# 计算概论：：流体世界
# Project Fluid World

徐智怡　　黄康靖　　沈钟灵

摘要

为了进一步在实践中掌握c++语言，理解面向对象编程；应用流体力学知识并初步了解一些数值方法的相关内容。 以c++语言为编程工具，数值模拟并可视化展示钝物体绕流的过程。对于一个粘性流体绕圆柱的二维层流流动问题，采用流函数-涡量法的高阶混合有限差分格式、初始条件和边界条件的处理方法。可视化的部分则采用染色线生成方法。

## 目录

# 1 课题目的

我们常常惊叹于流体的神秘莫测，它和生活本身一样难以预测。作为经典力学的最后一块堡垒，科学家在上个世纪下半叶找到了一种极为有效的攻克它的途径——计算流体力学[1]。随着计算机科学的不断发展，更强大的硬件与更巧妙的算法让这门方法逐渐成熟，能够解决许多工业上的问题。介于这门学科的复杂精深，我们想要实现的功能是单一的。即便如此，这个课题对我们的挑战，起到的提高促进作用仍是巨大的。

在本课题中，我们要实现的目标是：

1. 采用SIMPLE算流函数涡量高阶混合有限差分格式数值模拟二维层流流动。
2. 利用openGL，Qt提供的接口实现计算结果的可视化。
3. 由于计算时间漫长。在程序主要结构中镶嵌一个数独游戏，以供使用者消遣。

# 2 实现方案

## 2.1 流体力学基本原理

流体作为连续介质，在宏观尺度分析时可以忽略物质的分子结构和分子运动。我们用流动的控制方程表达基本物理守恒律的数学形式。

1. 流体质量守恒：

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \tag{1}$$

2. 流体动量方程：

$$\rho \frac{\mathrm{d}u}{\mathrm{d}t} = \frac{\partial(-p + \tau_{xx})}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} \tag{2}$$

3. 流体能量方程：

$$\begin{aligned}\rho \frac{\mathrm{d}E}{\mathrm{d}t} = -\nabla \cdot (p\vec{u}) + [&\frac{\partial(u\tau_{xx})}{\partial x} + \frac{\partial(u\tau_{xy})}{\partial y} + \frac{\partial(u\tau_z z)}{\partial z} + \frac{\partial(v\tau_y x)}{\partial x} + \frac{\partial(v\tau_y y)}{\partial y} + \\ &\frac{\partial(v\tau_y z)}{\partial z} + \frac{\partial(w\tau_z x)}{\partial x} + \frac{\partial(w\tau_z y)}{\partial y} + \frac{\partial(w\tau_z z)}{\partial z} + \nabla \cdot (k\nabla T) + S_E\end{aligned} \tag{3}$$

4. 牛顿流体的剪应力公式:

$$\tau_{xx} = 2\mu \frac{\partial u}{\partial x} + \lambda \nabla \cdot \vec{u}, \tau_{xy} = \tau_{yx} = \mu(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}) \tag{4}$$

$$\cdots\cdots$$

由以上方程1,2,3以及4可以推导出牛顿流体的Nacier-Stokes方程。对于建立有限体积法，其最有用的形式是：

[1]即computational fluid dynamics,CFD

5. N-s方程:

$$\rho\frac{\mathrm{d}u}{\mathrm{d}t} = -\frac{\partial p}{\partial x} + \nabla\cdot(\mu\nabla u) + S_{Mx} \tag{5}$$

$$\rho\frac{\mathrm{d}v}{\mathrm{d}t} = -\frac{\partial p}{\partial y} + \nabla\cdot(\mu\nabla v) + S_{My} \tag{6}$$

$$\rho\frac{\mathrm{d}w}{\mathrm{d}t} = -\frac{\partial p}{\partial z} + \nabla\cdot(\mu\nabla w) + S_{Mz} \tag{7}$$

6. 对于二维不可压缩流体的流动问题，可以用流函数方程和涡量方程来描述 将流函数定义为：

$$u = \frac{\partial\psi}{\partial y}, v = -\frac{\partial\psi}{\partial x}$$

将涡量定义为流体质点旋转角速度的两倍，二维流动的涡量只有一个分量

$$\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

由二维不可压缩流体运动方程可得对流扩散方程

$$\frac{\partial\zeta}{\partial t} + u\frac{\partial\zeta}{\partial x} + v\frac{\partial\eta}{\partial y} = \alpha(\frac{\partial^2\eta}{\partial x^2} + \frac{\partial^2\eta}{\partial y^2}) \tag{8}$$

式中，$\alpha = \frac{1}{R_e}$为粘性扩散系数。将流函数表示的速度分量代入涡量表达式，得到流函数的泊松方程。

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = -\eta \tag{9}$$

## 2.2 数值模拟二维不可压缩粘性流体绕圆柱的流动

以下内容摘自[1],是我们主要参考书目。

存在不连续的跃变,伴随阻力系数下降出现双稳态非对称流态,升力与阻力有相同频率,出现升力系数不为零的情况。双稳态,即在同一雷诺数下有两个斯特鲁哈数随机地出现,并都是稳定的。

## 7.2　流函数-涡量法解圆柱绕流问题的差分格式

### 一、不可压缩粘性流体平面流动的流函数方程和涡量方程

第 2 章已给出了不可压缩流体平面流动流函数和涡量的定义。第 6 章给出了正交曲线坐标($\xi$ $\eta$)下的涡量动力学方程(6.1.15)和流函数方程(6.1.16)。在圆柱绕流问题中,一般把圆柱半径作为特征长度。注意到雷诺数是以圆柱直径作为特征长度,因此无量纲方程的雷诺数项要相应地乘上因子 2。这里的控制方程为

$$\frac{\partial\zeta}{\partial t}+\frac{1}{h_\xi h_\eta}\frac{\partial\psi}{\partial\eta}\frac{2}{Re}\frac{\partial}{\partial\xi}\frac{h_\eta}{h_\xi}\frac{\partial\zeta}{\partial\xi}-\frac{1}{h_\xi h_\eta}\frac{\partial\psi}{\partial\xi}+\frac{2}{Re}\frac{\partial}{\partial\eta}\frac{h_\xi}{h_\eta}\frac{\partial\zeta}{\partial\eta}$$

$$=\frac{1}{h_\xi h_\eta}\frac{2}{Re}\left[\frac{h_\eta}{h_\xi}\frac{\partial^2\zeta}{\partial\xi^2}+\frac{h_\xi}{h_\eta}\frac{\partial^2\zeta}{\partial\eta^2}\right] \tag{7.2.1}$$

$$\frac{1}{h_\xi h_\eta}\left[\frac{\partial}{\partial\xi}\frac{h_\eta}{h_\xi}\frac{\partial\psi}{\partial\xi}+\frac{\partial}{\partial\eta}\frac{h_\xi}{h_\eta}\frac{\partial\psi}{\partial\eta}\right]=-\zeta \tag{7.2.2}$$

方程(7.2.1)中,雷诺数 $Re=UD/\nu$($D$ 是圆柱直径,$\nu$是运动粘度,$U$ 是均匀来流速度),$h_\xi$、$h_\eta$是正交曲线坐标下的拉梅系数。

在物理域内,沿 $\xi=$ 常数和 $\eta=$ 常数的曲线的切向速度分量

$$V_\xi=\frac{1}{h_\eta}\frac{\partial\psi}{\partial\eta}\qquad V_\eta=-\frac{1}{h_\xi}\frac{\partial\psi}{\partial\xi} \tag{7.2.3}$$

涡量动力学方程可简写为

$$\frac{\partial\zeta}{\partial t}+U_\xi\frac{\partial\zeta}{\partial\xi}+U_\eta\frac{\partial\zeta}{\partial\eta}=\frac{2}{Re}\frac{1}{h_\xi^2}\frac{\partial^2\zeta}{\partial\xi^2}+\frac{1}{h_\eta^2}\frac{\partial^2\zeta}{\partial\eta^2} \tag{7.2.4}$$

