

# 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：

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

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

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad (3)$$

其中,  $u$  和  $v$  分别是流体在  $x$  和  $y$  方向的速度分量,  $\rho$  是流体密度,  $p$  是压力,  $\nu$  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. 绘制流线和速度剖面
7. 输出结果

在本问题中, 密度  $\rho$  被视为常数, 不随空间和时间变化。由于  $\rho$  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  $\mathbf{u}^*$ :

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

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:

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

3. 修正步: 用新压力修正速度场, 得到下一个时刻的速度:

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

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：

$$\begin{aligned} \frac{u_{i,j}^* - u_{i,j}^n}{\Delta t} = & -u_{i,j}^n \frac{u_{i+1,j}^n - u_{i-1,j}^n}{2\Delta x} - v_{i,j}^n \frac{u_{i,j+1}^n - u_{i,j-1}^n}{2\Delta y} \\ & + \nu \left( \frac{u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n}{\Delta x^2} + \frac{u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n}{\Delta y^2} \right) \end{aligned} \quad (7)$$

对  $v$  分量同理。

2. 压力泊松方程：

$$\frac{p_{i+1,j}^{n+1} - 2p_{i,j}^{n+1} + p_{i-1,j}^{n+1}}{\Delta x^2} + \frac{p_{i,j+1}^{n+1} - 2p_{i,j}^{n+1} + p_{i,j-1}^{n+1}}{\Delta y^2} = \frac{1}{\Delta t} \left( \frac{u_{i+1,j}^* - u_{i-1,j}^*}{2\Delta x} + \frac{v_{i,j+1}^* - v_{i,j-1}^*}{2\Delta y} \right) \quad (8)$$

3. 速度修正步：

$$u_{i,j}^{n+1} = u_{i,j}^* - \Delta t \frac{p_{i+1,j}^{n+1} - p_{i-1,j}^{n+1}}{2\Delta x} \quad (9)$$

$$v_{i,j}^{n+1} = v_{i,j}^* - \Delta t \frac{p_{i,j+1}^{n+1} - p_{i,j-1}^{n+1}}{2\Delta y} \quad (10)$$

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：

$$p_{i,j}^{n+1} = \frac{1}{4} (p_{i+1,j}^{n+1} + p_{i-1,j}^{n+1} + p_{i,j+1}^{n+1} + p_{i,j-1}^{n+1}) - \frac{\Delta t}{4} \left( \frac{u_{i+1,j}^* - u_{i-1,j}^*}{2\Delta x} + \frac{v_{i,j+1}^* - v_{i,j-1}^*}{2\Delta y} \right) \quad (11)$$

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  $100 \times 100$ 。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。

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # 模拟参数
5 N = 101 # 网格点数（包括边界）
6 h = 1.0 / (N - 1) # 网格间距
7 nu = 0.001 # 运动粘度
8 dt = 0.0001 # 时间步长
9 max_iter = 1000000 # 最大迭代次数
10 max_p_iter = 10000 # 最大压力迭代次数
11 p_tol = 1e-5 # 压力残差容差
12 p_residual = 1.0 # 初始化残差
13 velocity_tol = 1e-7 # 速度收敛容差
14 check_interval = 100 # 收敛检查间隔
15 # 初始化场变量
16 u = np.zeros((N, N))
17 v = np.zeros((N, N))
18 p = np.zeros((N, N))
19
20 # 设置上边界的水平速度 ( $\sin^2(x)$ )
21 x = np.linspace(0, 1, N)
22 u_top = np.sin(np.pi * x) ** 2
23 u[:, -1] = u_top # 上边界条件
24
25
26 # 边界掩模（用于强制固定边界条件）
27 def apply_boundary_conditions(u, v):
```

```

28     # 上边界
29     u[:, -1] = u_top # 水平速度
30     v[:, -1] = 0 # 垂直速度
31
32     # 下边界
33     u[:, 0] = 0
34     v[:, 0] = 0
35
36     # 左边界
37     u[0, :] = 0
38     v[0, :] = 0
39
40     # 右边界
41     u[-1, :] = 0
42     v[-1, :] = 0
43     return u, v
44
45 # 压力边界条件函数
46 def apply_pressure_bc(p):
47     # Neumann边界条件（法向梯度为零）
48     p[0, :] = p[1, :] # 左边界
49     p[-1, :] = p[-2, :] # 右边界
50     p[:, 0] = p[:, 1] # 下边界
51     p[:, -1] = p[:, -2] # 上边界
52     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。

```

1 # 主求解循环
2 prev_u = np.zeros_like(u)

