CFD-HW4

李张鑫 2200011085 工学院

一、 数理算法原理

（1） 控制方程与边界条件

二维稳态无内热源热传导问题，服从拉普拉斯方程：

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0$$

边界条件：左侧长边（15cm）：T=100℃

右侧、上侧、下侧：T=20℃

（2） 网格离散化

将平板划分为$N_x \times N_y$的均匀网格，网格步长：

$$\Delta x=\frac{15}{N_x-1}cm$$

$$\Delta y=\frac{12}{N_y-1}cm$$

内部节点温度$T_{i\,j}$由周围节点决定，边界节点固定为给定温度

（3） 迭代方法

使用逐次超松弛（SOR）方法

由中心差分公式

$$\frac{\partial^2 T}{\partial x^2}\approx\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{(\Delta x)^2}$$

$$\frac{\partial^2 T}{\partial y^2}\approx\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta y)^2}$$

$$\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{(\Delta x)^2}+\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta y)^2}=0$$

令 $\Delta x=\Delta y$ 代入拉普拉斯方程

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0$$

则能得到迭代公式：

$$T_{i,j}^{\text{new}} = (1-\omega)T_{i,j}^{\text{old}} + \frac{\omega}{4}\left(T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}\right)$$

收敛条件：$max\left|T^{\text{new}} - T^{\text{old}}\right| < \epsilon$(如 $\epsilon = 0.01°\text{C}$)。