式中,一阶导数项的系数记为

$$\begin{aligned}U_\xi&=\frac{V_\xi}{h_\xi}-\frac{1}{h_\xi h_\eta}\frac{2}{Re}\frac{\partial}{\partial\xi}\frac{h_\eta}{h_\xi}\\ U_\eta&=\frac{V_\eta}{h_\eta}-\frac{1}{h_\xi h_\eta}\frac{2}{Re}\frac{\partial}{\partial\eta}\frac{h_\xi}{h_\eta}\end{aligned} \tag{7.2.5}$$

流函数方程(7.2.2)展开后可写为

$$g_1\frac{\partial^2\psi}{\partial\xi^2}+g_2\frac{\partial^2\psi}{\partial\eta^2}+g_3\frac{\partial\psi}{\partial\xi}+g_4\frac{\partial\psi}{\partial\eta}=-\zeta \tag{7.2.6}$$

式中,二阶和一阶导数项的系数为

$$g_1=\frac{1}{h_\xi^2},\quad g_2=\frac{1}{h_\eta^2},\quad g_3=\frac{1}{h_\xi h_\eta}\frac{\partial}{\partial\xi}\frac{h_\eta}{h_\xi},\quad g_4=\frac{1}{h_\xi h_\eta}\frac{\partial}{\partial\eta}\frac{h_\xi}{h_\eta}\tag{7.2.7}$$

## 二、涡量方程的混合差分格式

在这一章的计算中,采用6.2节的保角映射生成法生成贴体坐标网格。实际上,这样生成的网格坐标线也是不可压缩、无粘、均匀来流绕圆柱体无环量流动的流线和等势线。圆表面是$\psi=0$的流线。

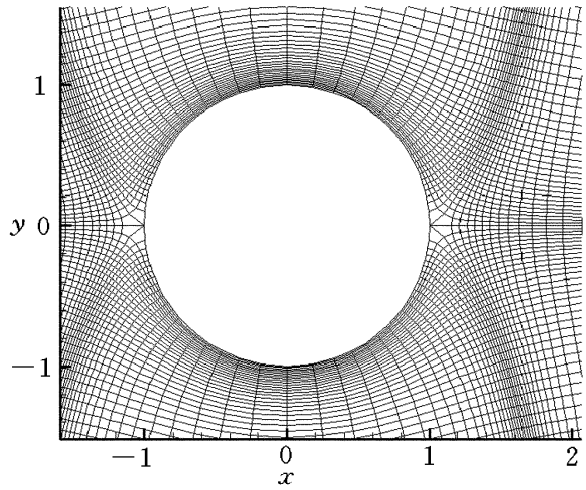


图7.3 圆柱绕流计算的网格(部分)

物理平面上,原点设在圆心的网格呈上、下对称和前、后对称的形式。但是,在上游方向只取10倍半径长度的网格,下游方向取到60倍半径长度的距离(见图7.3)。

在大雷诺数情况,涡量方程(7.2.4)左边的对流项不能用二阶差分格式。因为它的误差可能会大于粘性项本身。经验表明,采用三阶迎风格式和四阶中心差分格式的混合格式比较好。设$L_3$、$L_4$分别代表三阶格式和四阶格式的算子,则有混合格式

$$L=\lambda L_3+(1-\lambda)L_4\tag{7.2.8}$$

式中,加权因子$\lambda$定义为

$$\lambda=1-\frac{1}{\exp(U^*)}\tag{7.2.9}$$

式中,$U^*$分别用$U_\xi$、$U_\eta$代入,相应的$\lambda$为$\lambda_\xi$和$\lambda_\eta$。经验表明,当雷诺数超过1 000时,中心差分格式引起的伪扩散效应使计算不易收敛,此时可简单令$\lambda=1$,即只用迎风格式计算对流项。式(7.2.4)左边对流项的三阶迎风格式为

$$U_\xi\frac{\partial\zeta}{\partial\xi}=\frac{U_{\xi i,j}+|U_{\xi i,j}|}{2}\frac{2\zeta_{i+1,j}+3\zeta_{i,j}-6\zeta_{i-1,j}+\zeta_{i-2,j}}{6\Delta\xi}+\frac{U_{\xi i,j}-|U_{\xi i,j}|}{2}\frac{-\zeta_{i+2,j}+6\zeta_{i+1,j}-3\zeta_{i,j}-2\zeta_{i-1,j}}{6\Delta\xi}\tag{7.2.10}$$

$$U_\eta\frac{\partial\zeta}{\partial\eta}=\frac{U_{\eta i,j}+|U_{\eta i,j}|}{2}\frac{2\zeta_{i+1}+3\zeta_{j}-6\zeta_{j-1}+\zeta_{j-2}}{6\Delta\eta}+\frac{U_{\eta i,j}-|U_{\eta i,j}|}{2}\frac{-\zeta_{j+2}+6\zeta_{j+1}-3\zeta_{j}-2\zeta_{j-1}}{6\Delta\eta}\tag{7.2.11}$$

式中,$\xi$、$\eta$方向的标号分别为$i$、$j$。四阶中心差分格式为

$$\frac{\partial\zeta}{\partial\xi}=\frac{-\zeta_{i+2,j}+8\zeta_{i+1,j}-8\zeta_{i-1,j}+\zeta_{i-2,j}}{12\Delta\xi} \tag{7.2.12}$$

$$\frac{\partial\zeta}{\partial\eta}=\frac{-\zeta_{i,j+2}+8\zeta_{i,j+1}-8\zeta_{i,j-1}+\zeta_{i,j-2}}{12\Delta\eta} \tag{7.2.13}$$

涡量方程(7.2.4)右边的二阶导数可用二阶中心差分格式

$$\frac{\partial^2\zeta}{\partial\xi^2}=\frac{\zeta_{i+1,j}-2\zeta_{i,j}+\zeta_{i-1,j}}{\Delta\xi^2} \tag{7.2.14}$$

$$\frac{\partial^2\zeta}{\partial\eta^2}=\frac{\zeta_{i,j+1}-2\zeta_{i,j}+\zeta_{i,j-1}}{\Delta\eta^2} \tag{7.2.15}$$

涡量方程(7.2.2)左边的时间导数用如下形式的差分格式

$$\frac{\partial\zeta}{\partial t}=\frac{\zeta^{n+1}-\zeta^{n}}{\Delta t}$$

当时间导数在 $n+1/2$ 时间层上取值，时间步长为 $\Delta t/2$ 时，上式相当于二阶精度的中心差分格式，即

$$\left(\frac{\partial\zeta}{\partial t}\right)^{n+\frac{1}{2}}_{i,j}=\frac{\zeta^{n+1}_{i,j}-\zeta^{n}_{i,j}}{\Delta t} \tag{7.2.16}$$

相应地，空间方向的差分也必须取在 $n+1/2$ 时间层上，这里用 $n$ 和 $n+1$ 两个时间层的空间差分格式的平均作为 $n+1/2$ 时间层上的差分格式，即令

$$\zeta^{n+\frac{1}{2}}_{i,j}=\frac{\zeta^{n+1}_{i,j}+\zeta^{n}_{i,j}}{2} \tag{7.2.17}$$

式(7.2.3)、式(7.2.5)、式(7.2.7)中各偏导数均用二阶中心差格式计算。

采用以上差分格式后，涡量方程的差分方程形式表达如下：

$$\begin{aligned}
&\frac{2(\zeta^{n+1}-\zeta^{n})}{\Delta t}+\lambda_{\xi}U_{\xi}\frac{\partial\zeta^{n}}{\partial\xi}+\lambda_{\xi}U_{\xi}\frac{\partial\zeta^{n+1}}{\partial\xi}+\lambda_{\eta}U_{\eta}\frac{\partial\zeta^{n}}{\partial\eta}\\
&+\lambda_{\eta}U_{\eta}\frac{\partial\zeta^{n+1}}{\partial\eta}{}_{3}+(1-\lambda_{\xi})U_{\xi}\frac{\partial\zeta^{n}}{\partial\xi}+(1-\lambda_{\xi})U_{\xi}\frac{\partial\zeta^{n+1}}{\partial\xi}\\
&+(1-\lambda_{\eta})U_{\eta}\frac{\partial\zeta^{n}}{\partial\eta}+(1-\lambda_{\eta})U_{\eta}\frac{\partial\zeta^{n+1}}{\partial\eta}{}_{4}\\
&=\frac{2}{Re}\left(\frac{1}{h_{\xi}^2}\frac{\partial^2\zeta^{n}}{\partial\xi^2}+\frac{1}{h_{\xi}^2}\frac{\partial^2\zeta^{n+1}}{\partial\xi^2}+\frac{1}{h_{\eta}^2}\frac{\partial^2\zeta^{n}}{\partial\eta^2}+\frac{1}{h_{\eta}^2}\frac{\partial^2\zeta^{n+1}}{\partial\eta^2}\right)_{2}
\end{aligned} \tag{7.2.18}$$