```

```

3 prev_v = np.zeros_like(v)
4 converged = False
5 for iter in range(max_iter):
6     # 保存前次速度场用于收敛判断
7     if iter % check_interval == 0:
8         prev_u[:] = u
9         prev_v[:] = v
10    # 临时速度场
11    u_prev = u.copy()
12    v_prev = v.copy()
13
14    # 计算中间速度 (扩散项 + 对流项)
15    u[1:-1, 1:-1] += dt * (
16        nu * (u_prev[2:, 1:-1] + u_prev[:-2, 1:-1] +
17              u_prev[1:-1, 2:] + u_prev[1:-1, :-2] - 4 *
18              u_prev[1:-1, 1:-1]) / h ** 2
19        - (u_prev[1:-1, 1:-1] * (u_prev[2:, 1:-1] -
20              u_prev[:-2, 1:-1]) / (2 * h)
21          + v_prev[1:-1, 1:-1] * (u_prev[1:-1, 2:] -
22              u_prev[1:-1, :-2]) / (2 * h))
23    )
24
25    v[1:-1, 1:-1] += dt * (
26        nu * (v_prev[2:, 1:-1] + v_prev[:-2, 1:-1] +
27              v_prev[1:-1, 2:] + v_prev[1:-1, :-2] - 4 *
28              v_prev[1:-1, 1:-1]) / h ** 2
29        - (u_prev[1:-1, 1:-1] * (v_prev[2:, 1:-1] -
30              v_prev[:-2, 1:-1]) / (2 * h)
31          + v_prev[1:-1, 1:-1] * (v_prev[1:-1, 2:] -
32              v_prev[1:-1, :-2]) / (2 * h))
33    )
34
35    # 应用速度边界条件

```

```

28     u, v = apply_boundary_conditions(u, v)
29
30
31     for p_iter in range(max_p_iter):
32         p_old = p.copy()
33
34         p[1:-1, 1:-1] = (
35             (p[2:, 1:-1] + p[:-2, 1:-1] + p
36              [1:-1, 2:] + p[1:-1, :-2])
37             - h ** 2 / (4 * dt) * (
38                 (u[2:, 1:-1] - u[:-2,
39                  1:-1]) / (2 * h) +
40                 (v[1:-1, 2:] - v[1:-1,
41                  :-2]) / (2 * h)
42             )
43             ) / 4
44
45         # 应用边界条件
46         p = apply_pressure_bc(p)
47
48         # 计算残差（仅内部点）
49         p_residual = np.max(np.abs(p[1:-1, 1:-1] - p_old
50                                  [1:-1, 1:-1]))
51
52         # 收敛检查
53         if p_residual < p_tol:
54             print(f"压力场在 {p_iter+1} 次迭代后收敛，残差
55                   = {p_residual:.3e}")
56             break
57
58     # 最终收敛检查
59     if p_residual > p_tol:
60         print(f"警告：压力场未在{max_p_iter}次迭代内收敛，
61               最终残差{p_residual:.3e}")

```

```

55     # 速度修正
56     u[1:-1, 1:-1] -= dt * (p[2:, 1:-1] - p[:-2, 1:-1]) / (2
        * h)
57     v[1:-1, 1:-1] -= dt * (p[1:-1, 2:] - p[1:-1, :-2]) / (2
        * h)
58
59     # 最终应用边界条件
60     u, v = apply_boundary_conditions(u, v)
61     # 收敛性检查
62     if iter % check_interval == 0 and iter > 0:
63         # 计算速度场变化量
64         du_max = np.max(np.abs(u - prev_u))
65         dv_max = np.max(np.abs(v - prev_v))
66
67
68         # 输出监控信息
69         print(f"Iter {iter:04d}: Δu={du_max:.2e}, Δv={
            dv_max:.2e}")
70
71         # 双重收敛标准
72         if (du_max < velocity_tol and
73             dv_max < velocity_tol):
74             print(f"速度场在 {iter} 次迭代后收敛!")
75             converged = True
76             break

```

Listing 2: 主程序求解流场速度分布

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

```

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

