**使用涡量-流函数法求解顶盖驱动的方腔流动**

**实验报告**

管飞宇U201715262

李迪彦U201712043

蔡玉娟U201716132

2019年12月

一、实验背景

在计算流体力学的研究中，方腔驱动流求解问题是常见的用于各种数值计算方法有效性的检验手段。本文利用均匀网格开展计算，采用c++语言计算，使用涡量流函数法对方腔驱动流问题进行了模拟分析，并最终用Tecplot软件进行的绘图处理。并且尝试了simple方法进行模拟仿真分析并进行了比对。

二、问题描述

物理模型及边界条件如右图，方腔边长为1m，上端速度为*v=1m/s*。模拟Re=100，400， 1000三组雷诺数的流动情况。

三、问题原理分析

基本方程：

在直角坐标系下，不可压非定常流体所满足的流函数涡量形式的N-S方程为：

其中



差分格式：



对于本问题，将方腔四边同时分为等分，则有

故



边界条件：

在腔体的两侧和顶边，

在底边

其中代表与边界相邻的节点。

而



即

结果表达:

速度计算

四、实验过程

我们首先定义网格，我们生成均匀的（n+1）×（n+1）网格，其中Δx=1/（n），Δy=1/（n）。第二步，我们设置边界条件，初始化流场边界，上壁面*u=1*，*v=0*，其他壁面上*u, v*均为0，涡量/流函数边界各壁面上Φ=0，上壁面*∂*Φ*/∂y*=1。第三步中，编写控制方程求解新流场进行迭代。𝛷𝑛+1=𝛷𝑛+𝛼\*d 𝛷。第四步，我们进行了收敛判断精度条件，即在何时跳出循环；涡量，流函数的迭代残差的最大值均小于标准值时可跳出循环进行下一轮的计算。第五步为dat文件数据输出。第六步为将数据文件导入Tecplot绘图。

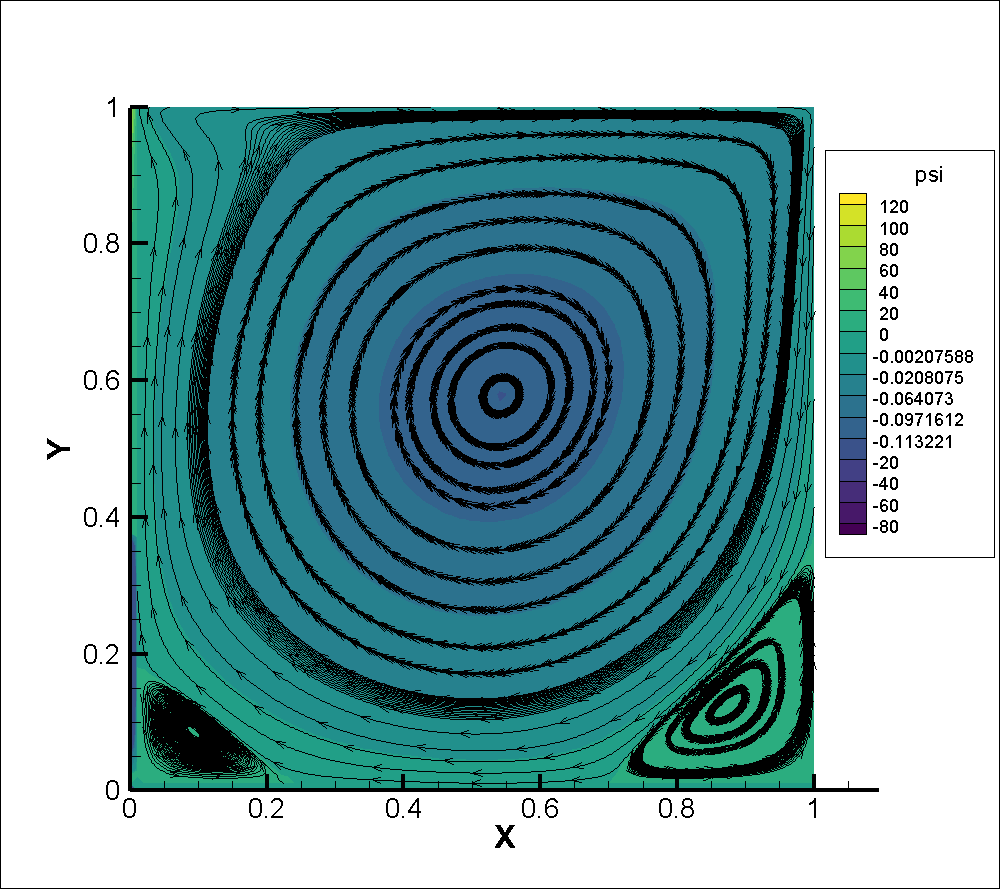
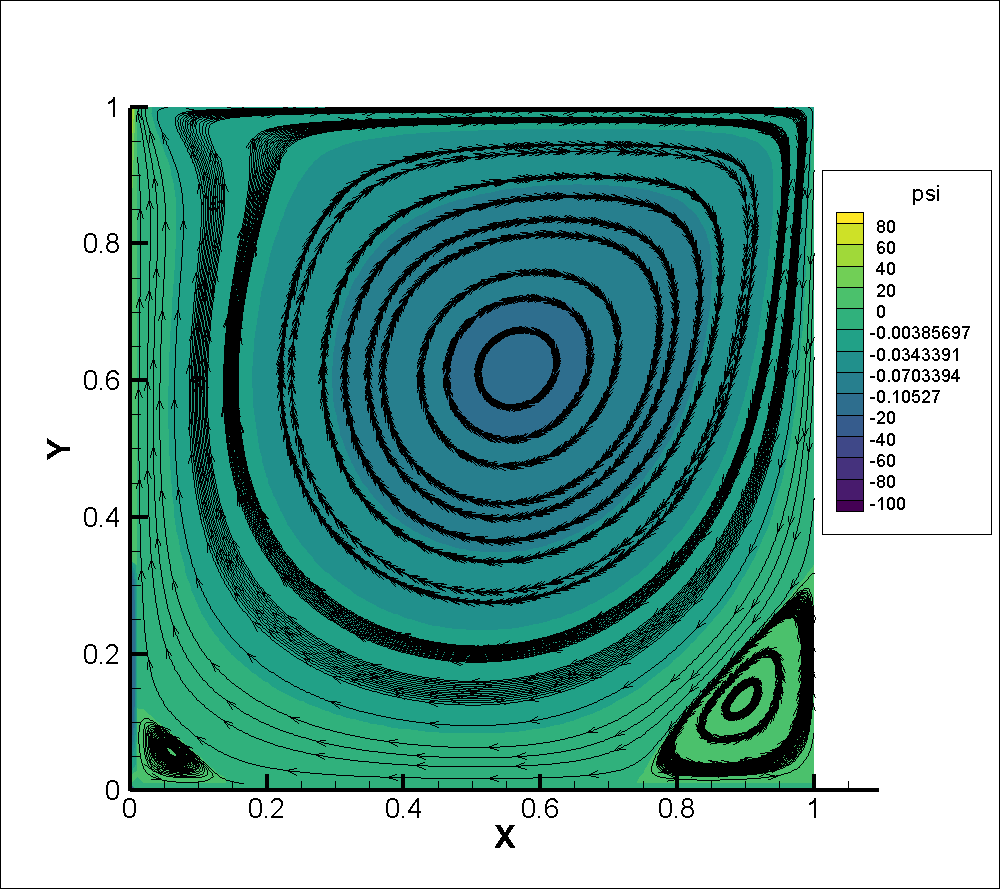
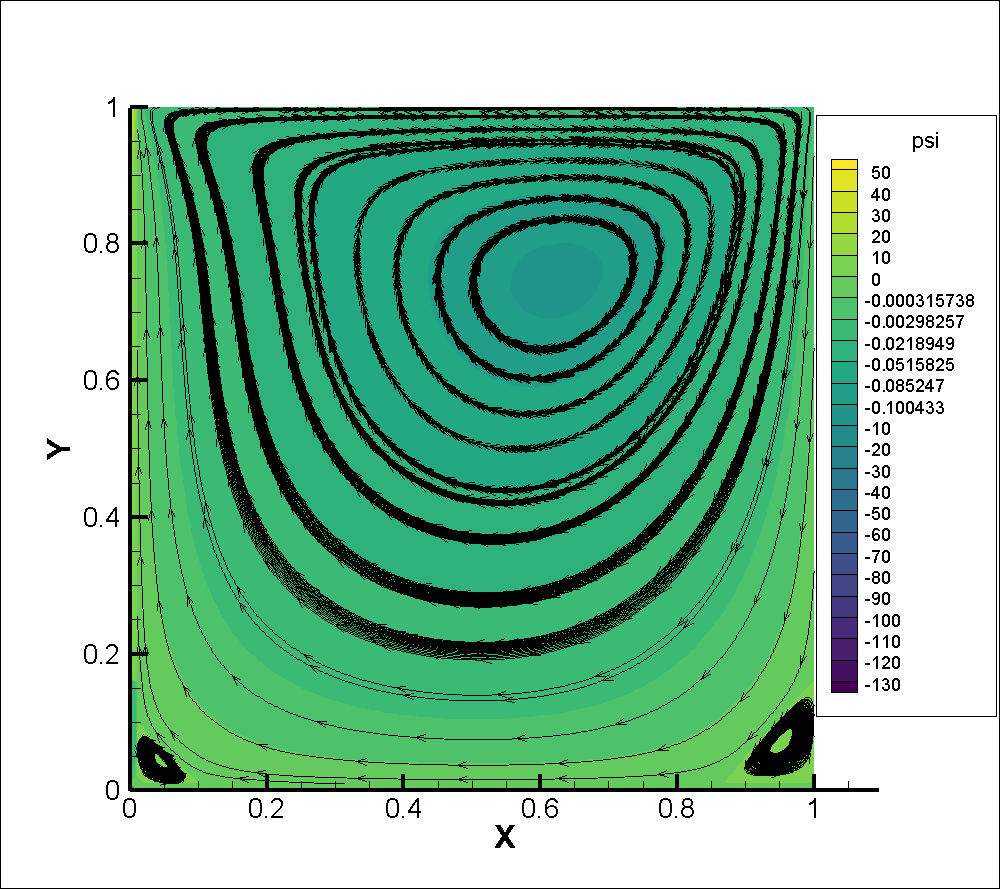
在代码实现方面：

