计 算 流 体 力 学 报 告

班 级 06012001

姓 名 薛茵午

学 号 2020303828

一、 问题分析

对于一维的定常不可压缩粘性流动问题进行求解

$$\begin{cases} (x^2+5)\frac{du}{dx} - 2\frac{d^2u}{dx^2} = 2x, & x \le 0 \le 1 \\ u = 5, & x = 0 \\ 2\frac{du}{dx} = x+3 & x = 1 \end{cases} \tag{1}$$

下面使用有限元和有限差分两种方法, 分别对此问题进行分析求解和代码计算，并对其结果进行数据处理和比较

二、 一维流动问题的有限元解法

对于上述问题，如果我们使用有限元解法进行求解，可以通过下述步骤

1．分析和建立弱形式。

我们使用Б о р и с Г р и г о р ь е в и ч Г а л ё р к и н方法，取检验函数为δu，将方程两边同乘检验函数并进行积分，得到

$$\int_\Omega \left[(x^2+5)\frac{du}{dx} - 2\frac{d^2u}{dx^2}\right]\delta u dx = \int_\Omega 2x\delta u dx \tag{2}$$

我们设单元内的位移函数可以表示为单元中两个节点的位移值再乘对应的插值函数，即有$u = u_i\Phi_i$, 其中Φ_i为单元插值基函数。

在本题目的分析中，采用Lagrange方法，即假设插值基函数为线性形式，有$\Phi_i = a + bx$, 并满足条件$\Phi_i(x_j) = \delta_{ij}$, 此时，对于某个单元A, 可以解得其左节点$x_A$的插值函数和右节点$x_{A+1}$的插值函数为:

$$\Phi_A(x) = \frac{x_{A+1} - x}{x_{A+1} - x_A} \qquad \Phi_{A+1}(x) = \frac{x - x_A}{x_{A+1} - x_A} \tag{3}$$

此时单元内的任意一点的位移为:

$$u = u_i\Phi_i(x) = \Phi_A(x)x_A + \Phi_{A+1}(x)u_{A+1} \tag{4}$$

将方程(2)进行分部积分并整理，得到：

$$\int_0^1 (x^2+5)\frac{du}{dx}\delta u dx + 2\int_0^1 \frac{du}{dx}\frac{d(\delta u)}{dx}dx = \int_0^1 2x\delta u dx + 2\left[(\frac{du}{dx})\delta u\right]\Big|_{x=0}^{x=1} \tag{5}$$

我们将插值函数表达的位移(4)代入弱形式(5)中，则得到使用插值基函数表达的位移方程。

应当说明，由于边界$x=0$处给出的是本质边界条件，我们设此处的检验函数为0，即$\delta u(0)=0$, 代入边界条件$2\ \frac{du}{dx}\big|_{x=1} = x+3$则得到如下方程:

$$\int_0^1 (x^2+5)\frac{d\Phi_i}{dx}\Phi_j u_i dx + 2\int_0^1 \frac{d\Phi_i}{dx}\frac{d\Phi_j}{dx}u_i dx = \int_0^1 2x\Phi_j dx + (x+3)|_{x=1}\,\Phi_j(1) \tag{6}$$

我们将方程(6)改写成各个单元的积分之和, 则方程(6)变为:

$$\int_{x_1}^{x_2} (x^2+5)\frac{d\Phi_i}{dx}\Phi_j u_i dx + 2\int_{x_1}^{x_2} \frac{d\Phi_i}{dx}\frac{d\Phi_j}{dx}u_i dx = \int_{x_1}^{x_2} 2x\Phi_j dx + (x+3)|_{x=1}\,\Phi_j(1) \tag{7}$$

我们令:

$$\int_{x_1}^{x_2} \left[2\frac{d\Phi_i}{dx}\frac{d\Phi_j}{dx} + (x^2+5)\frac{d\Phi_i}{dx}\Phi_j\right]dx = A_{ij}, \qquad \int_{x_1}^{x_2} 2x\Phi_j dx + 4\Phi_j(1) = f_j \tag{8}$$