```



```

4     y = np.linspace(0, 1, N)
5     X, Y = np.meshgrid(x, y)
6
7     # 流线图
8     plt.figure(figsize=(10, 8))
9     plt.streamplot(X, Y, u.T, v.T, density=2, color='
        lightgray')
10    plt.title('Vortex Core Locations')
11    plt.savefig('1_streamlines.pdf', bbox_inches='tight',
        dpi=300)
12    plt.close()
13
14    # 水平中线速度剖面
15    plt.figure(figsize=(10, 4))
16    mid_y = int(N / 2)
17    absolute_velocityy = np.sqrt(u[:, mid_y] ** 2 + v[:,
        mid_y] ** 2)
18    plt.plot(x, absolute_velocityy, 'r-', linewidth=2)
19    plt.xlabel('x coordinate')
20    plt.ylabel('Velocity')
21    plt.title(f'Horizontal Velocity Profile @ y={y[mid_y
        ]:.2f}')
22    plt.grid(True)
23    plt.savefig('2_horizontal_profile.pdf', bbox_inches='
        tight', dpi=300)
24    plt.close()
25
26    # 垂直中线速度剖面
27    plt.figure(figsize=(6, 8))
28    mid_x = int(N / 2)
29    absolute_velocityx = np.sqrt(u[mid_x, :] ** 2 + v[mid_x,
        :] ** 2)
30    plt.plot(absolute_velocityx, y, 'b-', linewidth=2)