第一个部分定义变量，输入网格数、雷诺数和时间步长，接着执行main函数，在主函数具体计算中，首先残余涡量，然后计算出涡量函数。然后用函数计算出边界处流函数和涡函数，最后计算u，v的分布。第三部分是检验 ，通过检验残余涡量和残余流量的最大值，决定是否要进行下一轮的计算。最后一部分是输出数据，每隔100次，输出*x,y,u,v,*xi（即涡量）和psi（即流函数）。

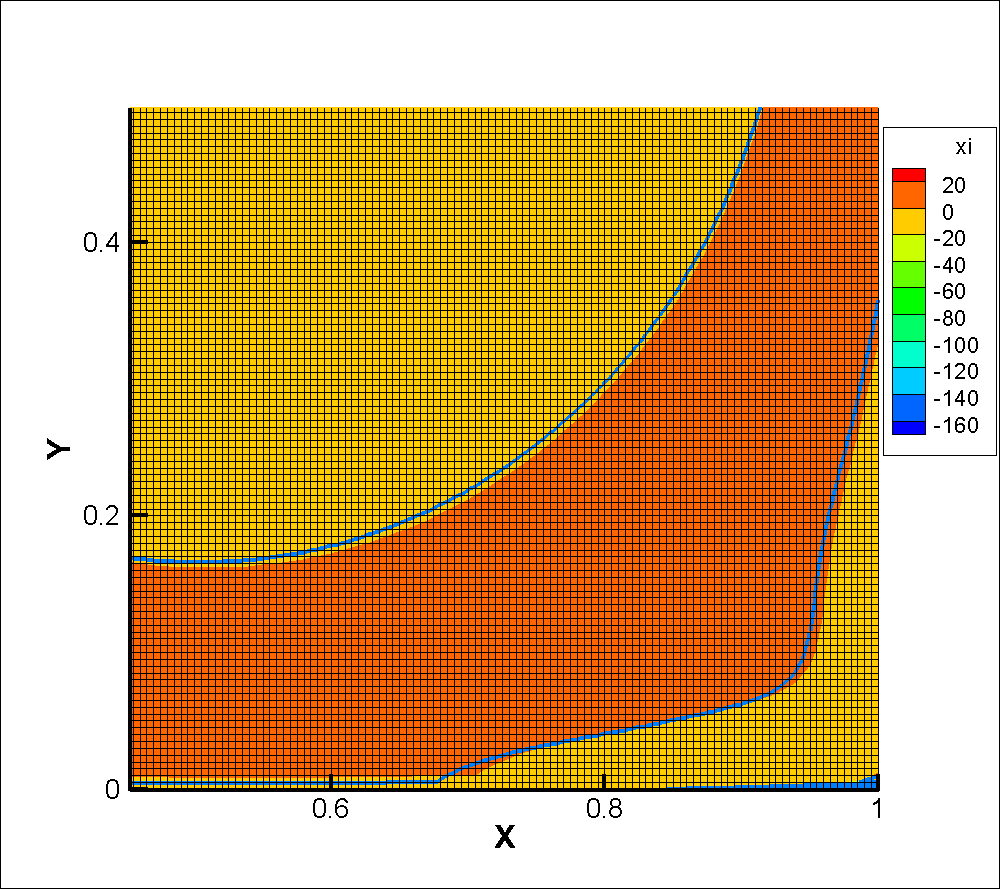
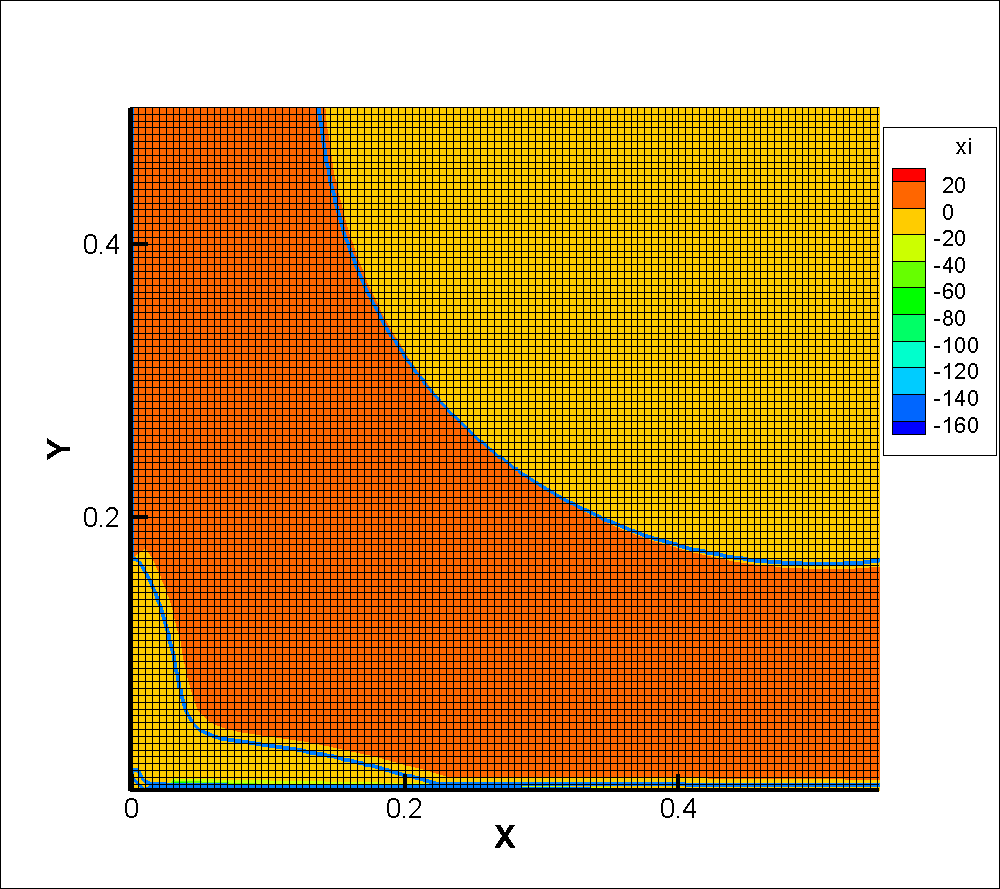
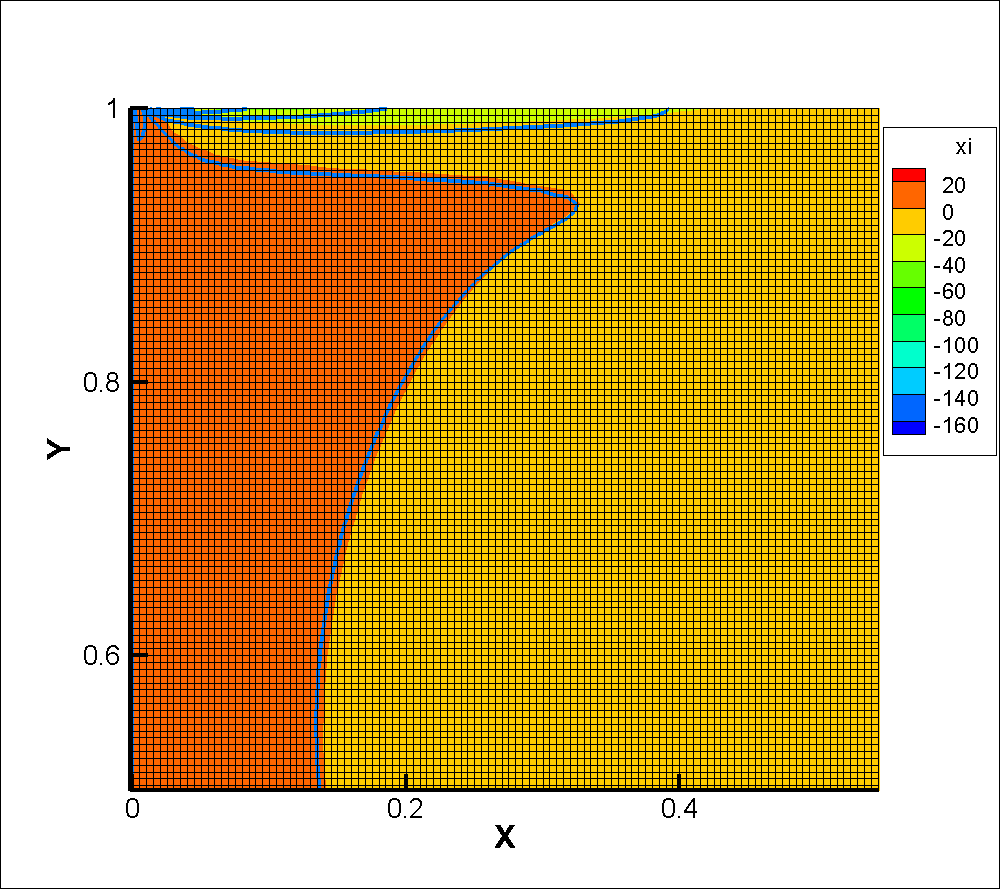
编写流程图为：