式中，下标 2、3、4 表示括弧内的偏导数项分别用二、三、四阶差分格式离散，上标表示取值的时间层位置。差分格式整理为下列简写形式

$$\begin{aligned}
c_1\zeta^{n+1}_{i,j}=&\,c_2\zeta^{n}_{i,j}+c_3(\zeta^{n+1}_{i,j+2}+\zeta^{n}_{i,j+2})+c_4(\zeta^{n+1}_{i,j+1}+\zeta^{n}_{i,j+1})\\
&+c_5(\zeta^{n+1}_{i,j-1}+\zeta^{n}_{i,j-1})+c_6(\zeta^{n+1}_{i,j-2}+\zeta^{n}_{i,j-2})+c_7(\zeta^{n+1}_{i+2,j}+\zeta^{n}_{i+2,j})\\
&+c_8(\zeta^{n+1}_{i+1,j}+\zeta^{n}_{i+1,j})+c_9(\zeta^{n+1}_{i-1,j}+\zeta^{n}_{i-1,j})+c_{10}(\zeta^{n+1}_{i-2,j}+\zeta^{n}_{i-2,j})
\end{aligned} \tag{7.2.19}$$

式中,系数 $c_1$～$c_{10}$ 分别为

$$
\begin{aligned}
c_1 &= -\frac{2}{\Delta t} - \frac{\lambda_\xi(\psi)_{\xi}'}{2\Delta\xi} - \frac{\lambda_\eta(\psi)_{\eta}'}{2\Delta\eta} - \frac{4}{h_\xi^2\Delta\xi^2 Re} - \frac{4}{h_\eta^2\Delta\eta^2 Re}\\
c_2 &= -\frac{2}{\Delta t} + \frac{\lambda_\xi(\psi)_{\xi}'}{2\Delta\xi} + \frac{\lambda_\eta(\psi)_{\eta}'}{2\Delta\eta} + \frac{4}{h_\xi^2\Delta\xi^2 Re} + \frac{4}{h_\eta^2\Delta\eta^2 Re}\\
c_3 &= -\frac{U_\eta - \lambda_\eta(\psi)_{\eta}'}{12\Delta\eta}\\
c_4 &= \frac{2U_\eta - \lambda_\eta(\psi)_{\eta}'}{3\Delta\eta} - \frac{2}{h_\eta^2\Delta\eta^2 Re}\\
c_5 &= -\frac{2U_\eta + \lambda_\eta(\psi)_{\eta}'}{3\Delta\eta} - \frac{2}{h_\eta^2\Delta\eta^2 Re}\\
c_6 &= \frac{U_\eta + \lambda_\eta(\psi)_{\eta}'}{12\Delta\eta}\\
c_7 &= -\frac{U_\xi - \lambda_\xi(\psi)_{\xi}'}{12\Delta\xi}\\
c_8 &= \frac{2U_\xi - \lambda_\xi(\psi)_{\xi}'}{3\Delta\xi} - \frac{2}{h_\xi^2\Delta\xi^2 Re}\\
c_9 &= -\frac{2U_\xi + \lambda_\xi(\psi)_{\xi}'}{3\Delta\xi} - \frac{2}{h_\xi^2\Delta\xi^2 Re}\\
c_{10} &= \frac{U_\xi + \lambda_\xi(\psi)_{\xi}'}{12\Delta\xi}
\end{aligned}
\tag{7.2.20}
$$

从圆柱表面向外的第二层网格点不能用加权平均的三阶迎风格式和四阶中心差分格式,因为缺少差分格式中的域外(进入圆柱内部)邻点。这时直接用差分格式(7.2.10)和式(7.2.11)中偏向流体侧的一个三阶格式,相应修改式(7.2.20)中差分格式各系数。比较修改后的系数和式(7.2.20)中的系数可以发现,这个变化相当于在系数 $c_1$～$c_{10}$ 的表达式中,对应上半圆用 $-U_\eta$ 代替 $\lambda_\eta(\psi)_{\eta}'$,对应下半圆用 $U_\eta$ 代替 $\lambda_\eta(\psi)_{\eta}'$,对应左驻点用 $U_\xi$ 代替 $\lambda_\xi(\psi)_{\xi}'$,对应右驻点用 $-U_\xi$ 代替 $\lambda_\xi(\psi)_{\xi}'$。

## 三、流函数方程的中心差分格式

流函数方程(7.2.6)中的一、二阶导数项都用二阶精度的中心差分格式离散,即有

$$
g_1\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{\Delta\xi^2} + g_2\frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{\Delta\eta^2}
+ g_3\frac{\psi_{i+1,j} - \psi_{i-1,j}}{2\Delta\xi} + g_4\frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\Delta\eta} = -\zeta_{i,j}
$$

上式整理为下列简写形式

$$
\psi_{i,j} = (b_1\psi_{i+1,j} + b_2\psi_{i-1,j} + b_3\psi_{i,j+1} + b_4\psi_{i,j-1} + \zeta_{i,j})/b_0 \tag{7.2.21}
$$

式中
$$b_0 = \frac{2g_1}{\Delta\xi^2} + \frac{2g_2}{\Delta\eta^2},\quad b_1 = \frac{g_1}{\Delta\xi^2} + \frac{g_3}{2\Delta\xi}$$
$$b_2 = \frac{g_1}{\Delta\xi^2} - \frac{g_3}{2\Delta\xi}\quad b_3 = \frac{g_2}{\Delta\eta^2} + \frac{g_4}{2\Delta\eta}\quad b_4 = \frac{g_2}{\Delta\eta^2} - \frac{g_4}{2\Delta\eta} \tag{7.2.22}$$

## 四、松弛迭代法

流动控制方程的离散产生一组线性代数方程，代数方程组的复杂性和规模大小与网格点数目、离散方法及问题的维数有关。任何有效的方法都可以用来解代数方程组，但受到计算机的能力限制。有两类解法，即直接法和间接法(迭代法)。简单的问题用直接法，求解N 个未知量的N 个代数方程，其运算量是$N^3$ 的量级，需要在内存同时存储方程组的$N^2$ 个系数。

迭代法是一种从已知近似解计算新的近似解的规则。如式(7.2.21)可写为
$$\psi_j^{(m+1)} = (b_1\psi_{j+1}^{(m)} + b_2\psi_{j}^{(m)} + b_3\psi_{j+1}^{(m)} + b_4\psi_{j-1}^{(m)} + \zeta_j)/b_0 \tag{7.2.23}$$
式中，上标m 表示迭代的次数。这就是一种简单迭代公式。此式表示了椭圆型问题中解的分布的均匀化倾向。迭代法逐步修正，使近似解收敛到满足方程的极限解。

简单迭代法每次迭代需要保留旧值给邻点的迭代用。如果每次尽可能的用已经更新了的值，就不必存储原有的旧值。由于提前使用更新值可使收敛速度成倍增加。

不同规则产生不同的迭代法，以松弛法最为常用。松弛迭代法将上面的差分格式写成
$$\psi_j^{(m+1)} = \psi_j + \omega(b_1\psi_{j+1} + b_2\psi_{j} + b_3\psi_{j+1} + b_4\psi_{j-1} + \zeta_j - b_0\psi_j)/b_0$$
方程右边用原有旧值或尽可能用更新值进行迭代。此式一般写成
$$\psi_j^{(m+1)} = (1-\omega)\psi_j + \omega(b_1\psi_{j+1} + b_2\psi_{j} + b_3\psi_{j+1} + b_4\psi_{j-1} + \zeta_j)/b_0 \tag{7.2.24}$$