其中，显然$\Phi_j(1)$仅在$x=1$处等于1,其余节点处均为0，得到$A_{ij}u_i = f_j$, 转换下标即可得到矩阵形式的单元有限特征式为：

$$A_{ij}u_j = f_i \tag{9}$$

其中A_{ij}为单元刚度矩阵, f_i为右端项。需要注意的是，由于公式换标，有：

$$A_{ij} = \int_{x_1}^{x_2} \left[2\frac{d\Phi_i}{dx} + (x^2+5)\Phi_i\frac{d\Phi_j}{dx}\right]dx, \qquad f_i = \int_{x_1}^{x_2} 2x\Phi_i dx + 4\Phi_i(1) \tag{10}$$

2. 单元划分

在对原始形式的单元进行划分网格时, 将直线均分为10个单元, 共有11个节点, 划分结果如图1所示

由于转换成局部坐标比较麻烦，因此不转换为局部坐标，直接按照方程(8)求解每一个单元的刚度矩阵并进行组装,在程序中组装成 11 x 11 的整体刚度矩阵即可求解。

图 1: 11节点的网格单元划分

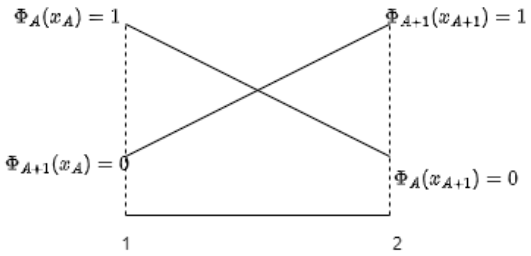

图 2: 单元内的形状函数Φ值图示

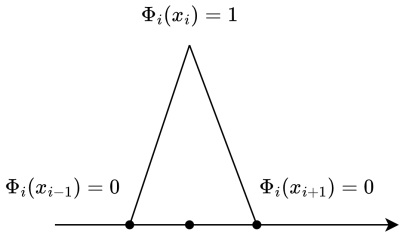

图 3: 全局形状函数Φ取值示例

在定义单元的形状函数时，每一个形状函数的值采用的是图2的插值方式, 我们为了简便, 使用图3所示方法来定义单元的整体形状函数。

在求解过程中，显然$\Phi(x)$是一次函数，因而区间内求导时, 结果必定为常数, 由于单元的形状函数是分两段的, 在对单元内的导数进行求解计算时, 我们只需取其中$\frac{x_A+x_{A+1}}{2}$处的导数值来代表整个区间的导数值即可。

3. 总体合成

单元形状插值基函数的构造和单元有限特征式的求解已经放在第1和第2部分。下面讲解单元坐标

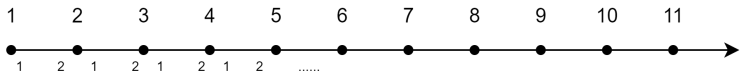

图 4: 单元整体编号和局部编号的示意图

建立下表存储每个单元的局部坐标和相对坐标的关系。列表如表1所示

单元号	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11

表 1: 单元号，整体坐标和局部坐标的表格

我们举例说明刚度矩阵的组接过程, 对于在第一部分求出的2x2单元刚度矩阵，以及对应的2x1的右端项, 可以组装如下：

$$\begin{bmatrix} A_{11}^1 & A_{12}^1 \\ A_{21}^1 & A_{22}^1 \end{bmatrix} \begin{bmatrix} u_1^1 \\ u_2^1 \end{bmatrix} = \begin{bmatrix} f_1^1 \\ f_2^1 \end{bmatrix} \qquad \begin{bmatrix} A_{11}^2 & A_{12}^2 \\ A_{21}^2 & A_{22}^2 \end{bmatrix} \begin{bmatrix} u_1^2 \\ u_2^2 \end{bmatrix} = \begin{bmatrix} f_1^2 \\ f_2^2 \end{bmatrix} \tag{11}$$

由于总体刚度矩阵是稀疏矩阵，因此可以直接调用Eigen库中的稀疏矩阵类，以减小存储空间和提高运算效率我们说明刚度矩阵和右端项的叠加过程。针对每一个单元，如(1)号单元和(2)号为例, 其刚度矩阵形式为则两个刚度矩阵的叠加形式为：

$$\begin{bmatrix} A_{11}^1 & A_{12}^1 & 0 \\ A_{21}^1 & A_{22}^1 + A_{11}^2 & A_{12}^2 \\ 0 & A_{21}^2 & A_{22}^2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} f_1^1 \\ f_2^1 + f_1^2 \\ f_2^2 \end{bmatrix} \tag{12}$$

