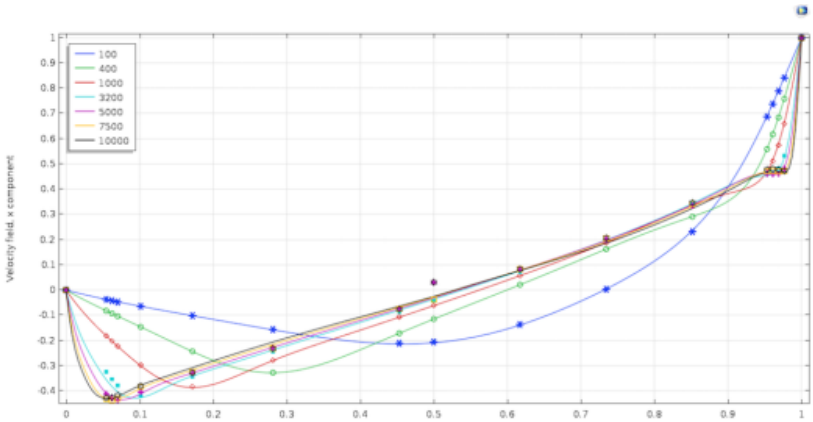
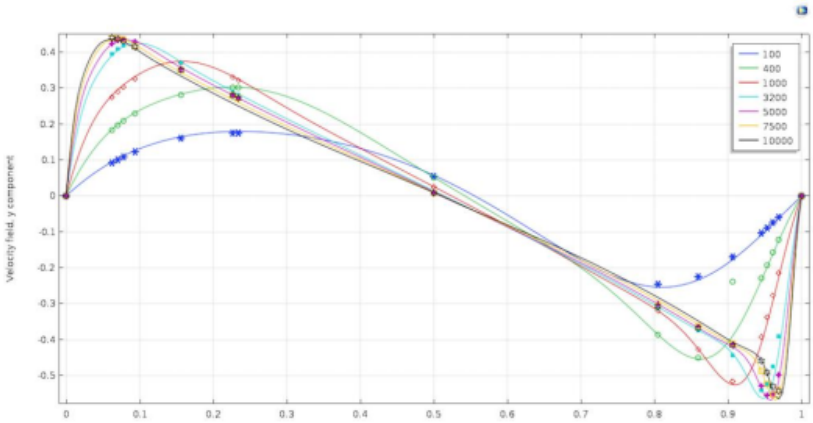
2 作业 2

2.1 （1）的问题描述

修改计算程序实现对 Re=100，400，1000 的方腔流动问题的数值模拟（网格数目：100*100）；流动发展稳定后，分别画出压力、速度分量云图；统计空腔中心线上的速度分量分布，并与图2.1所示结果进行对比。边界条件设置参考图2.2。



（a）沿垂直中心线速度 u 分量（X 方向速度分量）



（b）沿水平中心线的速度 v 分量（Y 方向速度分量）

图 2.1: 方腔流动问题的数值模拟结果

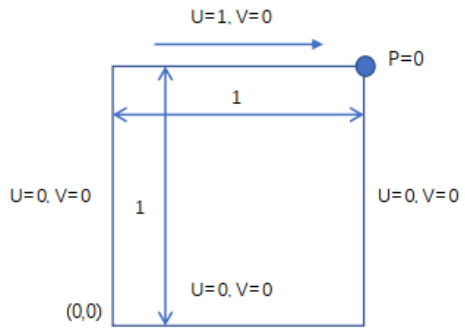


图 2.2: 方腔流动问题的数值模拟边界条件

2.1.1 程序修改

①首先修改所给示例程序的流动区域大小。改为边长为 1 的正方形。

②根据图2.2的边界条件，即

$$U|_{x=0} = U|_{x=1} = U|_{y=0} = 0, U|_{y=1} = 1$$

$$V|_{x=0} = V|_{x=1} = V|_{y=0} = V|_{y=1} = 0$$

$$\frac{\partial p}{\partial x}|_{x=0} = \frac{\partial p}{\partial x}|_{x=1} = \frac{\partial p}{\partial y}|_{y=0} = \frac{\partial p}{\partial y}|_{y=1} = 0$$