上式表示，每次迭代用一增量修正旧值。式中 ω称为松弛因子，用于控制修正的幅度。可以证明，$\omega < 2$ 的迭代都是收敛的。$\omega < 1$ 称为低松弛，为得到较快的收敛速度，最优值$\omega^*$ 在$1 < \omega < 2$ 之间的超松弛范围。对于大雷诺数或计算压强的方程的数值计算，较大的修正幅度导致解迅速发散，这时需要用较小的超松弛因子或者用低松弛因子。

对于简单迭代法，迭代中的扫描顺序与收敛无关。松弛法则不然。扫描顺序不同导致不同的迭代方案。对于不同的扫描顺序，最优ω值及最终的收敛速度是相同的。但首先用 $\omega = 1$ 有利于抑制迭代误差，以后再用最优值 $\omega^*$。

一般情况下，关于函数 u 的线性代数方程组可写为
$$\sum_j^N a_{ij}u_j = b_j,\quad i,j = 1,2,\cdots,N \tag{7.2.25}$$

规定一种迭代次序后,简单迭代为

$$u_i^{(m+1)} = \frac{1}{a_{ii}} b_i - \sum_{j \neq i} a_{ij} u_j^{(m)} \tag{7.2.26}$$

松弛法为

$$u_i^{(m+1)} = (1-\omega) u_i^{(m)} + \frac{\omega}{a_{ii}} b_i - \sum_{j<i} a_{ij} u_j^{(m+1)} - \sum_{j>i} a_{ij} u_j^{(m)} \tag{7.2.27}$$

对于绕流圆柱体的计算,涡量方程可用简单迭代法求解,流函数方程则用松弛迭代法求解。

迭代法的基本思想是多次反复应用相对简单的算法直到解收敛。典型的一次迭代有N 个运算,但无法事先确定迭代次数。迭代次数有时非常大,而且不能保证迭代一定收敛,除非方程组满足一定的准则。迭代法的主要优点是只有非零系数需要内存。有限差分格式规律性强,易编程。对于通常非常大的方程组,往往首选迭代法。用迭代法,存储、运算都比较节省。但对于某些问题,迭代法可能发散或收敛速度慢,这时仍需用直接法(冯康,1978)。

## 7.3 初始条件和边界条件

### 一、初始条件

求解流函数方程和非定常的涡量动力学方程时,将无粘、不可压缩流体平面流动的流函数和无旋条件作为初始条件。将定常来流绕圆柱体的稳态流动状态作为在初始条件下非定常发展过程的渐近结果,初始条件是第一步迭代的初值。

沿x 方向的均匀来流速度为U,圆半径为a。无量纲来流速度值与半径值均取为1,得到无粘、不可压缩流体绕圆柱无环量平面流动的流函数为

$$\psi(x,y) = y\left(1 - \frac{1}{x^2 + y^2}\right) \tag{7.3.1}$$

上式作为粘性流体绕流圆柱体计算的流函数初始条件,涡量初始条件是 $\zeta = 0$。

### 二、边界条件

以外边界为平行于坐标方向的矩形、内边界为圆的域为例,说明流函数和涡量的边界条件的设定。假定边界离圆柱足够远(见图7.4)。

1. 进口边界的流函数和涡量条件

求解域的进口(左边界)为沿x 方向的均匀无旋来流条件,在进口边界的速度 $u = 1, v = 0$。

于是,设进口边界的涡量和流函数条件为

$$\zeta = 0, \quad \psi = y \tag{7.3.2}$$

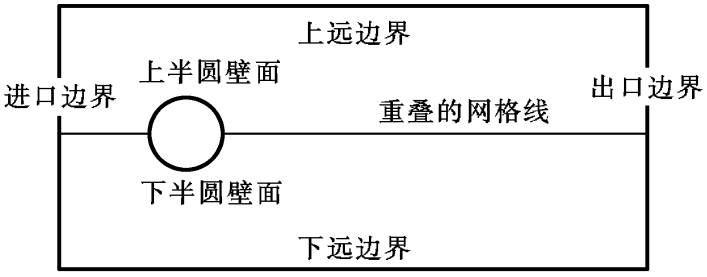


图 7.4　圆柱绕流计算网格的边界

2. 侧边界和出口的涡量条件

简单处理可以令远边界点的 $\zeta=0$, 即在边界点的差分格式中去掉涉及域外点的项。下面讨论比较准确的做法, 给出忽略粘性影响的侧面(上、下)边界和出口边界的涡量条件。在边界点利用无粘涡量方程来实现边界条件。无粘涡量方程为

$$\frac{\partial\zeta}{\partial t}+\frac{V_\xi}{h_\xi}\frac{\partial\zeta}{\partial\xi}+\frac{V_\eta}{h_\eta}\frac{\partial\zeta}{\partial\eta}=0 \tag{7.3.3}$$

在上、下边界, 离散涡量方程的 $\partial\zeta/\partial\xi$ 项用二阶精度迎风格式, $\partial\zeta/\partial\eta$ 项则根据不同情况利用向后差分或向前差分。

(1) 在上远边界, $V_\eta<0$ 时, $\zeta^{n+1}=\zeta$; $V_\eta>0$ 时, $\partial\zeta/\partial\eta$ 用二阶精度向后差分

$$\left.\frac{\partial\zeta}{\partial\eta}\right|_{i,j}=\frac{3\zeta_{i,j}-4\zeta_{i,j-1}+\zeta_{i,j-2}}{2\Delta\eta} \tag{7.3.4}$$

(2) 在下远边界, $V_\eta>0$ 时, $\zeta^{n+1}=\zeta$; $V_\eta<0$ 时, $\partial\zeta/\partial\eta$ 用二阶精度向前差分

$$\left.\frac{\partial\zeta}{\partial\eta}\right|_{i,j}=\frac{-3\zeta_{i,j}+4\zeta_{i,j+1}-\zeta_{i,j+2}}{2\Delta\eta} \tag{7.3.5}$$

(3) 在出口边界, $\partial\zeta/\partial\eta$ 用二阶精度迎风格式, $\partial\zeta/\partial\xi$ 则根据情况利用向后差分。

$V_\xi<0$ 时, $\zeta^{n+1}=\zeta$; $V_\xi>0$ 时, $\partial\zeta/\partial\xi$ 用二阶精度向后差分

$$\left.\frac{\partial\zeta}{\partial\xi}\right|_{i,j}=\frac{3\zeta_{i,j}-4\zeta_{i-1,j}+\zeta_{i-2,j}}{2\Delta\xi} \tag{7.3.6}$$

3. 壁面的流函数和涡量条件

流体力学在物理上给出的壁面条件一般是粘附条件, 对于固定物面即为法向速度和切向速度为零这两个条件。壁面法向速度为零等效于流线不可穿透条件, 即壁面曲线是一条流线(等流函数线)。流函数的参考值可以任意选取, 但这里要与式(7.3.7)规定的边界条件保持一致, 在圆表面令

$$\psi_{i,1}=0 \tag{7.3.7}$$

流函数方程有一个壁面条件就足够了, 壁面切向速度为零的条件将用于等价的壁面涡量条件。壁面涡量是很重要的物理量。内点的总涡量守恒, 然而在无滑移的壁面可以有涡量生成(正涡量或负涡量), 生成的涡量会在流体中扩散和随之对流。下面从壁面流函数方程导出壁面涡量与流函数的关系, 作为涡量的壁

面条件。在圆表面,流函数满足方程(7.2.6)

$$g_1\frac{\partial^2\psi}{\partial\xi^2}+g_2\frac{\partial^2\psi}{\partial\eta^2}+g_3\frac{\partial\psi}{\partial\xi}+g_4\frac{\partial\psi}{\partial\eta}=-\zeta$$

因为壁面曲线是一条$\psi=0$的流线,这里再设其为沿$\xi$方向的坐标线,所以在壁面有

$$\left.\frac{\partial\psi}{\partial\xi}\right|_w=0,\qquad \left.\frac{\partial^2\psi}{\partial\xi^2}\right|_w=0 \tag{7.3.8}$$