其中u_1, u_2, u_3是全局坐标系下的节点坐标。我们按照这个形式，将所有刚度矩阵项和右端项依次叠加起来，即可得到整体刚度矩阵对应的方程为：

$$\begin{bmatrix} A_{11}^1 & A_{12}^1 & 0 & \cdots & 0 \\ A_{21}^1 & A_{22}^1 + A_{11}^2 & A_{12}^2 & & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & A_{21}^{10} & A_{22}^{10} + A_{11}^{11} & A_{12}^{11} \\ 0 & \cdots & & A_{12}^{11} & A_{22}^{11} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{10} \\ u_{11} \end{bmatrix} = \begin{bmatrix} f_1^1 \\ f_2^1 + f_1^2 \\ \vdots \\ f_2^{10} + f_1^{11} \\ f_1^{11} \end{bmatrix} \tag{13}$$

4. 边界条件的处理

采用主对角元素扩大法对边界条件进行处理, 设定一个大数为$\alpha = 10^{15}$, 由于本质边界条件是$u(0) = 5$, 并将f在坐标0处节点的一项（第一项）的值设为$\alpha A_{11} * u(0)$，方程变为:

$$\begin{bmatrix} 10^{15}A_{11}^1 & A_{12}^1 & 0 & \cdots & 0 \\ A_{21}^1 & A_{22}^1 + A_{11}^2 & A_{12}^2 & & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & & A_{21}^{10} & A_{22}^{10} + A_{11}^{11} & A_{12}^{11} \\ 0 & \cdots & & A_{12}^{11} & A_{22}^{11} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{10} \\ u_{11} \end{bmatrix} = \begin{bmatrix} 10^{15}A_{11}u(0) \\ f_2^1 + f_1^2 \\ \vdots \\ f_2^{10} + f_1^{11} \\ f_1^{11} \end{bmatrix} \tag{14}$$

5. 编程求解部分

1. 首先定义单元类，单元类用于存储单元的刚度矩阵，以及两个相对坐标在整体坐标系下的下标。

2. 定义单元的整体形状函数$\Phi(x)$函数，便于在求解单元刚度矩阵时直接进行调用

3. 建立单元每个节点坐标的向量cord存储每个节点的位置, 且按全局下标索引位置的值

4. 定义单元刚度矩阵函数求解 2x2 的单元刚度矩阵。在求解过程中，首先定义了一个函数 elem_Stiffness 来求解单元的形状函数矩阵, 采用C++的第三方库Boost提供的autodiff函数对对应的自动求导法求解微分, 而使用Boost库提供的gauss数值积分方法对区间值进行数值积分。并使用第三方库Eigen提供的矩阵类型进行存储。在组装刚度矩阵时，使用Eigen库自带的SparseMatrix稀疏矩阵类进行组装。

5. 按单元顺序将刚度矩阵和节点相对位置(参考表1)存储到单元向量中

6. 组接整体刚度矩阵，组接并求解函数整体求解右端项。

7. 使用Eigen库的SparseLU分解方法，求解稀疏矩阵表示的线性方程组

8. 将求解结果写入文件并用matlab进行后处理

有限元的结果部分和有限差分方法的结果，后处理和分析部分在第四部分会详细给出。

有限元解法的程序代码部分可见附录，也可以我的的github项目地址找到, 项目的地址为https://github.com/FRIEDparrot/CFD_Example_FEM_and_FDM，另外Eigen和Boost库的配置方法也可以在此处找到。

三、 一维流动问题的有限差分解法

在问题中，对于给出的方程, 第一式为在流动区域内的流动方程(也即控制方程), 第二式和第三式为流动的边界条件。

对于流动区域内流动方程的有限差分方法求解，我们在求解过程中，将一条线划分为10个单元, 因此节点的数目共有11个, 如图1 所示。

在控制方程的求解中, 使用中心差商形式代替控制方程的一阶和二阶导数。使用泰勒公式进行展开，容易推导出对应的二阶精度有限差分公式, 即

$$u'(x) = (\frac{u_{i+1} - u_{i-1}}{2\Delta x}) + O(\Delta x^2) \tag{15}$$

以及二阶导数的替代公式

$$u''(x) = \frac{u_{i+1} + u_{i-1} - 2u_i}{2\Delta x} + O(\Delta x^2) \tag{16}$$

上述两式均为二阶精度的差分公式, 具有二阶误差项

代入上式后，方程变为

$$(x_i^2 + 5)\frac{u_{i+1} - u_{i-1}}{2dx} - 2\frac{u_{i+1} + u_{i-1} - 2u_i}{dx^2} = 2x \tag{17}$$