二、 具体任务实现

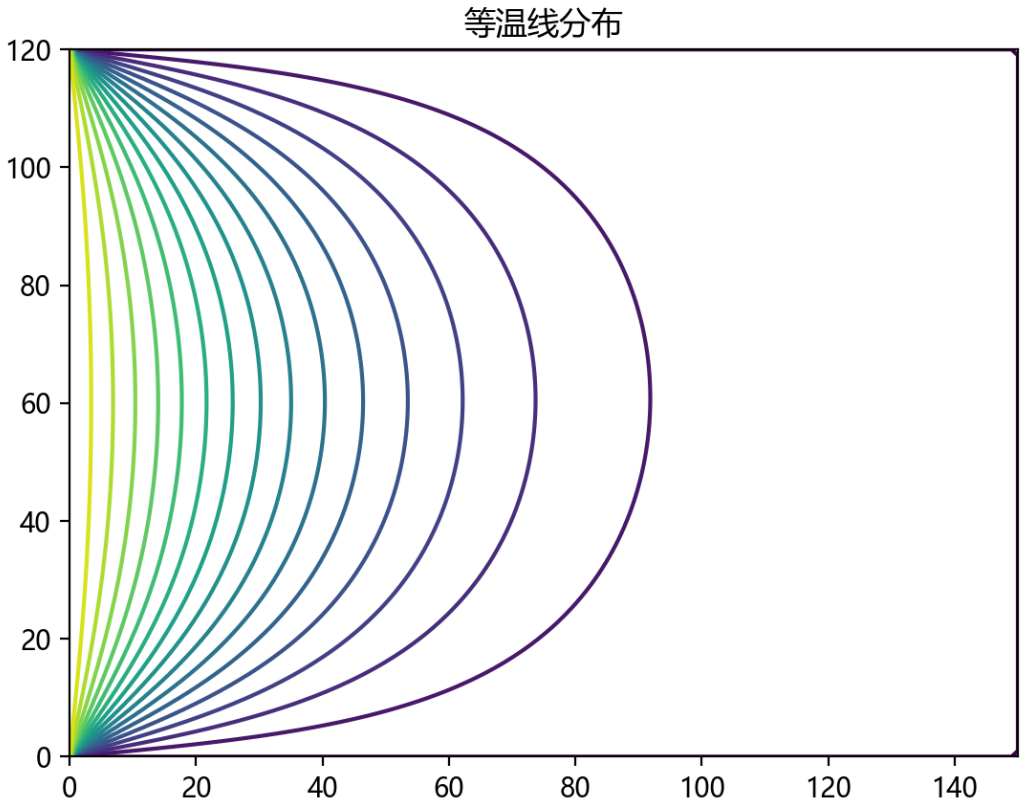
（1） 使用迭代法计算板内的稳定场

主体代码如图所示：

选择网格划分 151*121，进行逐次超松弛方法迭代

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

def solve_heat_equation(Nx, Ny, omega, epsilon=1e-4):
    T = np.ones((Ny, Nx)) * 20  # 初始化
    T[:, 0] = 100  # 左侧边界条件
    max_iter = 1000
    for _ in range(max_iter):
        max_error = 0
        for j in range(1, Nx-1):
            for i in range(1, Ny-1):
                old = T[i, j]
                T[i, j] = (1-omega)*old + omega*0.25*(T[i+1,j] + T[i-1,j] + T[i,j+1] + T[i,j-1])
                max_error = max(max_error, abs(T[i,j]-old))
        if max_error < epsilon:
            break
    return T, _

# 示例调用：计算151×121网格，ω=1.5
T, iterations = solve_heat_equation(151, 121, 1.5)

# 绘制等温线
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']  # 微软雅黑
plt.rcParams['axes.unicode_minus'] = False  # 解决负号显示问题
plt.contour(T, levels=np.arange(20, 101, 5))
plt.title("等温线分布")
plt.show()
```

等温线所示：

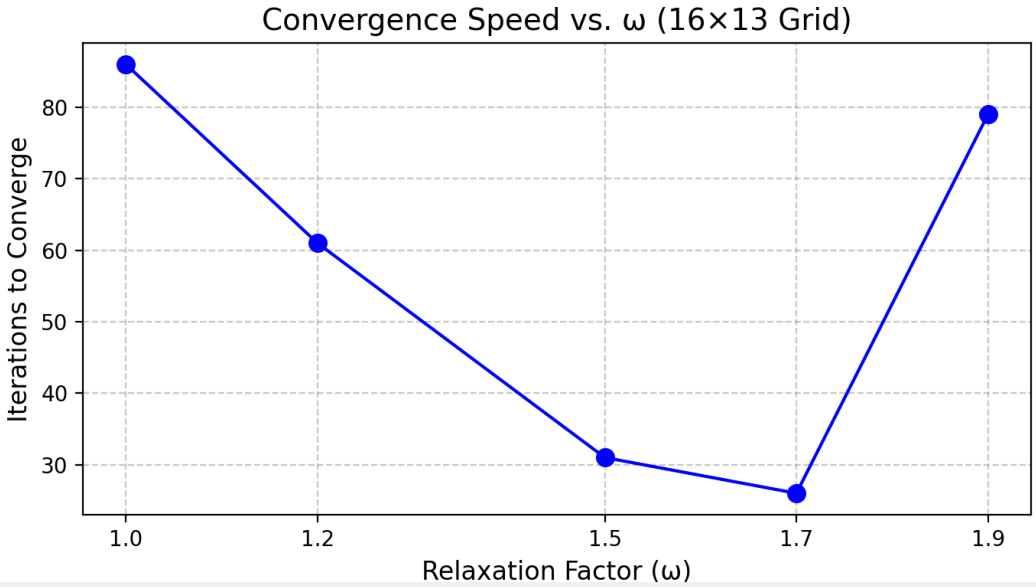


（2） 使用迭代法计算板内的稳定场

主体代码：

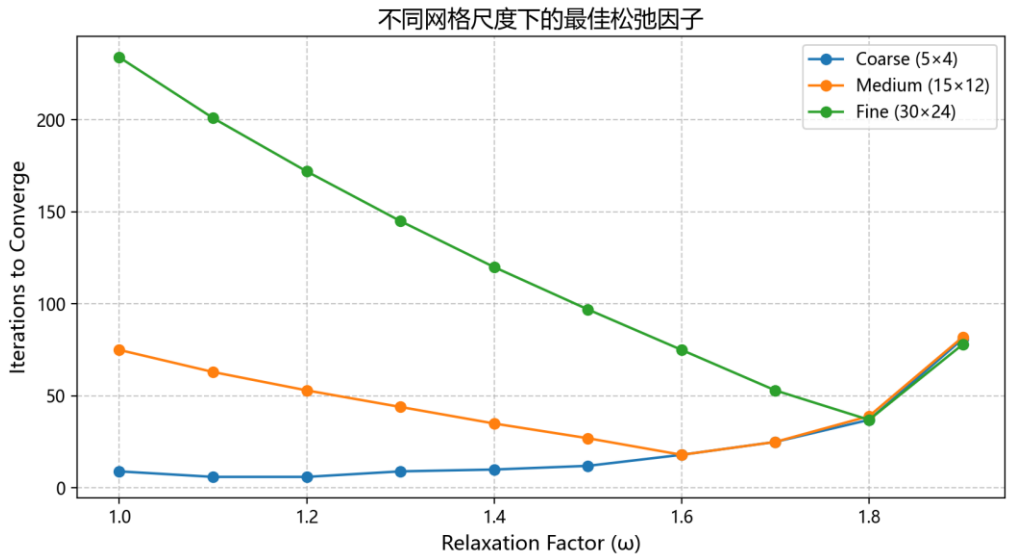
```
# 测试不同松弛因子的收敛速度
omegas = [1.0, 1.2, 1.5, 1.7, 1.9]
iterations_list = []