其次,壁面上速度为零的条件给出

$$u=\frac{\partial\psi}{\partial y}=-\frac{x_\eta}{J}\frac{\partial\psi}{\partial\xi}+\frac{x_\xi}{J}\frac{\partial\psi}{\partial\eta}=0$$

$$v=-\frac{\partial\psi}{\partial x}=-\frac{y_\eta}{J}\frac{\partial\psi}{\partial\xi}+\frac{y_\xi}{J}\frac{\partial\psi}{\partial\eta}=0$$

由此可知,在壁面上

$$\left.\frac{\partial\psi}{\partial\eta}\right|_w=0 \tag{7.3.9}$$

将式(7.3.8)、式(7.3.9)代入壁面流函数方程(7.2.6),得到壁面涡量的表达式:

$$\zeta_{w,j}=-g_2\left.\frac{\partial^2\psi}{\partial\eta^2}\right|_w \tag{7.3.10}$$

在壁面附近一点,流函数的泰勒级数展开式为

$$\psi_{w+1}=\psi_w+\left.\frac{\partial\psi}{\partial\eta}\right|_w\Delta\eta+\left.\frac{\partial^2\psi}{\partial\eta^2}\right|_w\frac{\Delta\eta^2}{2}+\left.\frac{\partial^3\psi}{\partial\eta^3}\right|_w\frac{\Delta\eta^3}{6}+\cdots$$

式中,下标w表示壁面上的网格点,下标w+1表示沿壁面外法向、最靠近壁面的网格点,这点到壁面间的距离为$\Delta\eta$,$\Delta\eta$的正负值分别对应上、下两个半圆。

将式(7.3.7)～式(7.3.10)代入泰勒级数展开式,并略去三阶以上的项,得到一阶近似的壁面涡量表达式为

$$\zeta_{w,j}=-\frac{1}{h_\eta^2}\frac{2(\psi_{w+1}-\psi_w)}{\Delta\eta^2} \tag{7.3.11}$$

用迭代法解离散流函数方程(7.2.21),得到流函数分布后由式(7.3.11)求出壁面涡量,用此式作为离散涡量方程(7.2.19)的壁面边界条件。

壁面流函数$\psi_w$可以不等于零。如果壁面是沿$\eta$方向的坐标线,可类似导出相应的壁面涡量的计算公式。

4. 侧边界的流函数条件

充分远两侧面(上、下边界)的流函数取为无粘流流函数条件——式(7.3.1)。或按充分远近似,简单取为均匀流条件

$$\psi=y \tag{7.3.12}$$

5. 出口的流函数条件

对于有尾流存在的绕流问题,出口边界的流函数不能完全给定。数值试验表

明,不稳定因素会从流出边界向上游传播,造成迭代发散。在流出边界上,应尽量放宽约束,使流动能自由发展。例如,仅仅限制 y 方向速度为零,即 $\partial\psi/\partial\xi = 0$;或进一步放宽为仅仅限制 y 方向速度沿流向的导数为零,即 $\partial^2\psi/\partial\xi^2 = 0$。后一个约束条件用中心差分可写为

$$\psi_{N,j} = 2\psi_{N-1,j} - \psi_{N-2,j} \tag{7.3.13}$$

式中,N 表示出口边界的节点。N - 1、N - 2 表示边界上游域内的两个邻点。

## 三、圆柱振荡尾流的启动

圆柱绕流出现振荡尾流与流动稳定性有关。但求解域的几何结构和边界条件是对称的,如果没有扰动引起流动失稳,收敛解也应该是对称的。实际流动自然会有天然的扰动源引起初始对称的流动失稳。精确的数值计算缺少这种天然的扰动源,然而经验表明,只要计算足够长的时间,它自身数值误差的积累足以诱发非对称的振荡流(Nair, 1996)。

为尽快达到稳定的振荡尾流状态,在对称边界条件下施加人为干扰仍是可取的。必须注意,一旦流场的对称性已被破坏,就要撤去人为干扰,在振荡尾流自动维持的稳定状态下才能输出有用的计算结果。

加扰动可有多种方式,例如让圆柱体先后按顺时针、反时针方向旋转,引起表面速度振荡。下面给出一个例子,是在进口边界的来流叠加小的横向扰动速度分量。

首先,按式(7.3.2)、式(7.3.12)和式(7.3.13)中的流函数边界条件进行计算,得到圆柱后方附着一对稳定的对称尾涡的结果。当无量纲时间 T 大于某个值,如 $T = T_0$ 时,开始施加人工扰动。

设小扰动幅度 ε是时间的函数,按来流速度 U = 1 半径 a = 1,给出 ε

$$\varepsilon(t) = 0.05\text{th}(0.1t)\sin(0.25\pi) \tag{7.3.14}$$

$\varepsilon(t)$ 的时间导数为扰动速度。扰动变化的周期取为接近尾流振荡周期的 $\Delta T = 8$。于是有

$$\dot{\varepsilon}(t) = 0.012\,5\pi\text{h}(0.1t)\cos(0.25\pi) + \frac{0.005}{\text{ch}^2(0.1t)}\sin(0.25\pi) \tag{7.3.15}$$

式中,$t = T - T_0$。将扰动流函数 $\Delta\psi = \dot{\varepsilon}x$ 加到式(7.3.2)和式(7.3.12)中流函数边界条件的右边,这相当于在入口边界和上、下边界加上 y 方向的扰动速度 $\dot{\varepsilon}$。扰动幅度将随时间推进从 0 增长到 0.05,并发生振荡。

当计算结果显示流场已失去对称性即可去掉扰动速度,并恢复规定的边界条件。加扰动的时间约需几个尾流振荡周期。撤去扰动后继续计算,观察到圆柱尾流呈现规则的周期性,即认为计算达到稳态,可以输出数值计算的结果。

# 3 程序结构说明（数据结构，主要算法，类，函数

## 3.1 后端

### 3.1.1 使用的库：UMFPACK

UMFPACK[3]是一系列解非对称稀疏矩阵$AX = B$的库，使用了多波前算法和直接的LU分解算法。它使用ANSI/ISO C写成，提供了MATLAB前端。UMFPACK依赖于BLAS-Level 3。UMFPACK支持Windows和许多Unix平台。 解大规模线性方程组是SIMPLE算法的重点，其运算效率直接影响到整个程序的效率。后文中会提到，我们尝试了简单迭代法和松弛迭代法，但是迭代发散，所以不得不寻求别的高效解法。首先，我们的方程组中系数为零的项占绝大多数，在流函数方程组中每个方程仅有5个非零项，而在涡量方程组中不超过9个，非零项个数和方程个数n线性相关，为稀疏矩阵。对于一般矩阵，存储空间和计算时间复杂度均为$O(n^2)$,而如能利用稀疏矩阵的特性， 则可降至$O(n)$，这对于快速解出大型方程组来说是必要的。UMFPACK能充分利用矩阵的稀疏性，本身算法先进，稳定，可扩展性好，而且UMFPACK可以使用用户提供的BLAS库而非静态的算法，因此用户可以提供高效的BLAS实现（如Intel MKL）来加快运算速度。

- solver.h

  将UMFPACK提供的C接口封装为C++类Solver，通过给定矩阵元$a_{ij}$的值和$B$便可解得X。 提供接口： 构造函数Solver(int num, int nonzero, double* right) num为矩阵阶数，nonzero为非零项个数，right为一维数组B。

- bool input(int i, int j, double x)

  设置aij=x，如果成功返回true，如果设置非零项的总数超过了nonzero则返回false。

- void solve()

  求解方程

- double* getSolution()

  返回一维数组X。注意Solver类析构时会释放X的内存空间，所以访问时请保证Solver类未被析构

### 3.1.2 主要的类

cylinderDataVariant cylinderDataVariant为DataVariant的一个子类，用于前端从后端获取数据。

cylinderNode cylinderNode用于记录每个计算节点的流函数、涡量和速度

cylinderSpotStain cylinderSpotStain是染色点类，用于染色点法绘制流线图。

cylinderProject cylinderProject是圆柱体绕流模型，提供接口如下：

构造函数 cylinderProject(int l = -100, int r = 400, int u = 100, int d = -100, double dens = 0.1, double dxi = 0.1, double deta = 0.1, double dt = 0.1, double rey = 40)。 构造并初始化系统。其中l为计算坐标系的左边界，r为右