且此方程具有二阶精度

我们在方程求解的迭代中, 每一次迭代先进行施加边界条件。边界条件的施加方法是使用向量存储各点的速度，在迭代之前, 为了流体域内的边界条件满足要求, 先对迭代的向量的边界先赋予边界条件，即将边界条件变为：

$$\begin{cases} u_0 = 5 \\ u_i = u_{i-1} + \frac{1}{2}(x + 3) \end{cases} \tag{18}$$

将边界条件施加后，将a[0] (第1个点) 和 a[size](第n个点，此处$n = 11$) , 直接赋值给新向量, 并使用原向量中的值代入控制方程,计算出中间点的值

使用离散化的控制方程(18) , 容易解得迭代公式(此处不使用隐式差分格式)

$$u_i = \frac{1}{2}(u_{i+1} + u_{i-1}) + \frac{1}{2}x\Delta x^2 - \frac{1}{8}(x^2 + 5)(u_{i+1} - u_{i-1})\Delta x \tag{19}$$

每一次施加边界条件并使用迭代公式进行迭代,设定误差e = 0.00001 并迭代至收敛, 即可解得每一点的速度分量。有限差分解答的结果和分析也已经放在结果分析部分，代码见附录。

四、 两种方法的结果分析和后处理部分

对于有限元和有限差分方法，的代码分别可以得到如下的我们分别将其进行列表和绘图处理

编号	1	2	3	4	5	6	7	8	9	10	11
位置	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
FEM解	5.0000	5.0280	5.0628	5.1055	5.1571	5.2195	5.2950	5.3868	5.4999	5.6408	5.8192
FDM解	5.0000	5.0299	5.0674	5.1133	5.1691	5.2370	5.3196	5.4208	5.5461	5.7032	5.9032

表 2: 使用有限差分方法求解得到的结果

所解得的速度结果和误差随迭代次数的收敛性进行分析和绘制图像，结果如图5, 6 所示

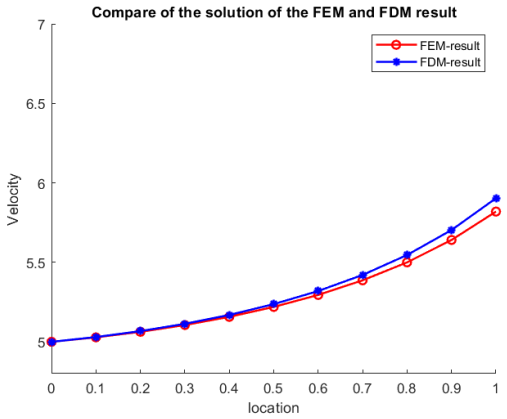

图 5: 速度结果分析

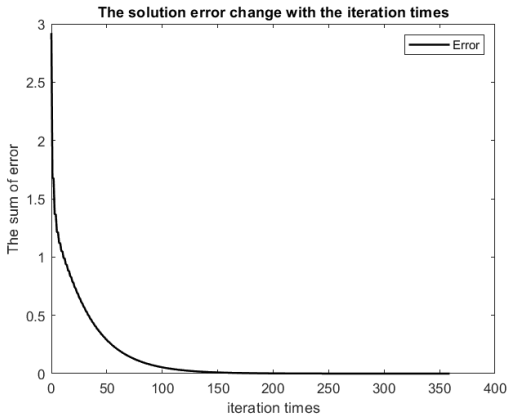

图 6: 有限差分法的误差收敛分析

从上图中的结果我们可以看出，有限元解法和有限差分的解法结果是比较接近的，考虑到有限元解的误差以及有限差分解的阶次舍入误差，我们可以认为求得的结果是较满意的。

附录

五、 代码部分

在代码中使用c++代码进行编写求解得到结果并使用matlab代码整理绘制图像，c++和matlab代码如下(注：由于Latex不支持汉语注释，我删除了代码中的注释部分, 如果需要注释部分和源代码，可以去github文件地址找到):