五、结果分析

1、模拟Re=100，400， 1000三个雷诺数下的流动情况

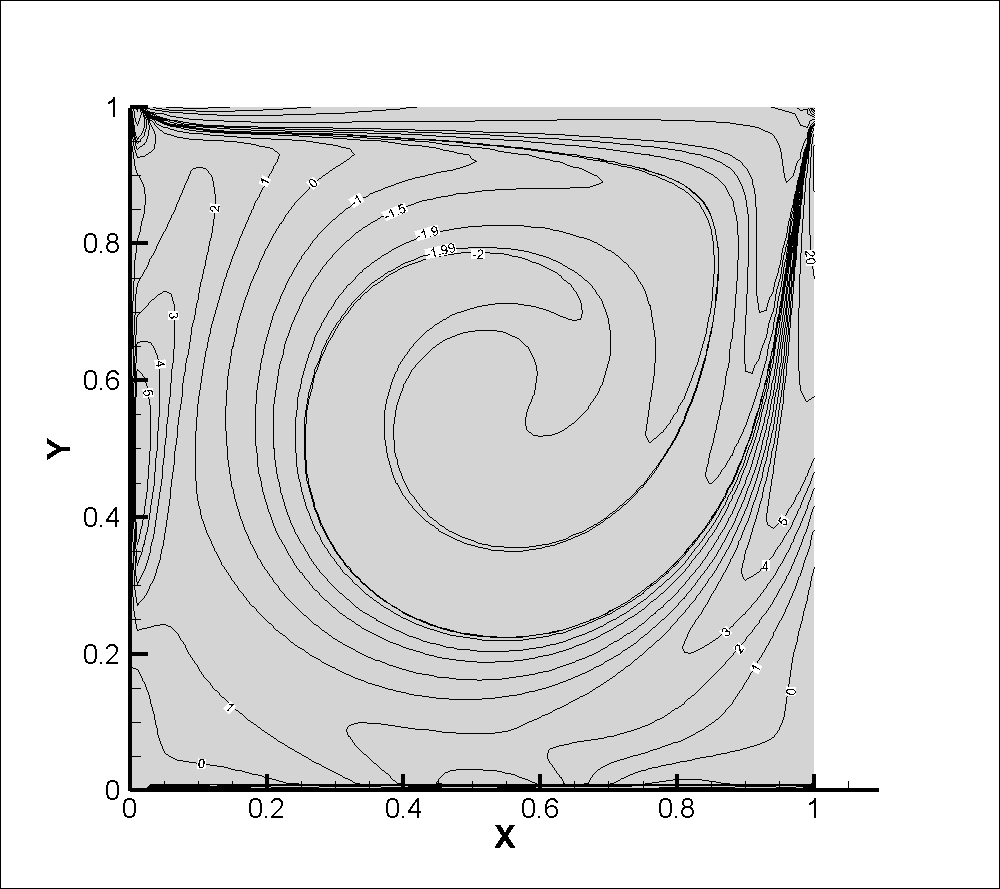
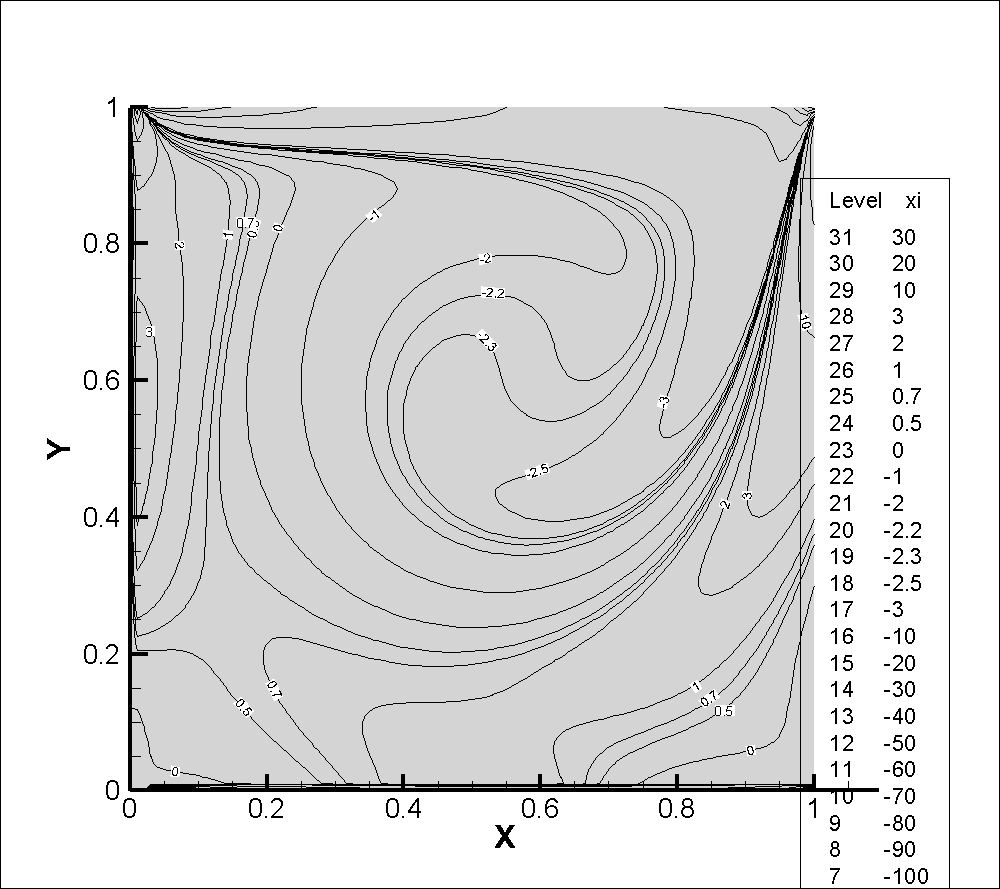
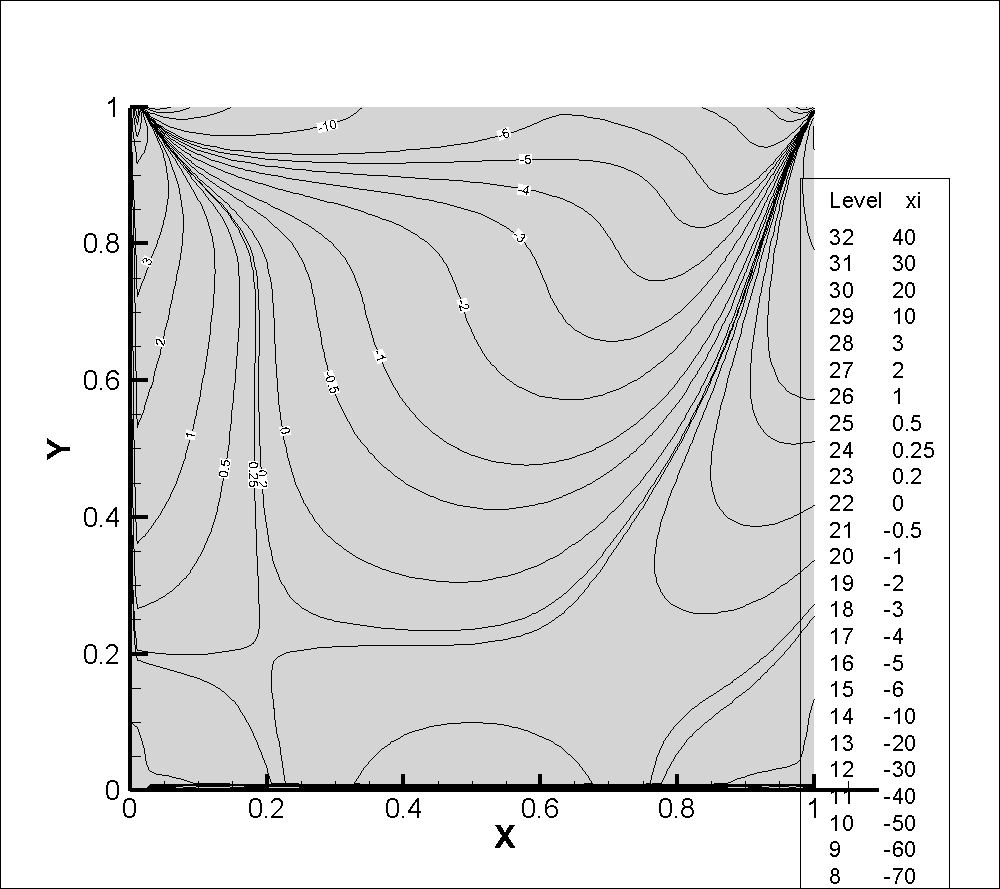
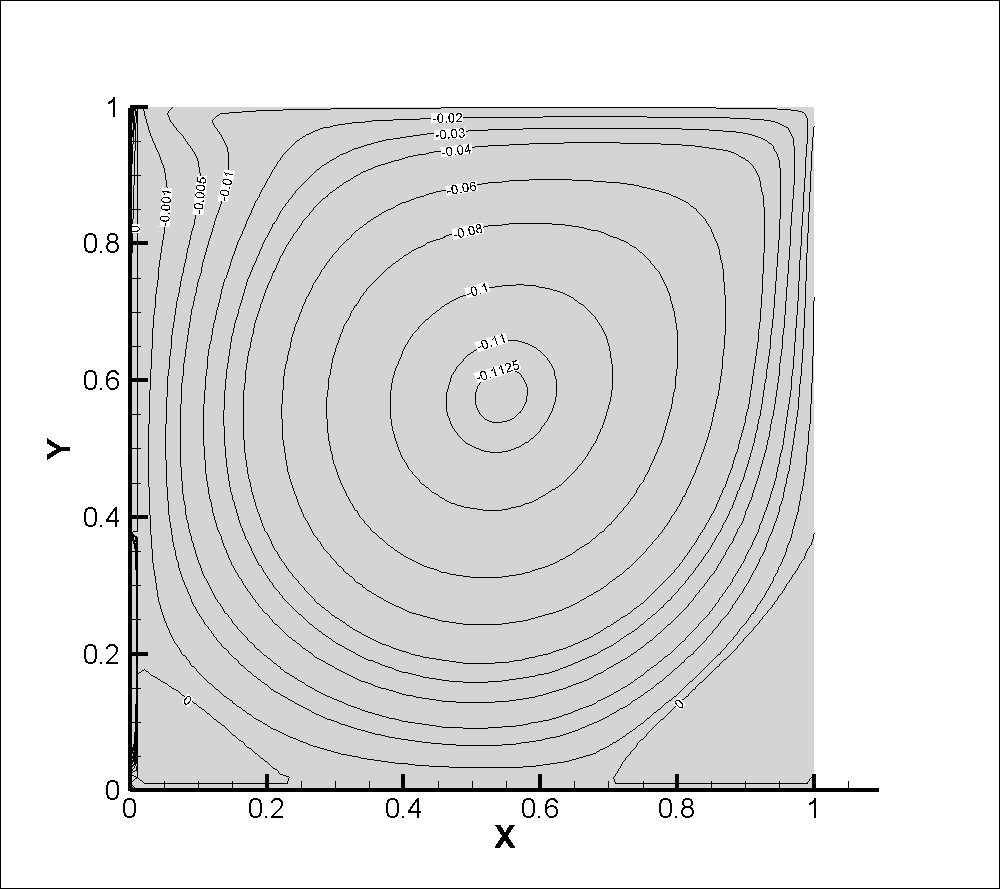
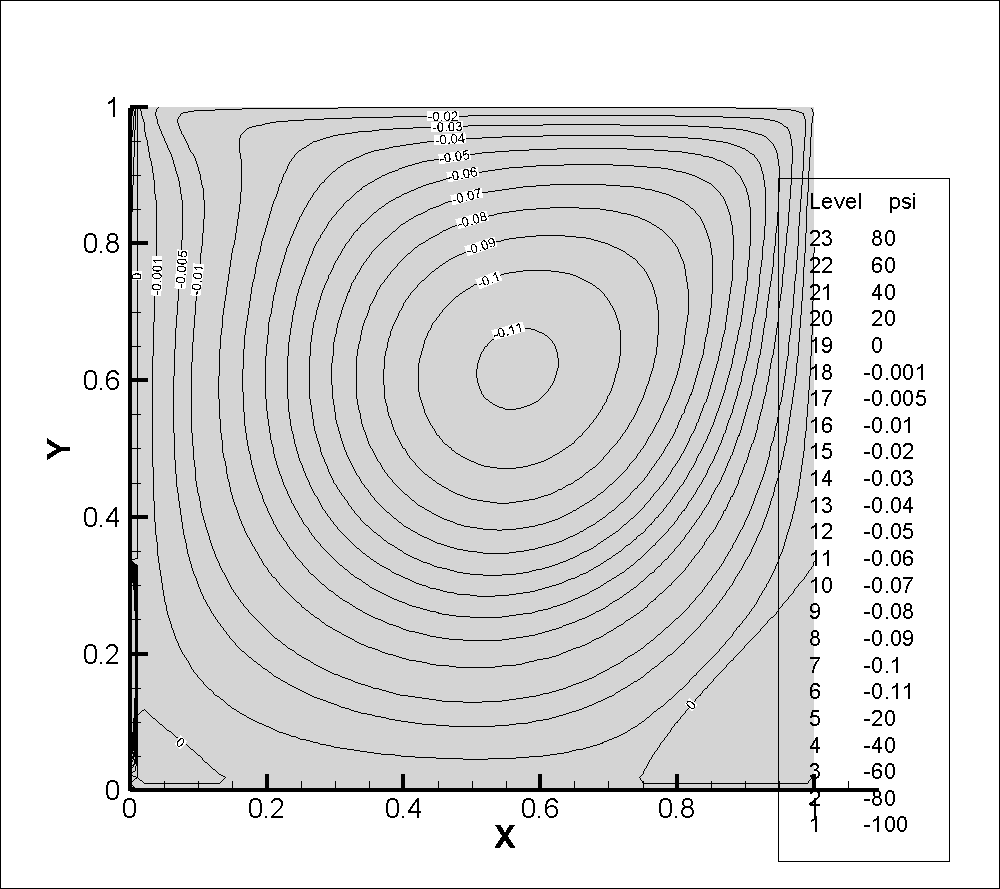
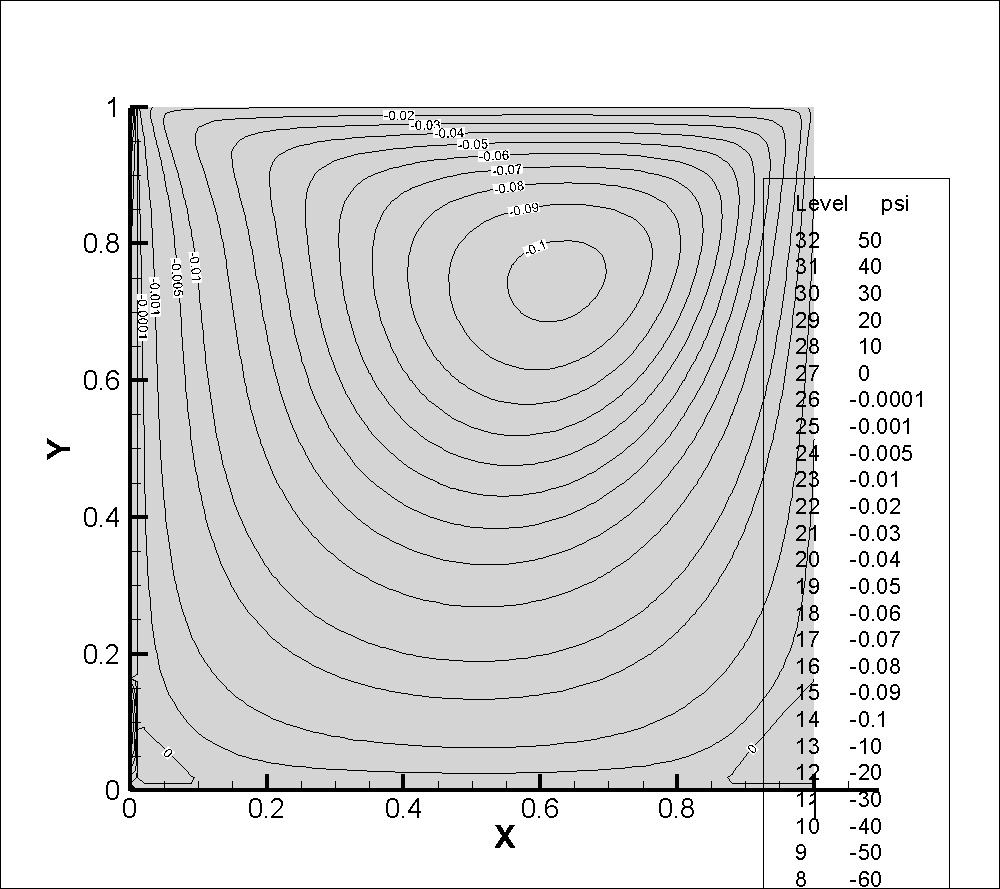