修改程序中的边界条件设定如下：

```
! Type: 1-Dirichlet; 0-Neumman
!c|-----------------------|
!c|         -4-          |
!c|                       |
!c|-1-                 -3-|
!c|                       |
!c|         -2-          |
!c|-----------------------|
          U_type = (/ 1, 1, 1, 1 /)
          V_type = (/ 1, 1, 1, 1 /)
          P_type = (/ 0, 0, 0, 0 /)
          U_valu = (/0.,0.,0., 1./)
          V_valu = (/0.,0.,0., 0./)
          P_valu = (/0.,0.,0., 0./)
```

③取 **nx=100，ny=100**，形成 100*100 的网格。

④已知雷诺数 $Re = \frac{\rho UL}{\mu}$。假设流体密度 $\rho = 1$，流速 U=1，长度 L=1，只需要改变 μ 即可改变 Re 的值。我们取 $\mu = 0.001, 0.0025, 0.01$，即对应 $Re = 1000, 400, 100$。

⑤此外，所给示例程序带有一个 imersed cube。

```
          cube_xmin = 0.4
          cube_xmax = 0.6
          cube_ymin =-0.1
          cube_ymax = 0.1
```

方腔边长为 1，cube 在方腔中心，cube 的边长为 0.2。如图。

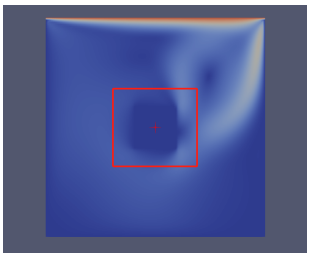


图 2.3: CUBE 示意图

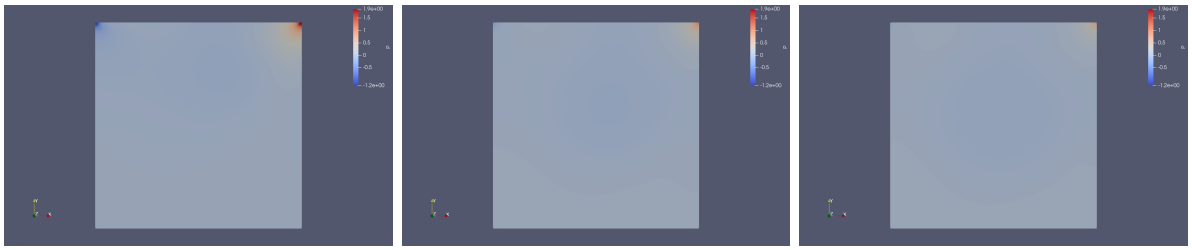
本次大作业中并没有这个 cube，故将 *Cart_bc.F90* 和 *Cart_ini.F90* 中有关 cube 的部分**注释掉**：

```
!     if ( xi>=cube_xmin .and. xi<=cube_xmax .and. &
!          yi>=cube_ymin .and. yi<=cube_ymax) then
```

```
3       !    Uflux(i,j) = 0.
4       !    endif
5       !    if  ( xi>=cube_xmin .and. xi<=cube_xmax .and. &
6       !            yi>=cube_ymin .and. yi<=cube_ymax) then
7       !    Vflux(i,j) = 0.
8       !    endif
9
10  ! immersed object
11  !        cube_xmin = 0.4
12  !        cube_xmax = 0.6
13  !        cube_ymin =-0.1
14  !        cube_ymax = 0.1
```