对于FEM求解部分，需要配置和使用第三方库Eigen和Boost，github文件中有详细教程

本项目的github开源地址为：https://github.com/FRIEDparrot/CFD_Example_FEM_and_FDM

1. FEM 问题求解代码部分

```
#include <iostream>
#include <vector>
#include <boost/lexical_cast/detail/converter_lexical.hpp>
#include <boost/math/differentiation/autodiff.hpp>
#include <boost/multiprecision/cpp_bin_float.hpp>
#include <boost/math/quadrature/gauss.hpp>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <unordered_map>
#include <fstream>

using namespace boost::math::differentiation;
using namespace boost::math::quadrature;
using namespace Eigen;

typedef std::vector<Triplet<float>> trip;

```

```
18 #pragma region ClassExtension
19 struct PairHash {
20 template <class T1, class T2>
21 std::size_t operator() (const std::pair<T1, T2>& p) const {
22 auto h1 = std::hash<T1>{}(p.first);
23 auto h2 = std::hash<T2>{}(p.second);
24 return h1 ^ h2;
25 }
26 };
27
28 class element {
29 public:
30 element(std::vector<int> nodeIndex, Matrix2f K_e) {
31 this->nodeIndex = nodeIndex;
32 this->K_e = K_e;
33 };
34 Matrix2f K_e;
35 std::vector<int> nodeIndex;
36 };
37
38 template<typename W, typename X, typename Y>
39 promote<W, X, Y> Phi(W x, X x_i, Y h) {
40 if (x <= x_i - h || x >= x_i + h) return 0;
41 if (x < x_i) return 1.0 - (x_i - x) / h;
42 return 1.0 - (x - x_i) / h;
43 }
44
```

```
45 void output_file(std::vector<float> location, std::vector<float> velocity) {
46 std::ofstream File("data_fem.csv");
47 if (!File) {
48 std::cout << "error when opening the file" << std::endl;
49 exit(0);
50 }
51 else {
52 File << "Location" << "," << "velocity" << std::endl;
53 for (int i = 0; i < location.size(); i++)
54 File << location[i] << "," << velocity[i] << std::endl;
55 File.close();
56 }
57 }
58
59 #pragma endregion
60
61 #pragma region Stiff_Derivation
62 Matrix2f Elem_Stiffness(double x_i, double x_j, double h) {
63 Matrix2f elem_mat;
64 elem_mat.setZero();
65 float xVec[]{ x_i, x_j };
66 float dPhi[2];
67 constexpr unsigned Order = 1;
68 auto const x = make_fvar<double, 1> (x_i + x_j)/2;
69 constexpr unsigned Nx = 1;
70 constexpr unsigned Nx_i = 0;
71 constexpr unsigned Nh = 0;
```

```
72 auto const variables1=make_ftuple<float,Nx,Nx_i,Nh>((x_i + x_j)/2,x_i,h);
73 auto const variables2=make_ftuple<float,Nx,Nx_i,Nh>((x_i + x_j)/2,x_j,h);
74 auto const& _x = std::get<0>(variables1);
75 auto const& _xi = std::get<1>(variables1);
76 auto const& _xj = std::get<1>(variables2);
77 auto const& _h = std::get<2>(variables1);
78 auto const y1 = Phi(_x, _xi, _h);
79 auto const y2 = Phi(_x, _xj, _h);
80 dPhi[0] = y1.derivative(Nx, Nx_i, Nh);
81 dPhi[1] = y2.derivative(Nx, Nx_i, Nh);
82 for (int i = 0; i < 2; i++) {
83 std::vector<float> rowVec;
84 for (int j = 0; j < 2; j++) {
85 auto a = [&](const float& x) {
86         return 2.0 * dPhi[i] * dPhi[j] +
87         (5.0 + (double)x * x) * Phi(x, xVec[i], h) * dPhi[j];
88 };
89 auto result = gauss<float, 10>::integrate(a, x_i, x_j);
90 elem_mat(i, j) = (float)result;
91 }
92 }
93 return elem_mat;
94 }
95
96 SparseMatrix<float> Tot_Stiffness(int elem_num,
97         int node_num ,std::vector<element*> elems){
98 SparseMatrix<float> A(node_num, node_num);
```

```
 99 A.setZero();
100 trip triplets;
101 triplets.reserve(3 * elem_num + 1);