边界，u为上边界，d为下边界。dens为绘制的流线密度，取值在0-1间，值越大流线越密集。dxi为计算坐标系xi方向的坐标间隔（相邻两个计算节点的横坐标差），deta为eta方向坐标间隔，dt为时间步长，rey为雷诺数。

构造函数cylinderProject(const char* location) 读取存盘文件并继续计算。location为存盘文件路径。存盘文件可由方法dumptofile生成。

DataVariant * getData(Project::DataType type, ...) 根据传入的数据类型传出交换数据。type可以为Project::TimeType, Project::PsiType, Project::SpotType, Project::NumberType之一，分别返回包含系统时间、流函数、染色点位置、染色点发生源数目。其中流函数需要用可变参数再提供节点的xi和eta坐标，染色点位置需要再提供染色点发生源的标号（从0到染色点发生源数目-1）。

void setDensity(double dens)void run(); 进行一步计算，系统时间增加一个时间步长

void spotstainrun(); 染色点运动一个时间步长

bool dumptofile(const char* location); 将当前计算项目存盘，location为存盘文件路径。如果成功返回true，失败返回false。

DataVariant DataVariant是前端与后端进行数据交换的类型。有四种类型：Project::TimeType, Project::PsiType, Project::SpotType, Project::NumberType。提供接口如下：

- getX()
  仅适用于Project::PsiType, Project::SpotType 获得目标的x坐标
- getY()
  仅适用于Project::PsiType, Project::SpotType 获得目标的y坐标
- getZ()
  仅适用于Project::PsiType, Project::SpotType 获得目标的z坐标
- getPsi()
  仅适用于Project::PsiType 获得节点的流函数值
- getTime()
  仅适用于Project::TimeType 获得系统的时间
- getNumber()
  仅适用于Project::NumberType 获得染色点发生源数目

### 3.1.3 代码示例

- 以下为简单迭代法算法的代码。我们之前采用这个算法，发现数值很快超过浮点数上限，算法不收敛。经过研究思考最后改用现在使用的算法。本部分代码部分采用了Intel提供的线性代数函数库[2]

```
1 #pragma omp parallel for private(j, i)
2
```

```
3       for (j = downboundary + 2; j < upboundary - 1; j++) {
4           for (i = leftboundary + 2; i < rightboundary - 1; i++) {
5               coordination->access(i, j).zetat = coordination->access(i, j).zeta;
6           }
7       }
8
9
10      converge = 30;
11
12      while (converge) {
13          //TODO:converge
14          #pragma omp parallel for private(j, i, node)
15          for (j = downboundary + 2; j < upboundary - 1; j++) {
16              for (i = leftboundary + 2; i < rightboundary - 1; i++) {
17                  if ((j <= 2) && (j >= -2) && (i >= leftterminal)
18                      && (i <= rightterminal)) {
19                      continue;
20                  }
21
22                  node = &coordination->access(i, j);
23                  node->newzetat =
24                      (node->c2 * node->zeta +
25                       node->c3 * (coordination->access(i, j + 2).zetat +
26                                   coordination->access(i,
27                                                        j + 2).zeta) +
28                       node->c4 * (coordination->access(i, j + 1).zetat +
29                                   coordination->access(i,
30                                                        j + 1).zeta) +
31                       node->c5 * (coordination->access(i, j - 1).zetat +
32                                   coordination->access(i,
33                                                        j - 1).
```

```
                                                                zeta↩
                                                                ) ↩
                                                                +
34                      node->c6 * (coordination->access(i, j ↩
                           - 2).zetat +
35                                  coordination->access(i,
36                                                       j - ↩
                                                            2)↩
                                                            .↩
                                                            zeta↩
                                                            ) ↩
                                                            +
37                      node->c7 * (coordination->access(i + ↩
                           2, j).zetat +
38                                  coordination->access(i + ↩
                                     2,
39                                                       j).↩
                                                            zeta↩
                                                            ) ↩
                                                            +
40                      node->c8 * (coordination->access(i + ↩
                           1, j).zetat +
41                                  coordination->access(i + ↩
                                     1,
42                                                       j).↩
                                                            zeta↩
                                                            ) ↩
                                                            +
43                      node->c9 * (coordination->access(i - ↩
                           1, j).zetat +
44                                  coordination->access(i - ↩
                                     1,
45                                                       j).↩
                                                            zeta↩
                                                            ) ↩
                                                            +
46                      node->c10 * (coordination->access(i - ↩
                           2, j).zetat +
47                                   coordination->access(i - ↩
                                      2, j).zeta)) / node->↩
                                      c1;
48              }
49          }
50
51          j = 2;
52          #pragma omp parallel for private(i, node)
53
54          for (i = leftterminal; i <= rightterminal; i++) {
55              node = &coordination->access(i, j);
56              node->newzetat =
57                  (node->c2 * node->zeta +
```

```
58                  node->c3 * (coordination->access(i, j + 2)↩
                       .zetat +
59                              coordination->access(i,
60                                                   j + 2).↩
                                                        zeta) ↩
                                                        +
61                  node->c4 * (coordination->access(i, j + 1)↩
                       .zetat +
62                              coordination->access(i,
63                                                   j + 1).↩
                                                        zeta) ↩
                                                        +
64                  node->c5 * (coordination->access(i, j - 1)↩
                       .zetat +
65                              coordination->access(i,
66                                                   j - 1).↩
                                                        zeta) ↩
                                                        +
67                  node->c6 * (cylinderBoundary->access(i, 1)↩
                       .zetat +
68                              cylinderBoundary->access(i,
69                                      1).zeta) +
70                  node->c7 * (coordination->access(i + 2, j)↩
                       .zetat +
71                              coordination->access(i + 2,
72                                                   j).zeta) ↩
                                                        +
73                  node->c8 * (coordination->access(i + 1, j)↩
                       .zetat +
74                              coordination->access(i + 1,
75                                                   j).zeta) ↩
                                                        +
76                  node->c9 * (coordination->access(i - 1, j)↩
                       .zetat +
77                              coordination->access(i - 1,
78                                                   j).zeta) ↩
                                                        +
79                  node->c10 * (coordination->access(i - 2, j↩
                       ).zetat +
80                               coordination->access(i - 2, j↩
                                    ).zeta)) / node->c1;
81          }
82
83          j = -2;
84          #pragma omp parallel for private(i, node)
85
86          for (i = leftterminal; i <= rightterminal; i++) {
87              node = &coordination->access(i, j);
88              node->newzetat =
89                  (node->c2 * node->zeta +
90                   node->c3 * (cylinderBoundary->access(i, 0)↩
```

```
                    .zetat +
91                              cylinderBoundary->access(i,
92                                      0).zeta) +
93                  node->c4 * (coordination->access(i, j + 1)↩
                    .zetat +
94                              coordination->access(i,
95                                                   j + 1).↩
                                                      zeta) ↩
                                                      +
96                  node->c5 * (coordination->access(i, j - 1)↩
                    .zetat +
97                              coordination->access(i,
98                                                   j - 1).↩
                                                      zeta) ↩
                                                      +
99                  node->c6 * (coordination->access(i, j - 2)↩
                    .zetat +
100                             coordination->access(i,
101                                                  j - 2).↩
                                                      zeta) ↩
                                                      +
102                 node->c7 * (coordination->access(i + 2, j)↩
                    .zetat +
103                             coordination->access(i + 2,
104                                                  j).zeta) ↩
                                                      +
105                 node->c8 * (coordination->access(i + 1, j)↩
                    .zetat +
106                             coordination->access(i + 1,
107                                                  j).zeta) ↩
                                                      +
108                 node->c9 * (coordination->access(i - 1, j)↩
                    .zetat +
109                             coordination->access(i - 1,
110                                                  j).zeta) ↩
                                                      +
111                 node->c10 * (coordination->access(i - 2, j↩
                    ).zetat +
112                              coordination->access(i - 2, j↩
                                 ).zeta)) / node->c1;
113         }
114
115         j = 1;
116         #pragma omp parallel for private(i, node)
117
118         for (i = leftterminal; i <= rightterminal; i++) {
119             node = &coordination->access(i, j);
120             node->newzetat =
121                     (node->c2 * node->zeta +
122                      node->c3 * (coordination->access(i, j ↩
                            + 2).zetat +
```

```
123                                 coordination->access(i,
124                                                      j + ↩
                                                        2)↩
                                                        .↩
                                                        zeta↩
                                                        ) ↩
                                                        +
125                         node->c4 * (coordination->access(i, j ↩
                               + 1).zetat +
126                                 coordination->access(i,
127                                                      j + ↩
                                                        1)↩
                                                        .↩
                                                        zeta↩
                                                        ) ↩
                                                        +
128                         node->c5 * (cylinderBoundary->access(i↩
                               , 1).zetat +
129                                 cylinderBoundary->access(i↩
                                    , 1).zeta) +
130                         node->c7 * (coordination->access(i + ↩
                               2, j).zetat +
131                                 coordination->access(i + ↩
                                    2,
132                                                      j).↩
                                                        zeta↩
                                                        ) ↩
                                                        +
133                         node->c8 * (coordination->access(i + ↩
                               1, j).zetat +
134                                 coordination->access(i + ↩
                                    1,
135                                                      j).↩
                                                        zeta↩
                                                        ) ↩
                                                        +
136                         node->c9 * (coordination->access(i - ↩
                               1, j).zetat +
137                                 coordination->access(i - ↩
                                    1,
138                                                      j).↩
                                                        zeta↩
                                                        ) ↩
                                                        +
139                         node->c10 * (coordination->access(i - ↩
                               2, j).zetat +
140                                  coordination->access(i - ↩
                                     2, j).zeta)) / node->↩
                                     c1;
141         }
142
```

```
143         j = -1;
144         #pragma omp parallel for private(i, node)
145
146         for (i = leftterminal; i <= rightterminal; i++) {
147             node = &coordination->access(i, j);
148             node->newzetat =
149                 (node->c2 * node->zeta +
150                  node->c4 * (cylinderBoundary->access(i, 0)↩
                        .zetat +
151                              cylinderBoundary->access(i,
152                                      0).zeta) +
153                  node->c5 * (coordination->access(i, j - 1)↩
                        .zetat +
154                              coordination->access(i,
155                                                   j - 1).↩
                                                      zeta) ↩
                                                      +
156                  node->c6 * (coordination->access(i, j - 2)↩
                        .zetat +
157                              coordination->access(i,
158                                                   j - 2).↩
                                                      zeta) ↩
                                                      +
159                  node->c7 * (coordination->access(i + 2, j)↩
                        .zetat +
160                              coordination->access(i + 2,
161                                                   j).zeta) ↩
                                                      +
162                  node->c8 * (coordination->access(i + 1, j)↩
                        .zetat +
163                              coordination->access(i + 1,
164                                                   j).zeta) ↩
                                                      +
165                  node->c9 * (coordination->access(i - 1, j)↩
                        .zetat +
166                              coordination->access(i - 1,
167                                                   j).zeta) ↩
                                                      +
168                  node->c10 * (coordination->access(i - 2, j↩
                        ).zetat +
169                               coordination->access(i - 2, j↩
                                  ).zeta)) / node->c1;
170         }
171
172
173         #pragma omp parallel for private(j, i)
174
175         for (j = downboundary + 2; j < upboundary - 1; j++)↩
                {
176             for (i = leftboundary + 2; i < rightboundary - ↩
                    1; i++) {
```

```
177                 coordination->access(i, j).zetat =
178                     coordination->access(i, j).newzetat;
179             }
180         }
181
182         converge -= 1;
183     }
184     */
185
186     /* flush zetat back to zeta
187     #pragma omp parallel for private(j, i)
188
189     for (j = downboundary + 2; j < upboundary - 1; j++) {
190         for (i = leftboundary + 2; i < rightboundary - 1; i↩
            ++) {
191             coordination->access(i, j).zeta = coordination↩
                ->access(i, j).zetat;
192         }
193     } */
```