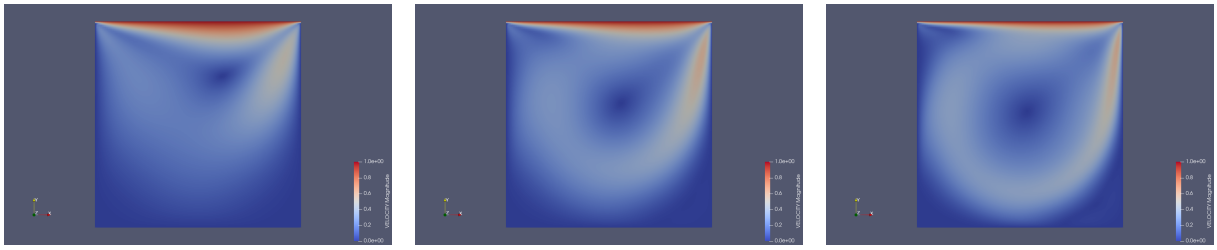
2.1.2 压力和速度分布云图

修改后不同雷诺数下的压力和速度分布云图如下：



(a)Re=100　　(b)Re=400　　(c)Re=1000

图 2.4: 压力分布云图



(a)Re=100　　(b)Re=400　　(c)Re=1000

图 2.5: 速度分布云图

采用红蓝色表绘制空腔内的压力和速度大小。从图 2.4 看到，空腔整体压力基本为 0，左上角为压力极小值处，右上角为压力极大值处，这是由于顶盖移动所驱动的。顶部的流体被顶盖向右带动后，左上角出现了空缺，聚集到了右上角，形成了两处压力极值。当雷诺数较低时，例如等于 100（左图）时，由于粘性项较大而造成的能量损耗，压力极值的数值较大。当雷诺数增加到 1000（右图）后，压力极值的数值减小。

从图 2.5 看到，空腔顶部的速度接近于 U =1，此处的流体流动是由移动壁驱动的。流体被推向右侧的壁后，先向下流动，再回到腔体左侧。运动在空腔中心产生了一个大型涡流。图片显示，当雷诺数较低，例如等于 100 时（左图），由于粘性项较大而造成的能量损耗，空腔中心的速度较小，涡也不明显。雷诺数增加到 1000 后（右图），空腔内的速度加快，涡流明显扩展到了空腔底部。

2.1.3 空腔中心线上的速度分量分布

使用 paraview 里的 plot over line 功能，可以画出不同雷诺数下的空腔中心线上的速度分量分布如图 2.7 和图 2.7，可以看出在 Re=100-1000 的范围内，本文的计算结果与图2.1结果非常相近。