如上图所示，从左到右分别是雷诺数Re=100、400和1000的流动情况，做出不同情况下的云图和迹线。可以看出随着雷诺数的增大，流体出现的漩涡更大，更倾向于表现出湍流，即随着雷诺数的增大，粘滞力带来的影响逐渐变小，惯性对流场的影响增大，流体稳定性变差，符合实际情况，侧面证明了仿真的正确性。

2、选取Re=1000，使用至少两种均匀网格开展计算，考察网格对计算结果的影响

上图分别为n=100和200的流场重叠后，左上角、左下角、右下角流动情况，橙色和黄色系图块代表n=200的流场，蓝色线条代表n=100的流场。可以看出流场的等值线位置存在差异。说明不同的网格密度会对最终结果产生一定的影响，我们猜想可能是由于网格精度变高之后结算结果更多更精细从而影响到了之后的计算结果。

3、画出涡量和流函数等值线图

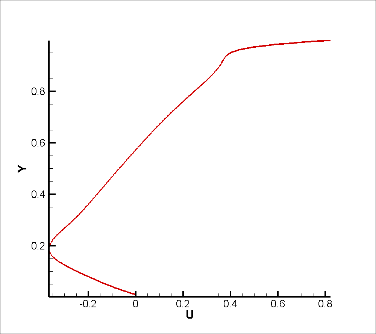
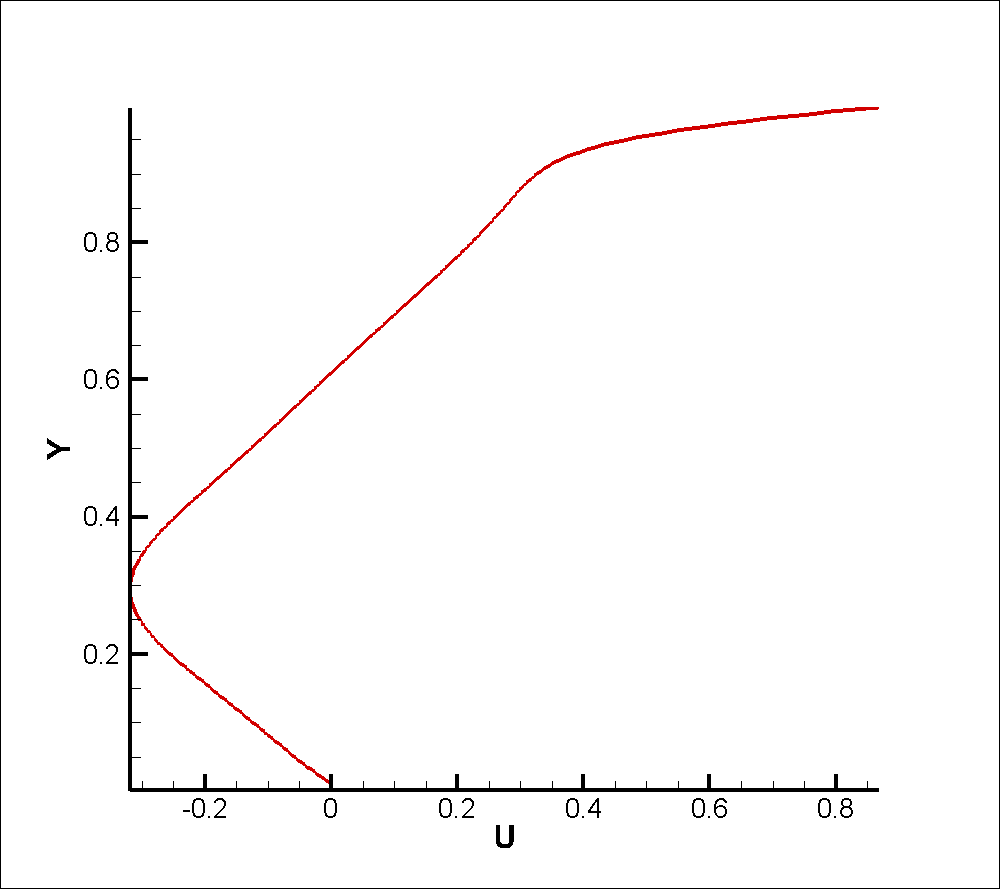
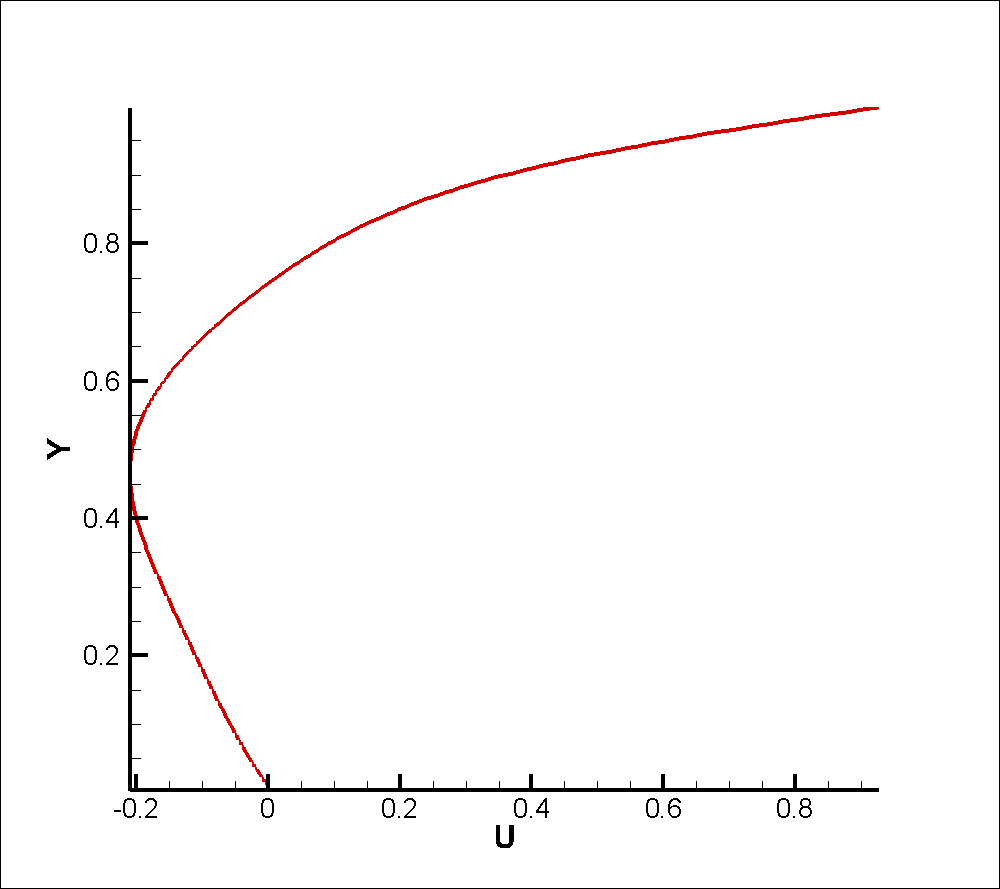