102 std::unordered_map<std::pair<int, int>, float, PairHash> visited;
103 std::vector<std::pair<int, int>> indexPair;
104 for (element* elem : elems) {
105 for (int i = 0; i < 2; i++) {
106 for (int j = 0; j < 2; j++) {
107         int m = elem->nodeIndex[i];
108         int n = elem->nodeIndex[j];
109         if (visited.find(std::make_pair(m, n)) != visited.end()) {
110                 visited[std::make_pair(m, n)] += elem->K_e(i, j);
111         }
112         else {
113                 visited[std::make_pair(m, n)] = elem->K_e(i, j);
114                 indexPair.push_back(std::make_pair(m, n));
115         }
116 }
117 }
118 }
119 for (std::pair<int, int> P : indexPair) {
120 triplets.push_back(Triplet<float>(P.first, P.second, visited[P]));
121 }
122 A.setFromTriplets(triplets.begin(), triplets.end());
123 A.makeCompressed();
124 return A;
125 }
```

```
126
127 VectorXf fVecGenerate(int elem_num, int node_num,std::vector<element*> elems,
128         std::vector<float> location) {
129 VectorXf FVec(node_num);
130 FVec.setZero();
131 for (int index = 0; index < elem_num; index++) {
132 int i_index = elems[index]->nodeIndex[0];
133 int j_index = elems[index]->nodeIndex[1];
134 float x_i = location[i_index];
135 float x_j = location[j_index];
136 float h = x_j - x_i;
137
138 for (int i = 0; i < 2; i++) {
139 int index_relative = elems[index]->nodeIndex[i];
140 float loc_relative = location[index_relative];
141 auto f = [&](const float& x) {
142         return 2.0 * x * Phi(x, loc_relative , h);
143 };
144 float append = 4 * Phi(1, loc_relative, h);
145 auto result = gauss<float, 10>::integrate(f, x_i, x_j) + append;
146 FVec(index_relative) += result;
147 }
148 }
149 return FVec;
150 }
151
152 #pragma endregion
```

```
153
154 #pragma region main_programs
155 void Solution1d(float length,int elem_num) {
156 int node_num = elem_num + 1;
157 float h = length / elem_num;
158
159 std::vector<element*> elems;
160 std::vector<float> cord;
161 for (int i = 0; i < node_num; i++) {
162 cord.push_back(i * h);
163 }
164 for (int i = 0; i < elem_num ; i++) {
165 elems.push_back(new element(std::vector<int> { i, i + 1 },
166  Elem_Stiffness(cord[i], cord[i + 1], h)));
167 }
168
169 SparseMatrix<float> tot_Stiff = Tot_Stiffness(elem_num, node_num, elems);
170 VectorXf FVec = fVecGenerate(elem_num, node_num, elems, cord);
171 for (SparseMatrix<float>::InnerIterator iter(tot_Stiff, 0); iter ; ++iter) {
172 if (iter.row() == 0 && iter.col() == 0) {
173 iter.valueRef() *= 1e15;
174 FVec[0] = 5 * iter.value();
175 }
176 }
177
178 std::cout << "---------- The Stiff Matrix ---------- " << std::endl <<
    tot_Stiff << std::endl;
```

```
179 std::cout << "---------- The FVector ---------- " << std::endl
    << FVec << std::endl;
180
181 SparseLU<SparseMatrix<float>> solver;
182 solver.analyzePattern(tot_Stiff);
183 solver.factorize(tot_Stiff);
184
185 VectorXf u = solver.solve(FVec);
186
187 std::cout <<"---------- The Result ---------- " << std::endl
188         << u << std::endl;
189 std::vector<float> velocity_vec(u.data(), u.data() + u.size());
190 output_file(cord, velocity_vec);
191 }
192
193 int main()
194 {
195 int elem_num = 10;
196 float length = 1.0;
197 Solution1d(length, elem_num);
198 return 0;
199 }
200 #pragma endregion
```