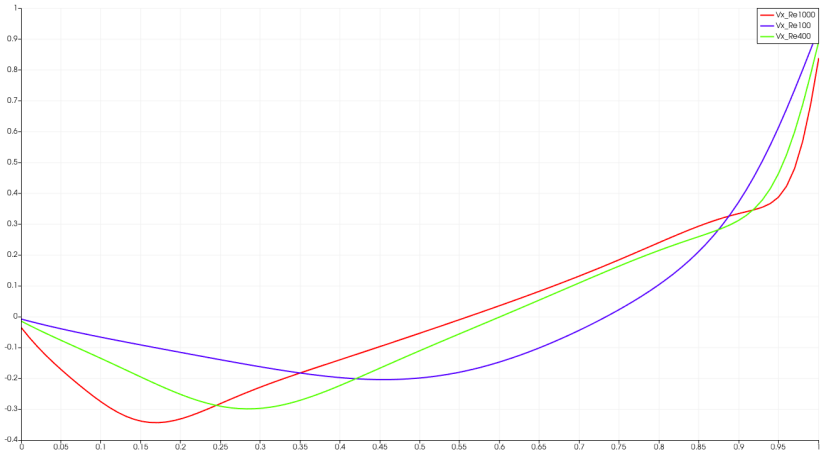


图 2.6: 沿垂直中心线速度 u 分量（X 方向速度分量）

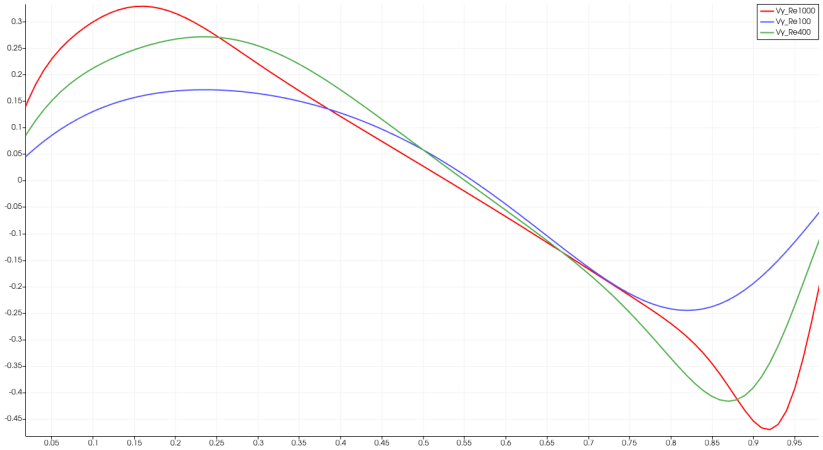


图 2.7: 沿水平中心线的速度 v 分量（Y 方向速度分量）

2.2 （2）的问题描述

采用不同的对流项数值离散格式（如：1 阶 UPWIND、2 阶 QUICK 等）、不同的网格数计算 Re=1000 的方腔流动问题，分析离散格式和网格数对模拟结果的影响规律。

2.2.1 程序修改

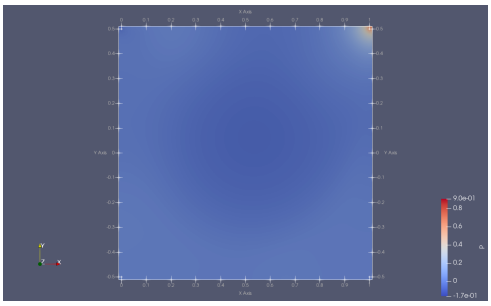
我们选择了两种离散格式：Adv_UV_WENO5 和 Adv_UV_QUICK；

```
1       ! * - Reconstruct face val -
2       call Adv_UV_WENO5  !Hybrid  !!Adv_UV_QUICK  !
```

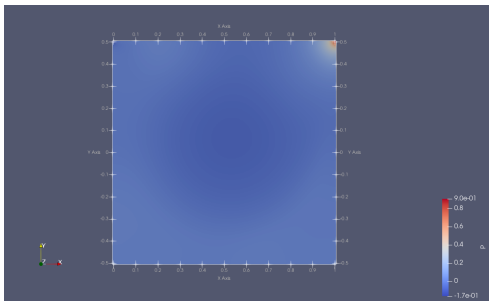
两种网格数：100*100 和 200*200。雷诺数 Re=1000.

2.2.2 压力和速度分布云图

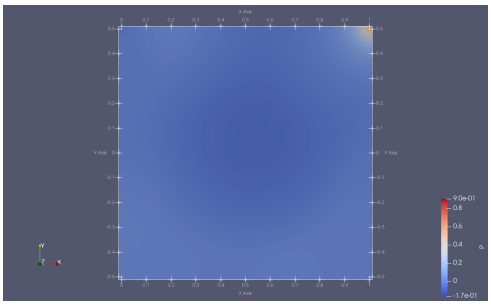
画出这四种对流项数值离散格式的压力和速度分布云图。可以看出，网格划分更密的对流格式计算出的云图更精细，计算结果更准确。