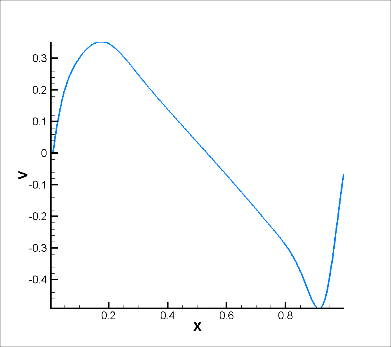
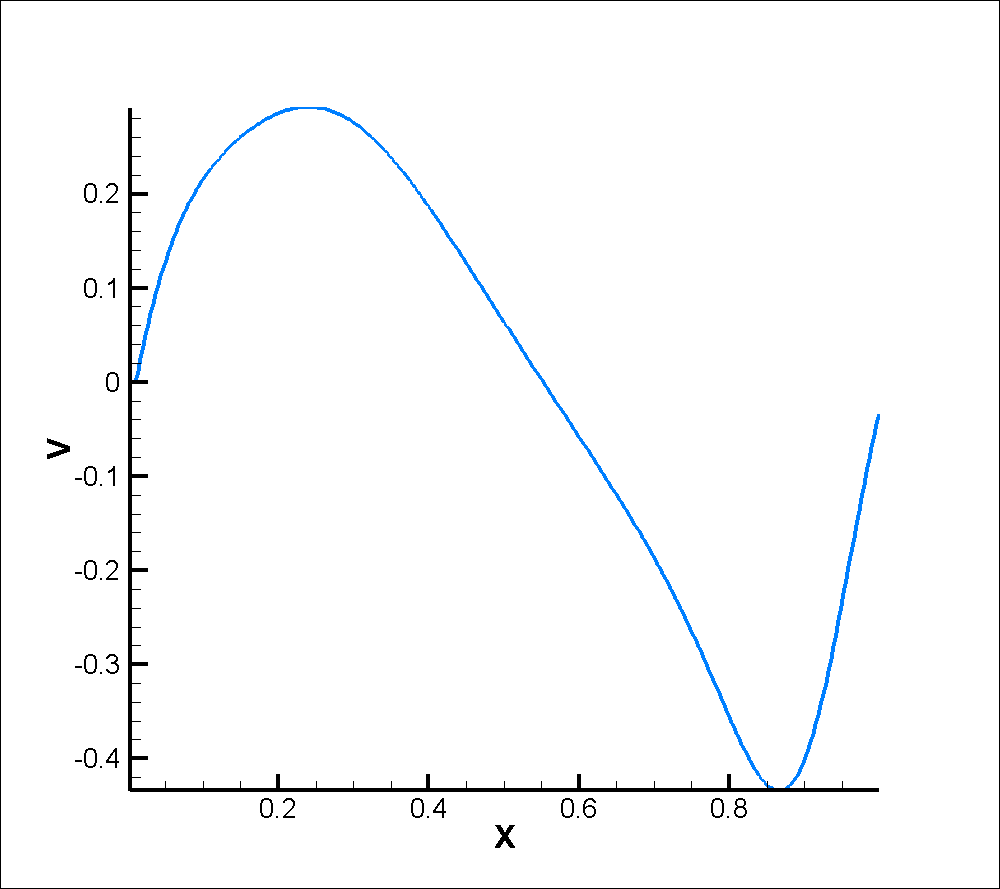
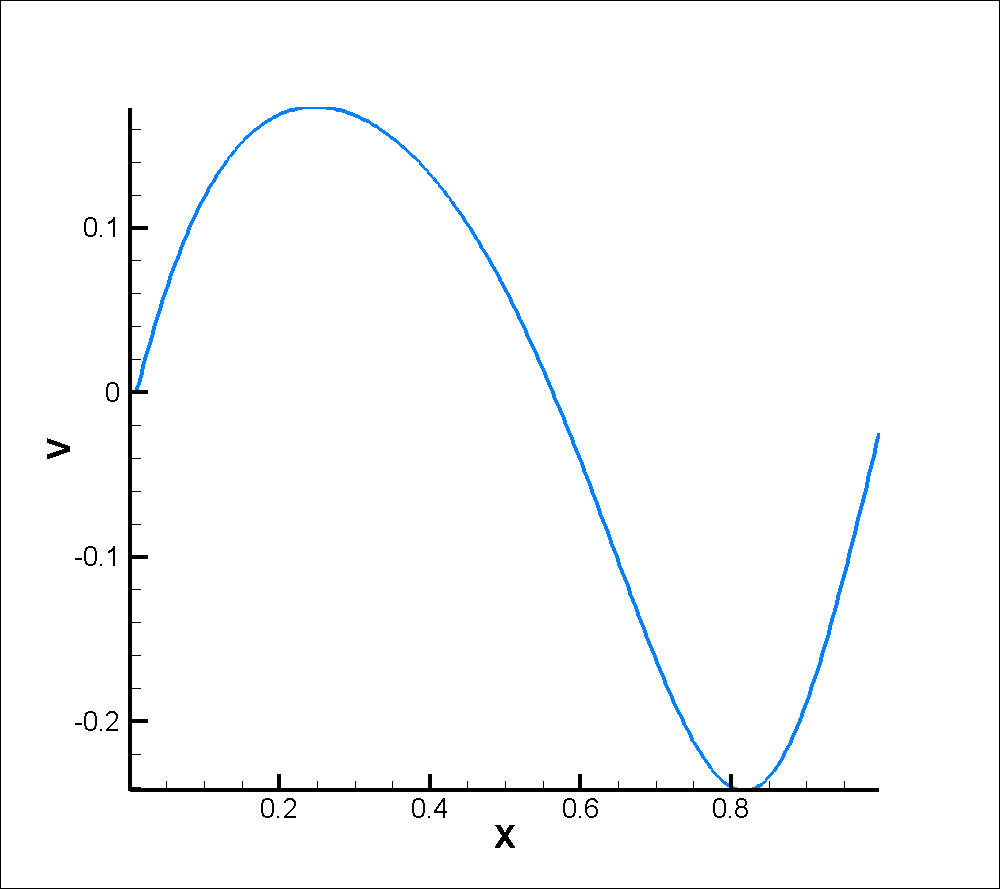
以上从左到右分别是 Re=100400，1000三组雷诺数下的ψ等值线图（上）和ω等值线图（下）。

对 ψ等值线图，可分析得知，随着雷诺数增加，(1)方腔流动中央低ψ区面积增大并且下移。(2)两个角落的流线小漩涡范围变大，即流动分离越明显。(3)随着Re的增大，左上角似乎也 有产生流线漩涡的趋势，但因为顶盖移动，所以这个趋势被覆盖，因此在顶盖附近无明显 流线漩涡产生。