```

```
31 plt.ylabel('y coordinate')
32 plt.xlabel('Velocity')
33 plt.title(f'Vertical Velocity Profile @ x={x[mid_x]:.2f
    }')
34 plt.grid(True)
35 plt.savefig('3_vertical_profile.pdf', bbox_inches='
    tight', dpi=300)
36 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：

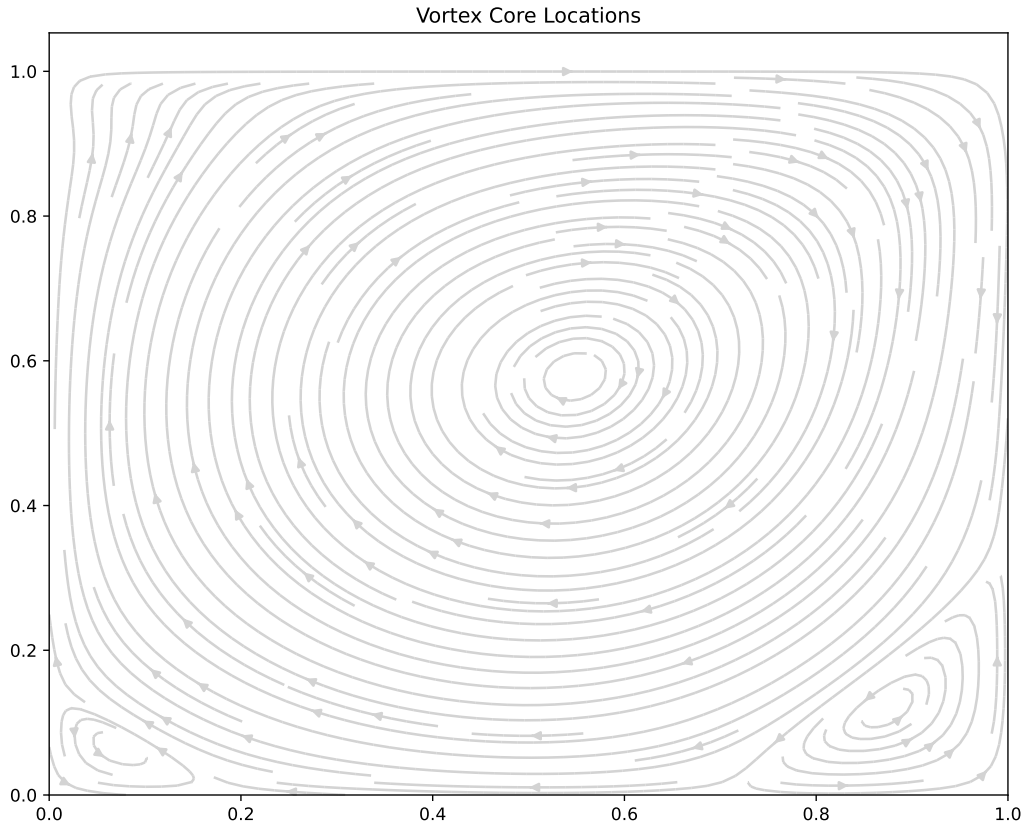


图 1: 流线图

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。

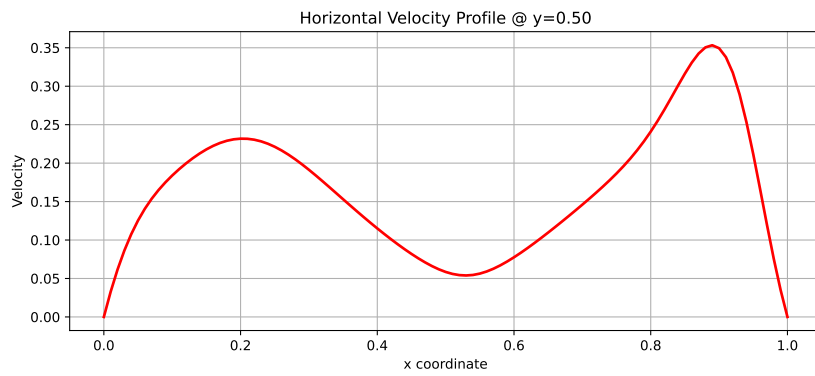


图 2:  $y=0.5$  速度水平剖面图

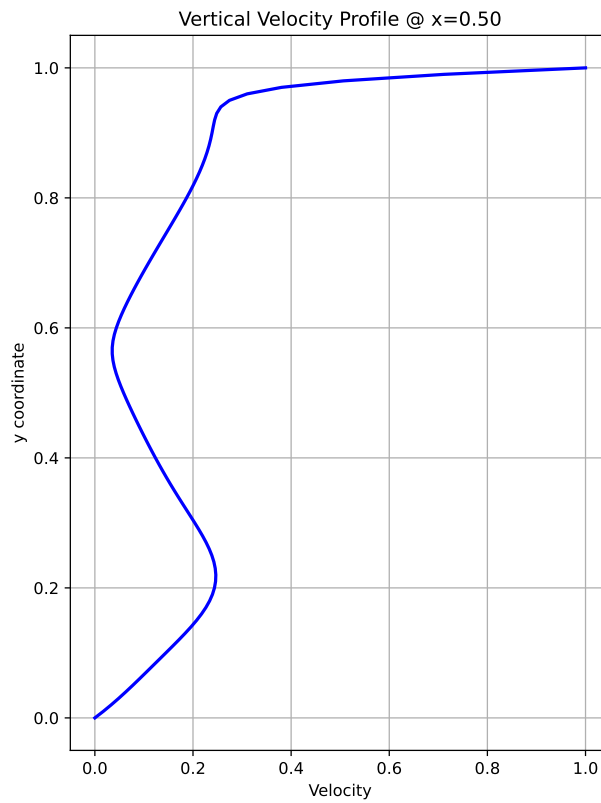


图 3:  $x=0.5$  速度垂直剖面图

由速度垂直剖面图可以看出，流速在上边界处达到最大值，随着距离上边界的增加，流速减小，后增大，最后减到 0。从速度水平剖面图可以看

出，流速在中间（主涡点）处达到一个极小值，左右满足边界条件为 0。

## 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	68-90 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

commit 图如下：

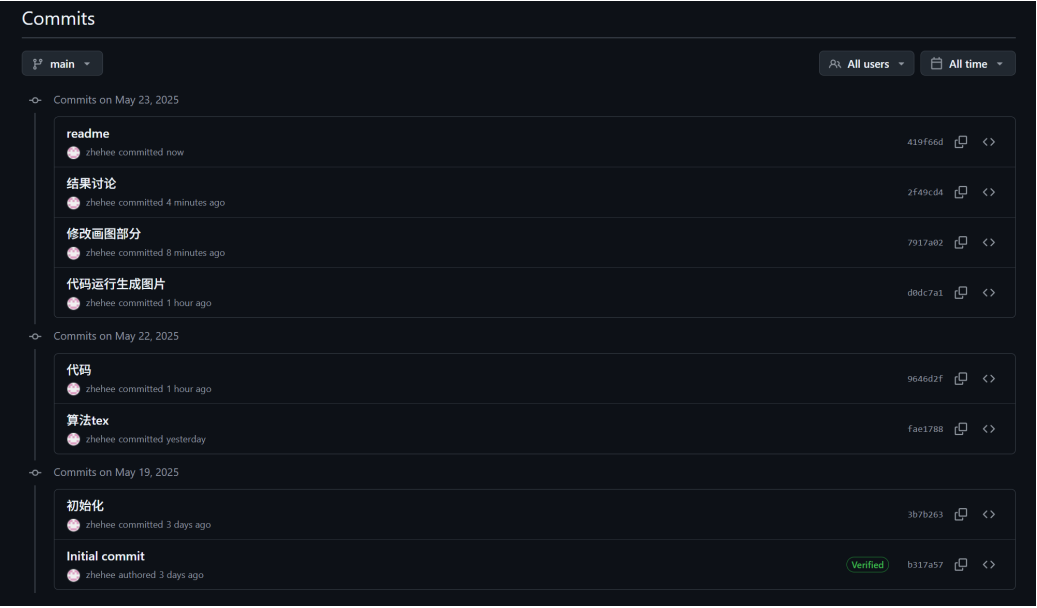


图 4: commits 截图