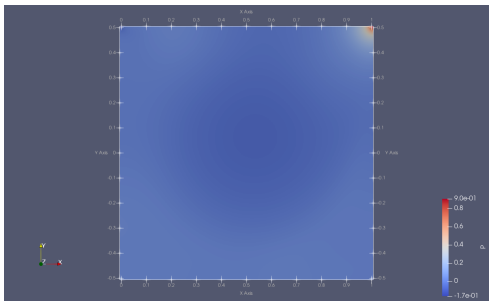
(a)QUICK100*100



(b)QUICK200*200

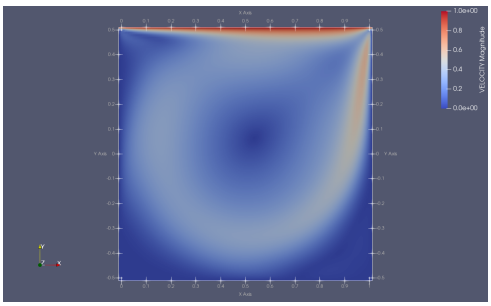


(c)WENO100*100

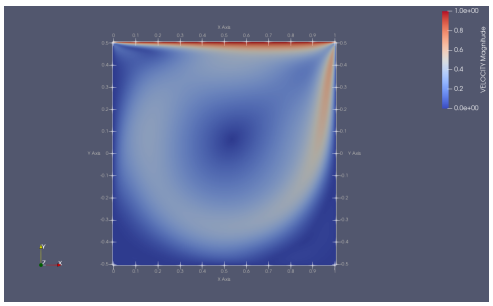


(d)WENO200*200

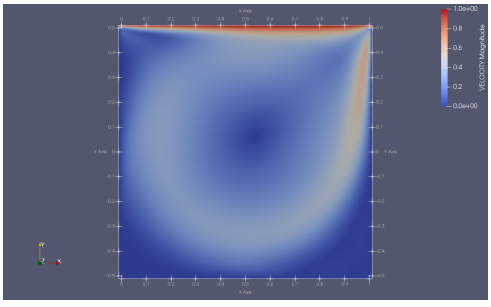
图 2.8: 不同离散格式和网格数的压力分布云图



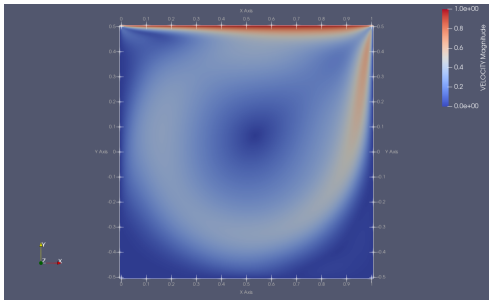
(a)QUICK100*100



(b)QUICK200*200



(c)WENO100*100



(d)WENO200*200

图 2.9: 不同离散格式和网格数的速度分布云图

2.2.3 空腔中心线上的速度分量分布

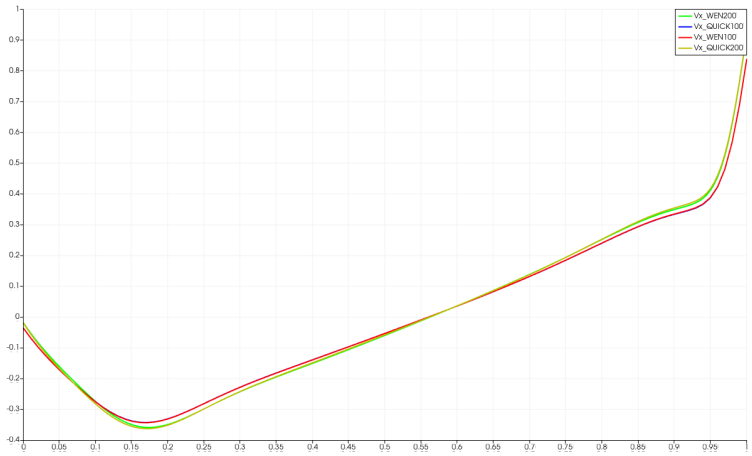


图 2.10: 沿垂直中心线速度 u 分量（X 方向速度分量）

图 2.11: 沿水平中心线的速度 v 分量（Y 方向速度分量）

可以看出，更细的网格计算出来的速度分布范围更大，比如说速度为正时，200*200 的速度的绝对值会大于 100*100 的，速度为负时，200*200 速度的绝对值也大于 100*100 的。这是因为网格划分越精密，计算出来的结果就越准确。