对于ω等值线图，可分析得知，随着雷诺数增加，(1)方腔流动中央高涡量区面积增大并且下移，且左上位置ψ逐渐增大。(2) 两个角落的高涡量区变大，即流动分离越明显。

4、画出沿竖直中心线 x=0.5 上的 u 速度剖面图，和沿水平中心线 y=0.5 上的 υ 速度剖面图

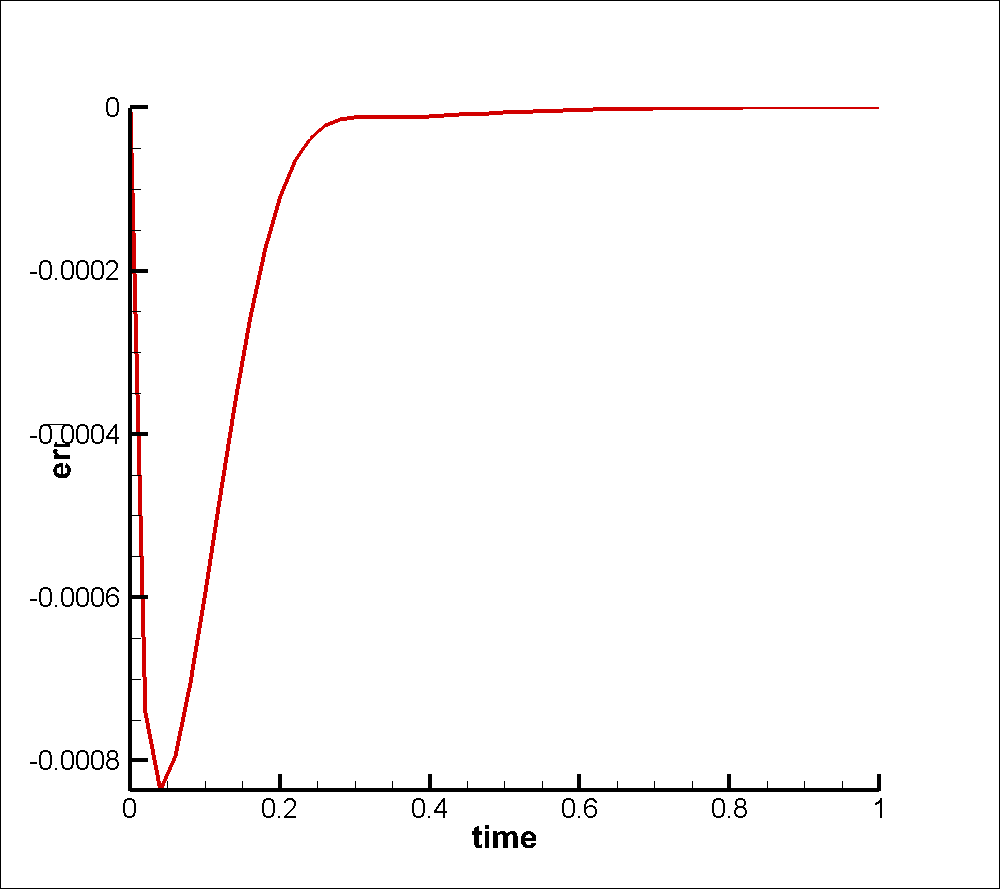
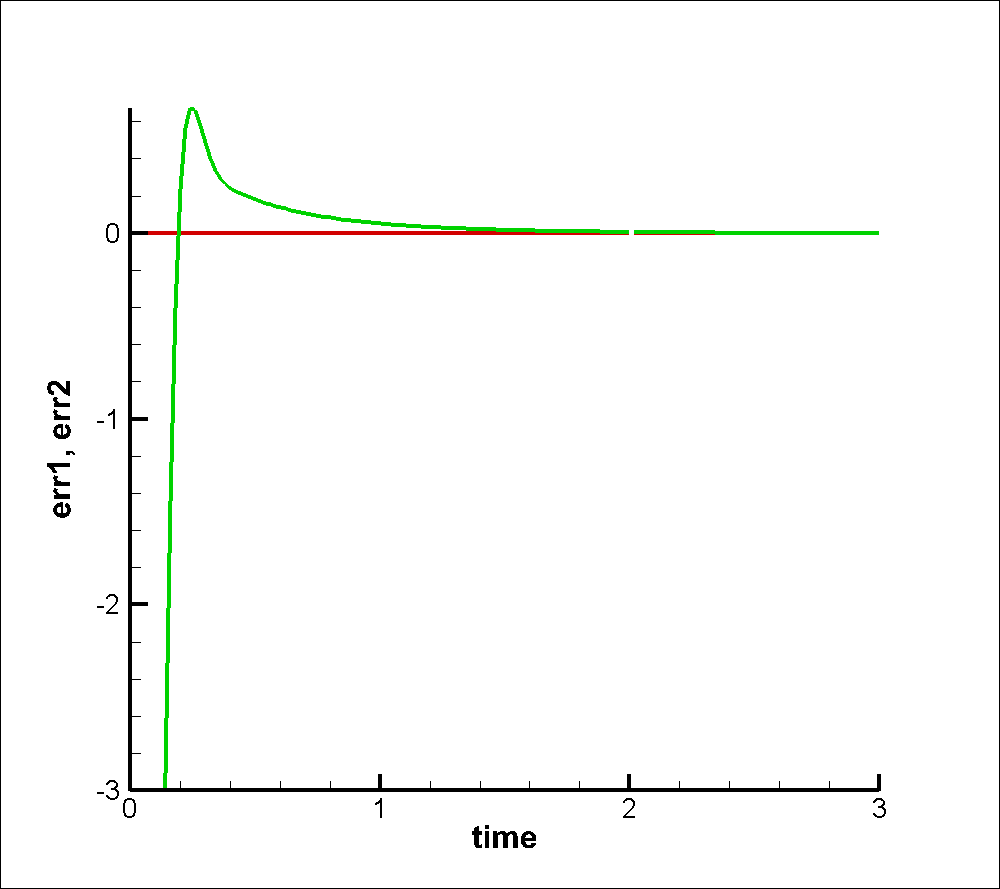




以上从左到右，分别是Re=100，400，1000三组雷诺数下的速度剖面图。其中上方的是在 x=0.5上，y ∈（0，1）这段区间上x方向上速度分量(u)的分布。下方的 是在y=0.5上，x ∈（0，1）这段区间上y方向速度分量(v)的分布。

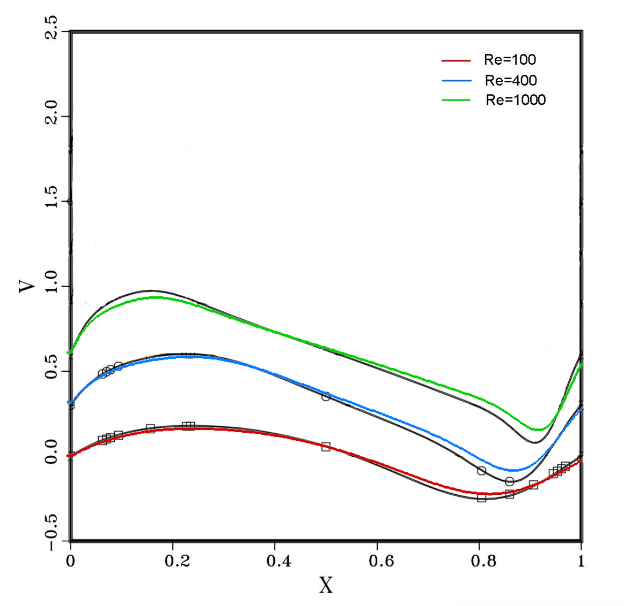
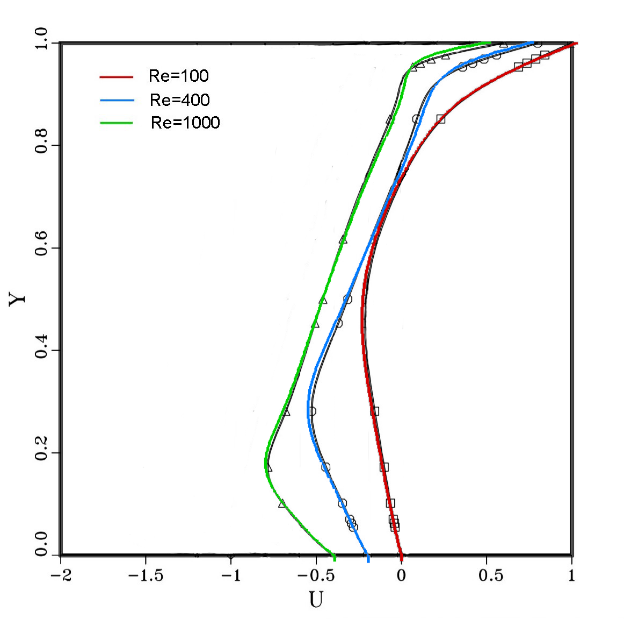
在x=0.5上可见，方腔底部x分量速度为零；中部x分量速度与顶盖移动方向相反且值较小；方腔靠近上部处，x分量速度逐渐接近顶盖速度。随着Re的增大，回升速率出现先慢后快的趋势越来越明显，速度最小值出现点越来越靠近方腔底部。在y=0.5上可见，方腔左侧和右侧速度为零，右侧（顶盖移动方向）速度与顶盖移动方向相反，左侧速度与顶盖移动方向相同。随着Re的增大，速度的幅值越来越大，速度拐角处的变化越来越剧烈，速度最值出现的点向着方腔左右两壁面靠近。