- 以下列出多波前算法计算$\zeta$的代码，是我们最终采用的版本。它用来求解大规模稀疏矩阵，在我们的课题里用来求解各个点的$eta$值。在经历前一种算法的挫折后，我们很惊喜地发现在当前算法下数据很快就收敛了。

```
1  void cylinderProject::calculateNewZeta()
2  {
3      double vxi, veta;
4      double uxi, ueta;
5      double hxi, heta;
6      double lambdaxi, lambdaeta;
7      int i, j;
8      cylinderNode * node;
9      /* Calculating new zeta at t + deltat on inner nodes */
10     /* Calculating coefficients */
11     #pragma omp parallel for private(i, j, node, hxi, heta,↩
           vxi, veta, uxi, ueta, lambdaxi, lambdaeta)
12
13     for (i = downboundary + 2; i < upboundary - 1; i++) {
14         for (j = leftboundary + 2; j < rightboundary - 1; j↩
               ++) {
15             if ((i <= 1) && (i >= -1) && (j >= leftterminal↩
                   )
16                 && (j <= rightterminal)) {
17                 /* Near or on the cylinder, do nothing here↩
                       */
18                 continue;
19             }
20
21             node = &coordination->access(j, i);
```

```
22              hxi = node->hxi;
23              heta = node->heta;
24              vxi =
25                  (coordination->access(j, i + 1).psi -
26                   coordination->access(j, i - 1).psi) / (2 *↩
                        heta * deltaeta);
27              veta =
28                  -(coordination->access(j + 1, i).psi -
29                    coordination->access(j - 1, i).psi) / (2 ↩
                        * hxi * deltaxi);
30              uxi = vxi / hxi;
31              ueta = veta / heta;
32
33              if (Re < 1000) {
34                  lambdaxi = 1 - 1 / exp(abs(uxi));
35                  lambdaeta = 1 - 1 / exp(abs(ueta));
36              } else {
37                  lambdaxi = 1;
38                  lambdaeta = 1;
39              }
40
41              node->c1 =
42                  -2 / deltat - lambdaxi * abs(uxi) / (2 * ↩
                        deltaxi) -
43                  lambdaeta * abs(ueta) / (2 * deltaeta) -
44                  4 / (hxi * hxi * deltaxi * deltaxi * Re) -
45                  4 / (heta * heta * deltaeta * deltaeta * Re↩
                        );
46              node->c2 =
47                  -2 / deltat + lambdaxi * abs(uxi) / (2 * ↩
                        deltaxi) +
48                  lambdaeta * abs(ueta) / (2 * deltaeta) +
49                  4 / (hxi * hxi * deltaxi * deltaxi * Re) +
50                  4 / (heta * heta * deltaeta * deltaeta * Re↩
                        );
51              node->c3 = -(ueta - lambdaeta * abs(ueta)) / ↩
                    (12 * deltaeta);
52              node->c4 =
53                  (2 * ueta - lambdaeta * abs(ueta)) / (3 * ↩
                        deltaeta) -
54                  2 / (heta * heta * deltaeta * deltaeta * Re↩
                        );
55              node->c5 =
56                  -(2 * ueta + lambdaeta * abs(ueta)) / (3 * ↩
                        deltaeta) -
57                  2 / (heta * heta * deltaeta * deltaeta * Re↩
                        );
58              node->c6 = (ueta + lambdaeta * abs(ueta)) / (12↩
                    * deltaeta);
59              node->c7 = -(uxi - lambdaxi * abs(uxi)) / (12 *↩
                    deltaxi);
```

```
60              node->c8 =
61                  (2 * uxi - lambdaxi * abs(uxi)) / (3 * ↩
                       deltaxi) -
62                  2 / (hxi * hxi * deltaxi * deltaxi * Re);
63              node->c9 =
64                  -(2 * uxi + lambdaxi * abs(uxi)) / (3 * ↩
                       deltaxi) -
65                  2 / (hxi * hxi * deltaxi * deltaxi * Re);
66              node->c10 = (uxi + lambdaxi * abs(uxi)) / (12 *↩
                    deltaxi);
67          }
68      }
69
70      #pragma omp parallel for private(j, hxi, heta, vxi, ↩
            veta, uxi, ueta, lambdaxi, lambdaeta, node)
71
72      for (j = leftterminal; j <= rightterminal; j++) {
73          /* Upper half */
74          node = &coordination->access(j, 1);
75          hxi = node->hxi;
76          heta = node->heta;
77          vxi =
78              (coordination->access(j, 2).psi -
79               cylinderBoundary->access(j, 1).psi) / (2 * ↩
                     heta * deltaeta);
80          veta =
81              -(coordination->access(j + 1, 1).psi -
82                coordination->access(j - 1, 1).psi) / (2 * ↩
                       hxi * deltaxi);
83          uxi = vxi / hxi;
84          ueta = veta / heta;
85
86          if (Re < 1000) {
87              lambdaxi = 1 - 1 / exp(abs(uxi));
88              lambdaeta = 1 - 1 / exp(abs(ueta));
89          } else {
90              lambdaxi = 1;
91              lambdaeta = 1;
92          }
93
94          node->c1 =
95              -2 / deltat - lambdaxi * abs(uxi) / (2 * ↩
                     deltaxi) -
96              (-ueta) / (2 * deltaeta) -
97              4 / (hxi * hxi * deltaxi * deltaxi * Re) -
98              4 / (heta * heta * deltaeta * deltaeta * Re);
99          node->c2 =
100             -2 / deltat + lambdaxi * abs(uxi) / (2 * ↩
                    deltaxi) +
101             (-ueta) / (2 * deltaeta) +
102             4 / (hxi * hxi * deltaxi * deltaxi * Re) +
```

```
103             4 / (heta * heta * deltaeta * deltaeta * Re);
104         node->c3 = -(ueta - (-ueta)) / (12 * deltaeta);
105         node->c4 =
106             (2 * ueta - (-ueta)) / (3 * deltaeta) -
107             2 / (heta * heta * deltaeta * deltaeta * Re);
108         node->c5 =
109             -(2 * ueta + (-ueta)) / (3 * deltaeta) -
110             2 / (heta * heta * deltaeta * deltaeta * Re);
111         node->c6 = (ueta + (-ueta)) / (12 * deltaeta);
112         node->c7 = -(uxi - lambdaxi * abs(uxi)) / (12 * ↩
               deltaxi);
113         node->c8 =
114             (2 * uxi - lambdaxi * abs(uxi)) / (3 * deltaxi)↩
                  -
115             2 / (hxi * hxi * deltaxi * deltaxi * Re);
116         node->c9 =
117             -(2 * uxi + lambdaxi * abs(uxi)) / (3 * deltaxi↩
                ) -
118             2 / (hxi * hxi * deltaxi * deltaxi * Re);
119         node->c10 = (uxi + lambdaxi * abs(uxi)) / (12 * ↩
               deltaxi);
120
121         /* Lower half */
122         node = &coordination->access(j, -1);
123         hxi = node->hxi;
124         heta = node->heta;
125         vxi =
126             (cylinderBoundary->access(j, 0).psi -
127              coordination->access(j, -2).psi) / (2 * heta *↩
                     deltaeta);
128         veta =
129             -(coordination->access(j + 1, -1).psi -
130               coordination->access(j - 1, -1).psi) / (2 * ↩
                     hxi * deltaxi);
131         uxi = vxi / hxi;
132         ueta = veta / heta;
133
134         if (Re < 1000) {
135             lambdaxi = 1 - 1 / exp(abs(uxi));
136             lambdaeta = 1 - 1 / exp(abs(ueta));
137         } else {
138             lambdaxi = 1;
139             lambdaeta = 1;
140         }
141
142         node->c1 =
143             -2 / deltat - lambdaxi * abs(uxi) / (2 * ↩
                    deltaxi) -
144             ueta / (2 * deltaeta) -
145             4 / (hxi * hxi * deltaxi * deltaxi * Re) -
146             4 / (heta * heta * deltaeta * deltaeta * Re);
```

```
147         node->c2 =
148             -2 / deltat + lambdaxi * abs(uxi) / (2 * ↩
                   deltaxi) +
149             ueta / (2 * deltaeta) +
150             4 / (hxi * hxi * deltaxi * deltaxi * Re) +
151             4 / (heta * heta * deltaeta * deltaeta * Re);
152         node->c3 = -(ueta - ueta) / (12 * deltaeta);
153         node->c4 =
154             (2 * ueta - ueta) / (3 * deltaeta) -
155             2 / (heta * heta * deltaeta * deltaeta * Re);
156         node->c5 =
157             -(2 * ueta + ueta) / (3 * deltaeta) -
158             2 / (heta * heta * deltaeta * deltaeta * Re);
159         node->c6 = (ueta + ueta) / (12 * deltaeta);
160         node->c7 = -(uxi - lambdaxi * abs(uxi)) / (12 * ↩
               deltaxi);
161         node->c8 =
162             (2 * uxi - lambdaxi * abs(uxi)) / (3 * deltaxi)↩
                   -
163             2 / (hxi * hxi * deltaxi * deltaxi * Re);
164         node->c9 =
165             -(2 * uxi + lambdaxi * abs(uxi)) / (3 * deltaxi↩
                   ) -
166             2 / (hxi * hxi * deltaxi * deltaxi * Re);
167         node->c10 = (uxi + lambdaxi * abs(uxi)) / (12 * ↩
               deltaxi);
168     }
```