2. FDM 问题求解代码部分

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <fstream>
using namespace std;

int inlet_velocity = 5;

void boundary_condition(vector<float>& velocity_vec, float length, float dx) {
velocity_vec[0] = inlet_velocity;
int size = velocity_vec.size() -1;
int x2 = length;
velocity_vec[size] = velocity_vec[size - 1] + (float)1/2 * (length + 3) * dx;
}

float calcu_err(vector<float> vec1, vector<float> vec2) {
        if (vec1.size() != vec2.size()) exit(-1);
        float sum = 0;
        for (int i = 0; i < vec1.size(); i++) {
                sum +=abs(vec2[i] - vec1[i]);
        }
        return sum;
}

void writefile(vector<float> velocity, vector<float> error, float dx) {
```

```
26 std::ofstream velocity_data("velocity_data.csv");
27 std::ofstream error_data("error_data.csv");
28 if (!velocity_data) {
29         cout << "error when opening the file" << endl;
30         exit(0);
31 }
32 for (int i = 0; i < velocity.size(); i++) {
33         velocity_data  << i << ","<< dx * i << "," << velocity[i] << endl;
34 }
35 for (int i = 0; i < error.size(); i++) {
36         error_data << i << "," << error[i] << endl;
37 }
38 velocity_data.close();
39 error_data.close();
40 }
41
42 int main()
43 {
44 float max_error = 0.00001;
45 int xelem_num = 10;
46 float length = 1;
47
48 float dx = length / xelem_num;
49 float error;
50
51 vector<float> velocity_vec(xelem_num +1, 0);
52 vector<float> loc_vec;
```

```
53 vector<float> error_data;
54
55 for (int i = 0; i < xelem_num + 1; i++) loc_vec.push_back(i * dx);
56
57 do{
58         boundary_condition(velocity_vec,length, dx);
59         vector<float> new_velocity_vec(velocity_vec);
60         for (int i = 1; i < xelem_num; i++) {
61         new_velocity_vec[i] = (float)1/2 * loc_vec[i] * dx * dx
62         + (float)1/2 * (velocity_vec[i-1] + velocity_vec[i+1])
63         - (float)1/8 * dx *(loc_vec[i] * loc_vec[i] + 5) *
64         (velocity_vec[i + 1] - velocity_vec[i-1]);
65         }
66         error = calcu_err(velocity_vec, new_velocity_vec);
67         error_data.push_back(error);
68         velocity_vec = new_velocity_vec;
69 }while (error >= max_error);
70
71 for (int i = 0; i < xelem_num + 1; i++) {
72         cout << i << " " << velocity_vec[i] << endl;
73 }
74
75 vector<float> prof_vec1(xelem_num + 1,0);
76 vector<float> prof_vec2(xelem_num + 1,0);
77 for (int i = 1; i < xelem_num; i++) {
78 prof_vec1[i] = (velocity_vec[i + 1] - velocity_vec[i - 1]) / (2 * dx);
79
```

```
80  prof_vec2[i] = (velocity_vec[i + 1] + velocity_vec[i - 1] -
81  2 * velocity_vec[i]) / (dx * dx);
82
83  float x = loc_vec[i];
84  cout << "calculated: " << (x * x + 5) * prof_vec1[i] - 2 * prof_vec2[i]
85  << "; predicted: " << 2 * x << endl;
86  }
87
88  writefile(velocity_vec, error_data, dx);
89  return 0;
90  }
```

3. Matlab数据整理代码部分

```
1  addpath("FDM_solution\CFD_Project3");
2  addpath("FEM_soution\CFD_Project1");
3
4  FDM_velocity = load("FDM_solution\CFD_Project3\velocity_data.csv");
5  FDM_error = load("FDM_solution\CFD_Project3\error_data.csv");
6  FEM_velocity = readmatrix("FEM_soution\CFD_Project1\data_fem.csv");
7
8  figure("Name", "velocity graph");
9
10 hold on
11 x = FDM_velocity(:,2);
12 plot (x, FEM_velocity(:,2), 'ro-' ,'LineWidth', 1.5);
13 plot (x, FDM_velocity(:,3), 'b*-' ,'LineWidth', 1.5);
14 axis([0 1 4.8 7])
```

```
15 legend("FEM-result", "FDM-result");
16 title("Compare of the solution of the FEM and FDM result");
17 xlabel("location");
18 ylabel("Velocity");
19
20 figure("Name","error graph");
21 plot (FDM_error(:, 1), FDM_error(:,2),'k-','LineWidth', 1.5);
22 legend("Error");
23 title("The solution error change with the iteration times");
24 xlabel("iteration times")
25 ylabel("The sum of error")
```