5、画出残差随迭代步数的变化，考察收敛速度

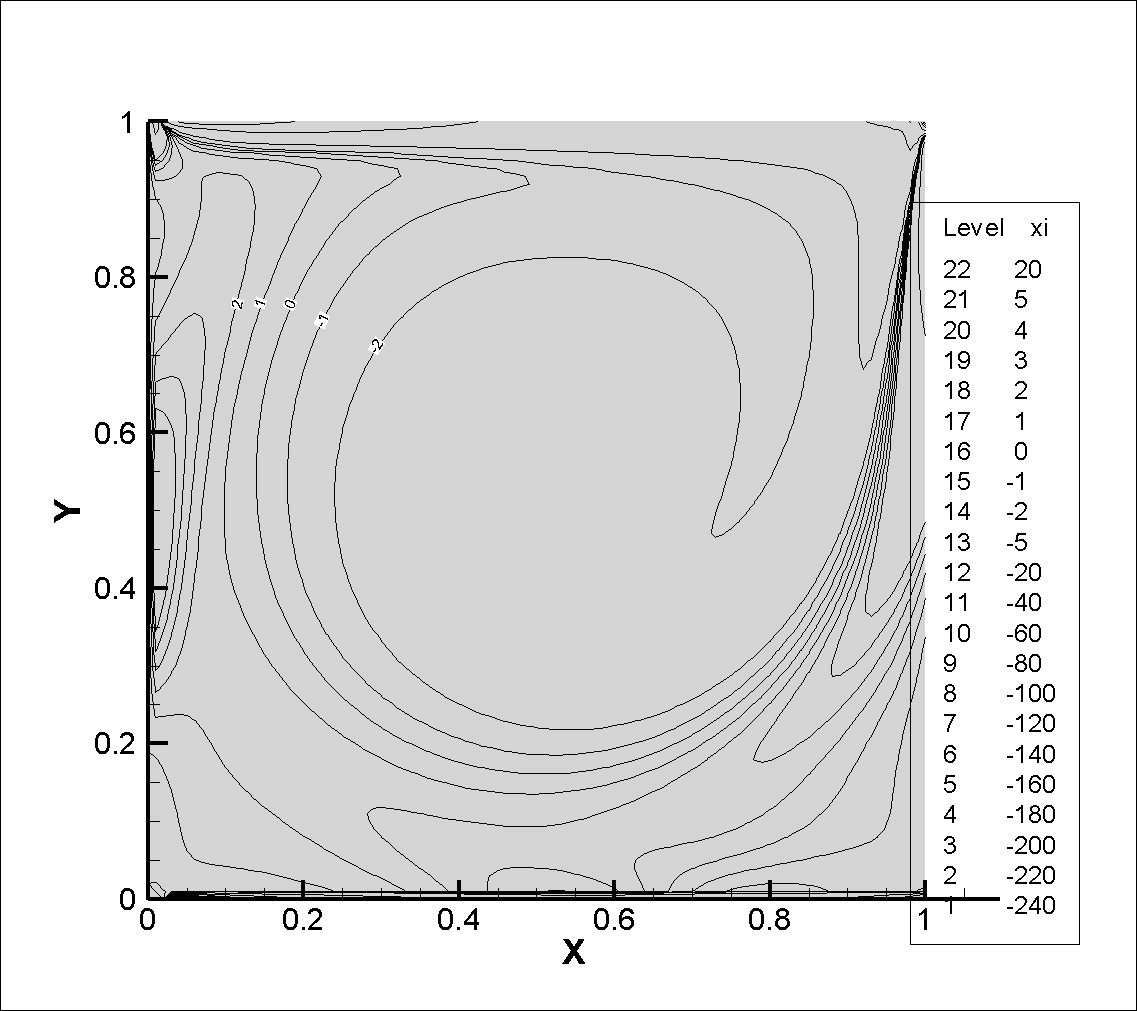
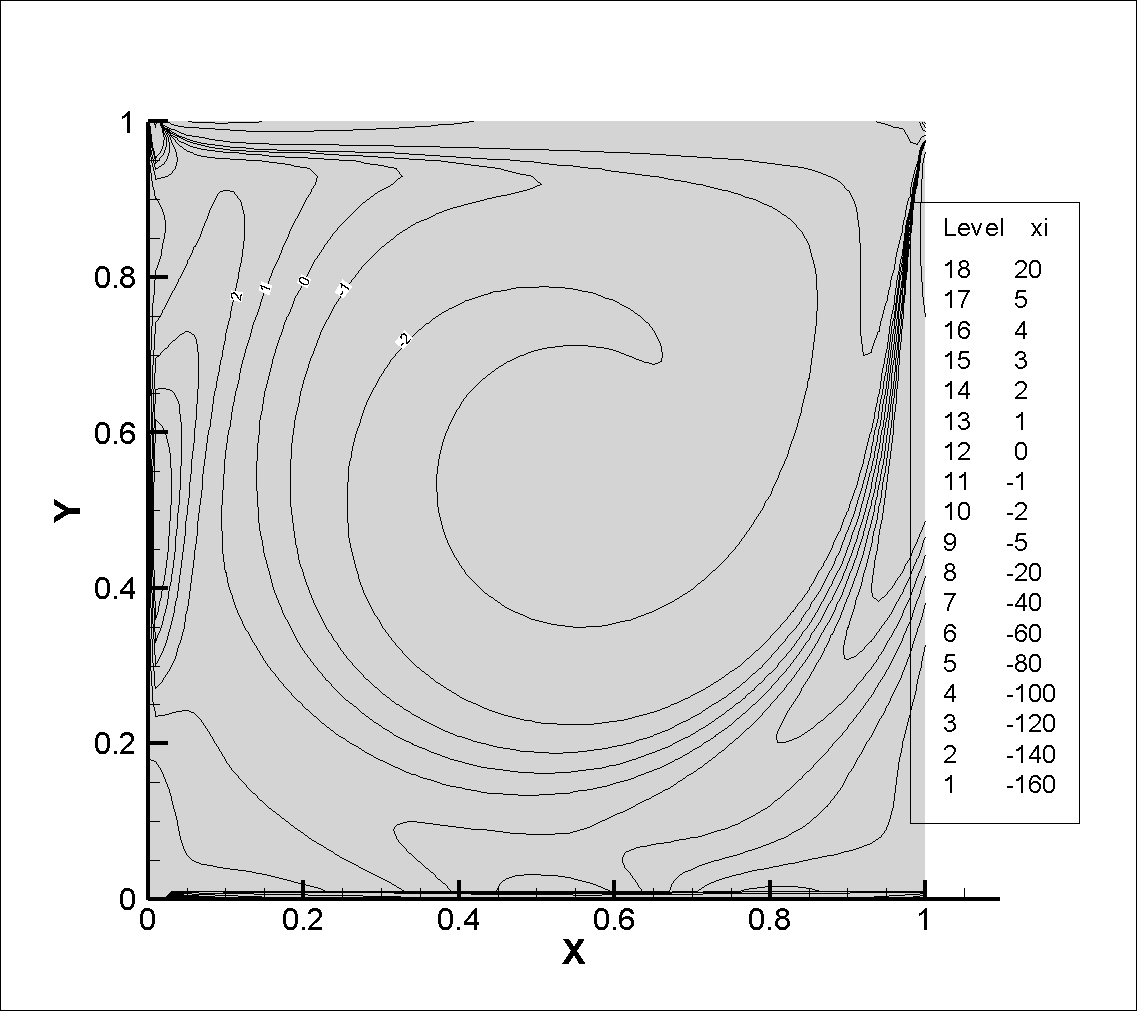


以上为残差随时间变化图，其中横坐标单位为Δt。两个残差最后都明显收敛于0，说明仿真成功。

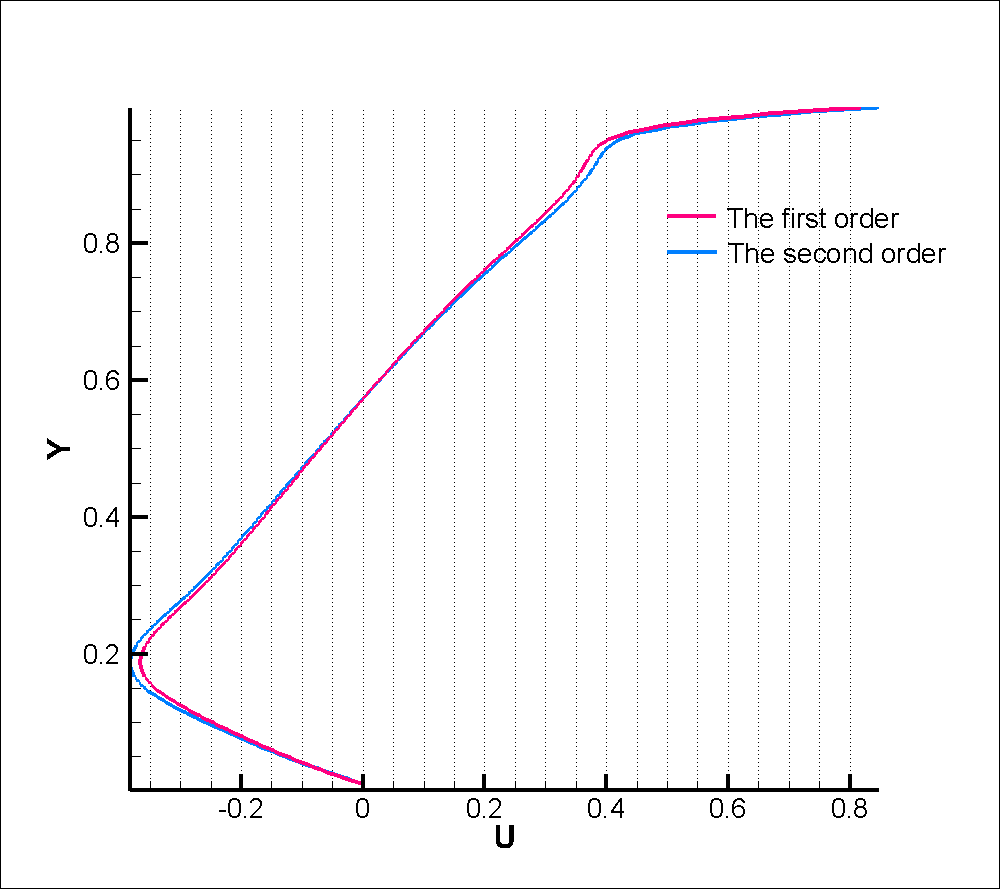
六、拓展实验

1、使用软件截取文献“An Upwind Differencing Scheme for the Incompressible Navier-Stokes Equations, Rogers and Kwak，1988”中图2中相应雷诺数的速度剖面，并将其与自己计算得到的 速度剖面画在同一张图片里面，以检验结果的正确性

黑色线条为文献中原始数据线条，红、蓝、绿三色分别对应雷诺数Re=100、400、1000的数据，左图为x=0.5的速度剖面，右图为y=0.5的速度剖面图。可以看到线条变化趋势相同，数值上也是十分的接近，再一次证明了仿真的正确性。