## 3.2 前端

# 4 课题过程及结果分析

## 4.1 课题完成情况

在开学第二次计算概论课上，我们几个人就觉得组成小组，完成课题。由于基础较好，对课题的前景十分乐观。我们挑选了很久项目的主题，最终决定做计算流体力学相关的课题。一来比较有挑战性，二来也切合物理学科。我们信心百倍的完成了开题报告，但在接下来的时间里只是看参考资料，相关书目。真正开始做已经是11月初。由于课题难度之大，直到近日才初步做完，可谓蹉跎。

具体分工来看，徐智怡完成后端的大部分工作。沈钟灵负责数学物理的相关问题、部分后端程序以及项目报告的书写。黄康靖同学则完成了前端的工作。在这个项目上，三人都投入大量时间精力，同时也收获了很多。

最后的效果差强人意，远不及当初那么理想。甚至还存在一些我们暂时无法解决的bug.比如圆球的大小显得很不合适。以下将列出在完成项目过程中我们所遇到的问题，以及项目的一些优点特点。

## 4.2 项目的问题和优点

### 4.2.1 问题

- 使用迭代法计算流函数和涡量时不收敛，应该是算法的精度不够高而至。最后我们换了方程的迭代方法，并利用umfpack求解。得到了较好的结果。
- 不能从数值上判断收敛，只能从图像上看。在作出前端以后我们才对初始值设定进行了进一步的修正，使得结果有更高可信度。

### 4.2.2 亮点

-
- 使用了Openmp进行并行计算。OpenMp是由OpenMP Architecture Review Board牵头提出的，并已被广泛接受的，用于共享内存并行系统的多线程程序设计的一套指导性注释(Compiler Directive)。OpenMP支持的编程语言包括C语言、C++和Fortran；而支持OpenMp的编译器包括Sun Compiler，GNU Compiler和Intel Compiler等。OpenMp提供了对并行算法的高层的抽象描述，程序员通过在源代码中加入专用的pragma来指明自己的意图，由此编译器可以自动将程序进行并行化，并在必要之处加入同步互斥以及通信。当选择忽略这些pragma，或者编译器不支持OpenMp时，程序又可退化为通常的程序(一般为串行)，代码仍然可以正常运作，只是不能利用多线程来加速程序执行。
- 可以中途存盘(仅后端可以，前端暂未实现）

# 参考文献


[1] 李万平 计算流体力学,华中科技大学 (2004)

[2] Intel Math Kernal Library

[3] UMPACK User Guide

[4] Qt 4.7 reference Documentation