for omega in omegas:
    _, iterations = solve_sor(Nx=16, Ny=13, omega=omega, eps=1e-2)
    iterations_list.append(iterations)
    print(f"ω={omega}: {iterations}次迭代")

# 绘制收敛曲线
plt.figure(figsize=(8, 4))
plt.plot(omegas, iterations_list, 'bo-', markersize=8)
plt.xlabel("Relaxation Factor (ω)", fontsize=12)
plt.ylabel("Iterations to Converge", fontsize=12)
plt.title("Convergence Speed vs. ω (16×13 Grid)", fontsize=14)
plt.grid(True, linestyle='--', alpha=0.7)
plt.xticks(omegas)
plt.show()
```

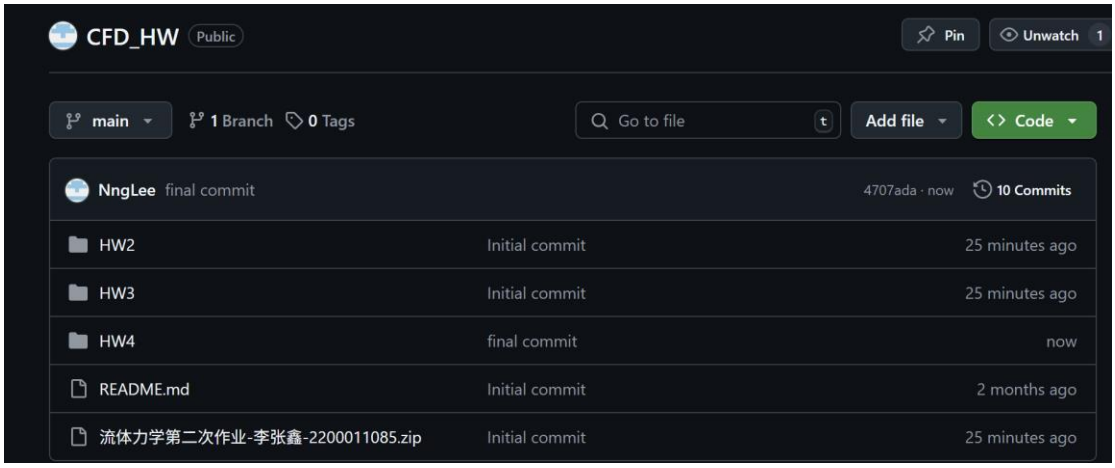
结果：在网格尺度为 16*13 时，最佳 omega 为 1.7



（3） 根据不同网格尺度，比较最佳 omega

利用第二问的算法，分别算出不同网格尺度下的最佳 omega 值，结果如图所示





Github 提交记录

三、 AI 工具使用说明

使用的 AI 工具名称：deepseek

核心算法部分比例:100%