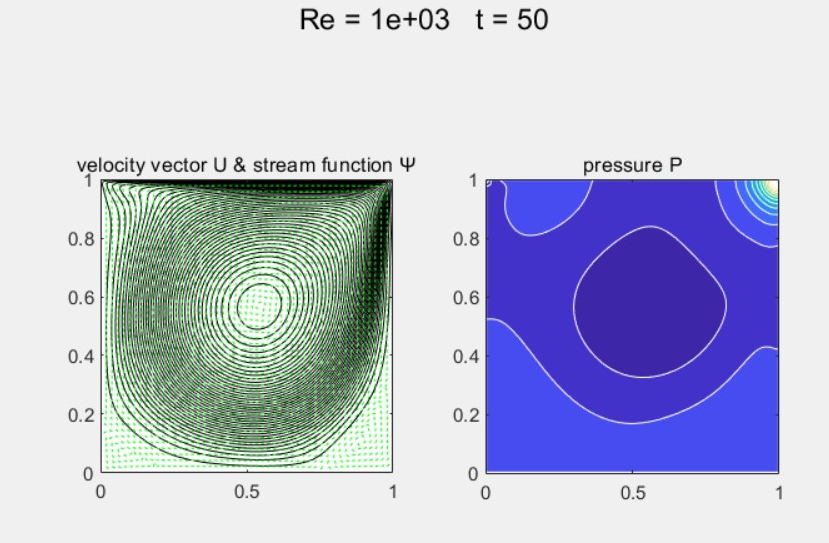
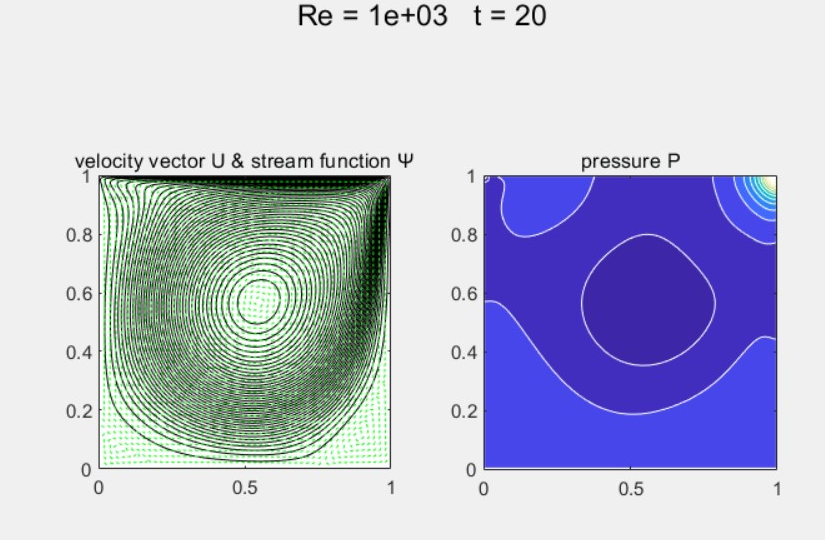
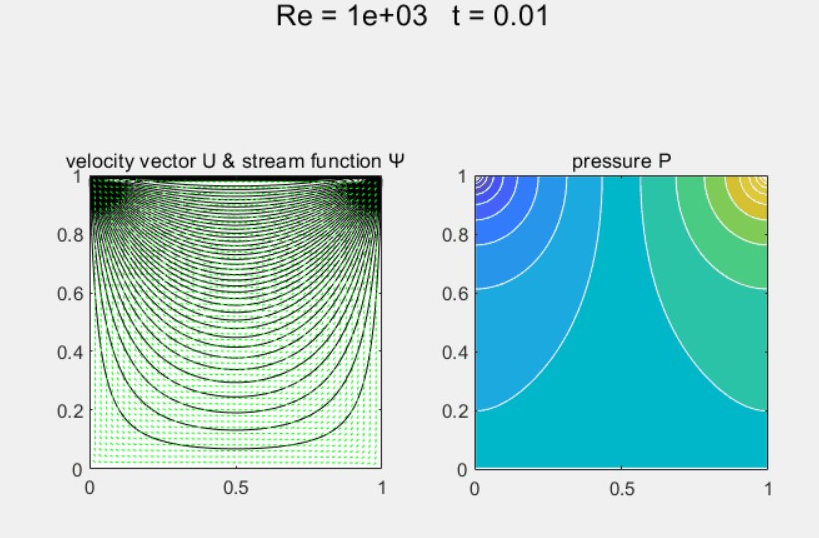
2、比较一阶和二阶涡量边界条件对计算结果的影响

上图左边为一阶涡量边界条件，右边为二阶涡量边界条件，通过等值线可以看出还是有比较明显的区分的，说明不同的边界条件会得到不一样的结果。



上图左边为x=0.5的速度剖面图，右图为y=0.5的速度剖面图，红色线条为一阶涡量边界条件，蓝色线条为二阶涡量边界条件，可以更直观的看出存在的微小差距。

七、与simple方法对照



这是组员用matlab写出的程序，我们粗略地尝试了运行，没有精细地调试最后也得到了一些结果。可以看出，整体流场情况大同小异，证明虽然建模方式不同，但两种方法都能比较准确地进行仿真模拟。这也加深了我们对这两种方法的理解。

八、总结与反思。

总结：本次作业遇到的主要困难是正确迭代循环在编程中的实现以及基本概念的理解。另外Tecplot中取得较好可视化效果亦是一个难点。今后学习中应要更注重实际应用，培养编程能力，而实际应用训练也有助于我们更好地理解课程的理论内容。

本次大作业是我们第一次系统的编写一个计算程序，我们深刻地体会到计算流体力学的复杂和不易。在一开始，我们团队采用simple编写算法，但是因为种种原因一直没有办法输出一个稳定解，我尝试了很多办法检查，一遍又一遍的核查代码，但是都没有办法找出错误。我们一度丧失信心，大家各自找来了涡流函数法的相关文件，一个模块一个模块的了解公式和算法，我们再一次振作起来编写了这段代码，这次我们兵分两路，有同学继续尝试SIMPLE方法解决，另一部分同学尝试用C++编写涡流函数法来解决问题。然而事情没有这么简单，我们的程序又一次出现了各种各样的问题，残差发散，没有出现角涡等等，这些问题让我们深刻的意识到自己的不足。我们不断地在网络上搜寻我们遇到的困难的可能解决办法，一一尝试，在失败中学习，在挫折中成长。经过不断的debug，我们在涡流函数法上取得了突破性进展，能够成功的输出我们需要的所有数据。我们细细数来，这段代码不过只有100行，但是中间给我们留下的思考和经验却是非常令人难忘的。这一次大作业的经历，是我们大学四年中的宝贵财富，受益匪浅就是最好的总结。

九、附源代码

1. #include<iostream>
2. #include<cmath>
3. #include<cstdlib>
4. #include<iomanip>
5. #include<fstream>
6. #include<sstream>
7. #include<string>
8. **using** **namespace** std;

11. **int** rho = 1;
12. **int** Re = 1000;//雷诺数
13. **double** L = 1;//方腔尺寸
14. **const** **int** n = 100;
15. **double** dh = L / n;//detla h
16. **double** dt = 1e-2;//时间步长
17. **double** xi[n + 1][n + 1] = { 0 };
18. **double** psi[n + 1][n + 1] = { 0 };
19. **double** u[n + 1][n + 1], v[n + 1][n + 1];
20. **double** err1, err2, temp;
21. **double** err;
22. **int** i, j;
23. **double** err1\_line[50000], err2\_line[50000];
24. **double** max(**double** a, **double** b) {
25. **if** (a > b)
26. **return** a;
27. **else**
28. **return** b;
29. }
30. **void** output1(**int** m) //输出
31. {
32. ostringstream name;
33. name << "cavity\_" << m << ".dat";
34. ofstream out(name.str().c\_str());
35. out << "Title=\" Lid Driven Flow\"\n"
36. << "VARIABLES=\"X\",\"Y\",\"U\",\"V\",\"psi\",\"xi\"\n"
37. << "ZONE T= \"BOX\", I= "
38. << n + 1 << ", J=" << n + 1 << ", F=POINT" << endl;
39. **for** (j = 0; j <= n; j++)
40. **for** (i = 0; i <= n; i++)
41. {
42. out << **double**(i) / n << " " << **double**(j) / n << " "
43. << u[i][j] << " " << v[i][j] << " " << psi[i][j] << " " << xi[i][j] << endl;
44. }
45. }
47. **void** output2(**double** a[50000],**double** b[50000])   //输出
48. {
49. ostringstream name;
50. name << "err"  << ".dat";
51. ofstream out(name.str().c\_str());
52. out << "Title=\" Lid Driven Flow\"\n"
53. << "VARIABLES=\"time\",\"err1\",\"err2\"\n"
54. << "ZONE T= \"BOX\", I= "
55. << n + 1 << ", J=" << n + 1 << ", F=POINT" << endl;
57. **for** (i = 0; i <= 50000; i++)
58. {
59. out << **double**(i)/50 << " " << a[i]  << " " << b[i] << endl;
60. }
61. }
62. **int** main() {
63. **for** (**int** k = 1; k < 50000; k++) {
64. err = 0;
65. //边界条件
66. **for** (**int** i = 2; i <= n; i++) {
67. xi[i][1] = -2 \* (psi[i][2] ) / (dh \* dh);
68. xi[i][n + 1] = -2 \* (psi[i][n]  + dh) / (dh \* dh);
69. }
70. **for** (**int** j = 2; j <= n; j++) {
71. xi[1][j] = -2 \* (psi[2][j]) / (dh \* dh);
72. xi[n + 1][j] = -2 \* (psi[n][j] ) / (dh \* dh);
73. }
74. //控制方程
75. **for** (i = 2; i <= n; i++) {
76. **for** (j = 2; j <= n; j++) {
77. u[i][j] = (psi[i][j + 1] - psi[i][j - 1]) / (2 \* dh);
78. v[i][j] = -(psi[i + 1][j] - psi[i - 1][j]) / (2 \* dh);
79. err1 = (psi[i + 1][j] + psi[i - 1][j] + psi[i][j + 1] + psi[i][j - 1] + xi[i][j] \* dh \* dh) / 4 - psi[i][j];
80. psi[i][j] = psi[i][j] + rho \* err1;
81. err2 = dt \* (-dh / 2 \* (u[i][j] \* (xi[i + 1][j] - xi[i - 1][j]) + v[i][j] \* (xi[i][j + 1] - xi[i][j - 1])) + (xi[i + 1][j] + xi[i - 1][j] + xi[i][j + 1] + xi[i][j - 1] - 4 \* xi[i][j]) / Re) / (dh \* dh);
82. xi[i][j] = xi[i][j] + rho \* err2;
83. temp = max(abs(err1), abs(err2));
84. **if** (err < temp)
85. err = temp;
86. }
87. }
88. **if** (k % 1000 == 0) {
89. cout << "k=" << k << "  " << "err=" << err << "   " << err1 << "   " << err2 << endl;
90. output1(k);
91. }
92. err1\_line[k - 1] = err1;
93. err2\_line[k - 1] = err2;
94. **if** (err < 1e-6)
95. **break**;
96. }
98. output2(err1\_line,err2\_line);
99. **return** 0;
100. }