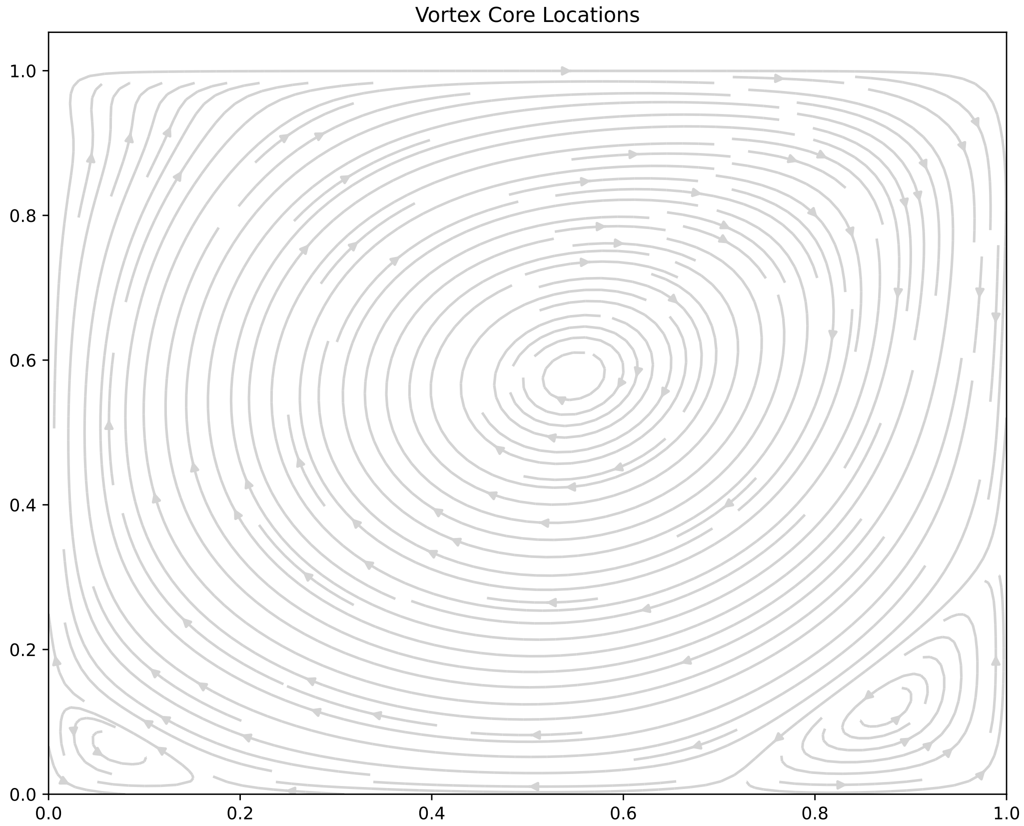

图 1: 流线图

由流场图可以看出，主涡的涡心位置在 $(0.5, 0.6)$ 左右，在左下角和右下角各有一个二次涡。其形成原因主要是流动边界层分离和粘性耗散效应。由于模拟条件雷诺数 Re 约等于 1000，流动较为平稳，流线图中没有明显的三次涡旋结构。随着雷诺数的增加，流动会变得更加复杂，可能会出现更强的涡旋结构和不稳定性。

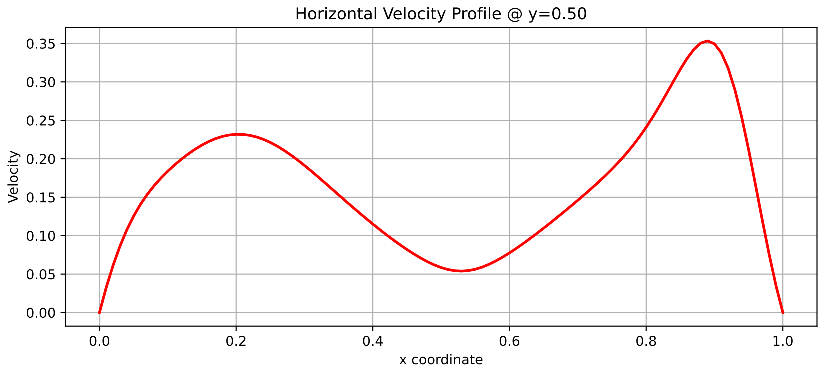

图 2: y=0.5 速度水平剖面图

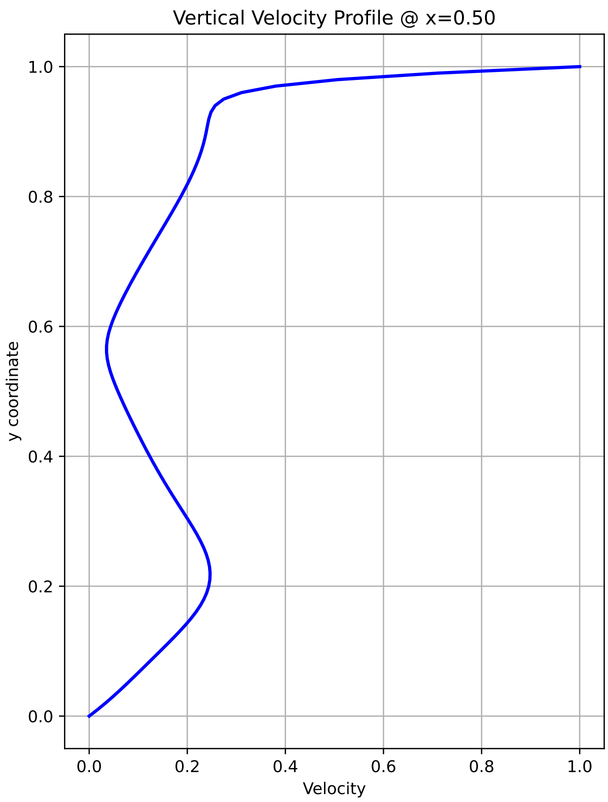

图 3: x=0.5 速度垂直剖面图

由速度剖面图可以看出，流速在上边界处达到最大值，随着距离上边界的增加，流速逐渐减小。流速在 y=0.5 处的水平剖面图中，流速在 x=0.5

处达到最大值，随着 x 的增加，流速逐渐减小。流速在 x=0.5 处的垂直剖面图中，流速在 y=0.5 处达到最大值，随着 y 的增加，流速逐渐减小。

5 AI 工具使用说明表

AI 名称	**生成代码功能**	**使用内容**
Copilot	latex 格式框架	figure 参数调整、图片插入
Deepseek	python 绘图调整	83-104、106-118 行图绘制的具体参数调整
Deepseek	gitignore 文件忽略	全由 ai 添加

6 commit 信息