QUICK 格式和 WENO5 格式计算出来结果比较近似。

附录

A

```
import time
start_time = time.time()

def read_elements(file_path):
    with open(file_path, 'r') as file:
        elements = []
```

```
7               for line in file:
8                   parts = line.strip().split()
9                   if parts and parts[0].isdigit():
10                      elements.append(list(map(int, parts)
                            ))
11          return elements
12
13
14      def construct_node_element_map(elements):
15          node_to_elements = {}
16
17          for element_index, element in enumerate(elements
                ):
18              node_count = element[0]
19              node_ids = element[1:1 + node_count]
20
21              for node_id in node_ids:
22                  if node_id not in node_to_elements:
23                      node_to_elements[node_id] = []
24                  node_to_elements[node_id].append(
                        element_index)
25
26          return node_to_elements
27
28
29      def output_node_element_map(node_to_elements,
            output_file):
30          with open(output_file, 'w') as file:
31              for node_id in sorted(node_to_elements.keys
                    ()):
32                  element_list = node_to_elements[node_id]
33                  file.write(f"{node_id}{len(element_list)
                        }␣{''.join(map(str,element_list))}\n"
                        )
34
35
36      # 文件路径
37      elements_file_path = 'Elements.txt'
38      output_file_path = 'result1.txt'
39
40      # 读取数据
41      elements = read_elements(elements_file_path)
42
43      # 构建节点到单元的映射
44      node_to_elements = construct_node_element_map(
            elements)
45
46      # 输出结果到文件
```

```
47      output_node_element_map(node_to_elements,
            output_file_path)
48      end_time = time.time()
49      print("运行时间:{:.6f}秒".format(end_time -
            start_time))
```

B

```
1       from collections import defaultdict
2       import time
3       start_time = time.time()
4       def read_elements(file_path):
5           with open(file_path, 'r') as file:
6               elements = []
7               for line in file:
8                   parts = line.strip().split()
9                   if parts and parts[0].isdigit():
10                      elements.append(list(map(int, parts)
                            ))
11          return elements
12
13
14      def construct_mappings(elements):
15          element_to_edges = {}
16          edge_to_elements = defaultdict(list)
17
18          for element_index, element in enumerate(elements
                ):
19              node_count = element[0]
20              node_ids = element[1:1 + node_count]
21
22              # Create edges for the element
23              edges = []
24              for i in range(node_count):
25                  edge = tuple(sorted((node_ids[i],
                        node_ids[(i + 1) % node_count])))
26                  edges.append(edge)
27                  edge_to_elements[edge].append(
                        element_index)
28
29              element_to_edges[element_index] = edges
30
31          return element_to_edges, edge_to_elements
32
33
```

```
34      def find_adjacent_elements(element_to_edges,
            edge_to_elements):
35          element_adjacency = {i: set() for i in
                element_to_edges}
36
37          for element, edges in element_to_edges.items():
38              adjacent_elements = set()
39
40              for edge in edges:
41                  adjacent_elements.update(
                        edge_to_elements[edge])
42
43              # Remove the element itself from its
                    adjacency list
44              adjacent_elements.discard(element)
45
46              element_adjacency[element] =
                    adjacent_elements
47
48          return element_adjacency
49
50
51      def output_adjacent_elements(element_adjacency,
            output_file):
52          with open(output_file, 'w') as file:
53              for element, adjacent_elements in
                    element_adjacency.items():
54                  file.write(f"{element}{len(
                        adjacent_elements)}{''.join(map(str,
                        adjacent_elements))}\n")
55
56
57      # 文件路径
58      nodes_file_path = 'Nodes.txt'
59      elements_file_path = 'Elements.txt'
60      output_file_path = 'result2.txt'
61
62
63      elements = read_elements(elements_file_path)
64
65      # 构建单元和节点的映射
66      element_to_nodes, node_to_elements =
            construct_mappings(elements)
67
68      # 找到相邻单元
69      element_adjacency = find_adjacent_elements(
            element_to_nodes, node_to_elements)
70
```

```
71      # 输出结果到文件
72      output_adjacent_elements(element_adjacency,
            output_file_path)
73      end_time = time.time()
74      print("运行时间:{:.6f}秒".format(end_time -
            start_time))
```