CFD Assignment 3 Lid Driven Cavity

110033640 Yi-En Chou

January 14, 2022

1 Problem Description

The present task is to provide numerical solution of a steady flow within a square cavity, where the top lid is moving to the right at a constant velocity. The following figure shows a sample solution for Re=100 with central difference scheme. The Reynolds number is defined as

$$Re = \frac{\rho U_{lid} L}{\mu}$$

, where U_{lid} is the top lid velocity (1m/s) and L is the cavity height. The boundary conditions are, shown in Figure.

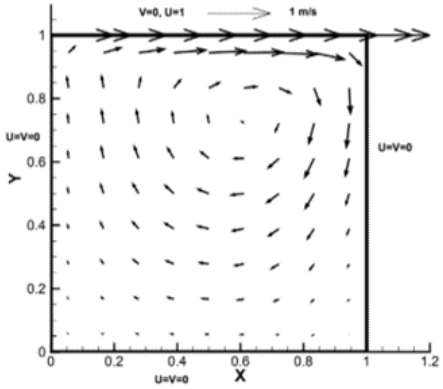


Figure 1: Boundary Condition

The flow field can be solved by the continuity and Navier-Stokes equations (ρ = constant):

$$\frac{\partial \rho U}{\partial x} + \frac{\partial \rho V}{\partial y} = 0$$

$$\frac{\partial \rho U}{\partial t} + \frac{\partial \rho UU}{\partial x} + \frac{\partial \rho VU}{\partial x} = -\frac{\partial P}{\partial x} + \mu(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})$$

$$\frac{\partial \rho V}{\partial t} + \frac{\partial \rho UV}{\partial x} + \frac{\partial \rho VV}{\partial x} = -\frac{\partial P}{\partial y} + \mu(\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2})$$

2 Introduction of methodology adopted

2.1 Computational method

Instead of Finite Difference Method(FDM), Finite Volume method(FVM) is the computational method implemented in this task, for it being a better approach in the aspect of conservation.

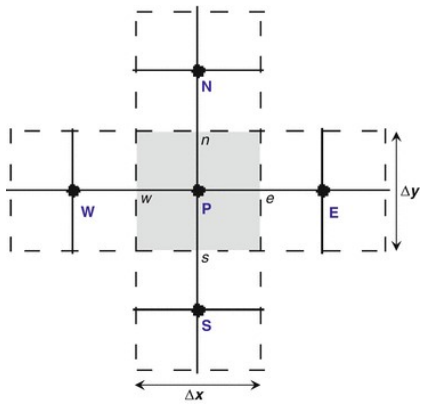


Figure 2: Finite Volume method(FVM)

2.2 Grid Arrangement

In this assignment, staggered grid is applied , aiming to deal with checkerboard distribution arise from difference equation. With staggered grid applied, nodes in the neighbor share the same set of input, where this results in a better approximation.

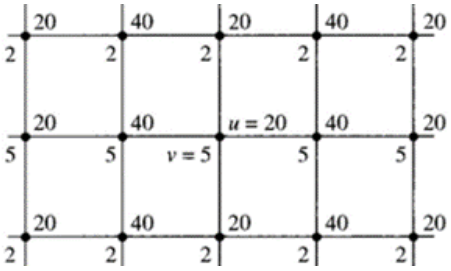


Figure 3: Checkerboard Distribution

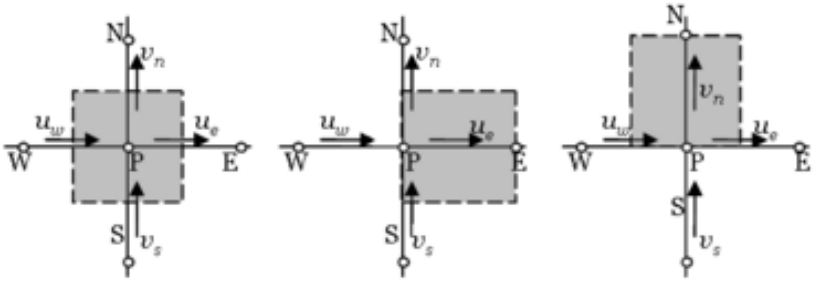


Figure 4: Staggered Grid

2.3 Boundary treatment

For this task, the boundary condition is set that, for the velocity, if the location of node is originally on the actual boundary, the velocity is set 0, else the velocity is set a value that its average with the second outer layer becomes 0 at wall, and 1 at the lid; for the pressure, it is set equal to the second outer layer so that it correlates the physical world.

2.4 Velocity and Pressure coupling

In the coupling of velocity and pressure, Semi-Implicit Method for Pressure Linked Equations(SIMPLE) is adopted. It is a widely used numerical procedure to solve the Navier–Stokes equations. In this algorithm, Partankar's assumption is introduced, where the velocity correction term is assumed to be proportion to the pressure term.

$$u'_P \propto -\frac{(P'_{i,j} - P'_{i,j+1})\Delta y}{A^u_P}$$

$$v'_P \propto -\frac{(P'_{i,j} - P'_{i+1,j})\Delta x}{A^v_P}$$

Firstly, we first solve the pseudo velocity; secondly, solve the pressure correction; thirdly, we further obtain the velocity correction from the pressure correction. and then add the velocity correction to the pseudo velocity to get the solution of velocity for the iteration. After an iteration, the divergence is our criterion of convergence, if the residual is less than the tolerance we set, than it is perceive as our final results.The procedure of the SIMPLE algorithm is as shown below.

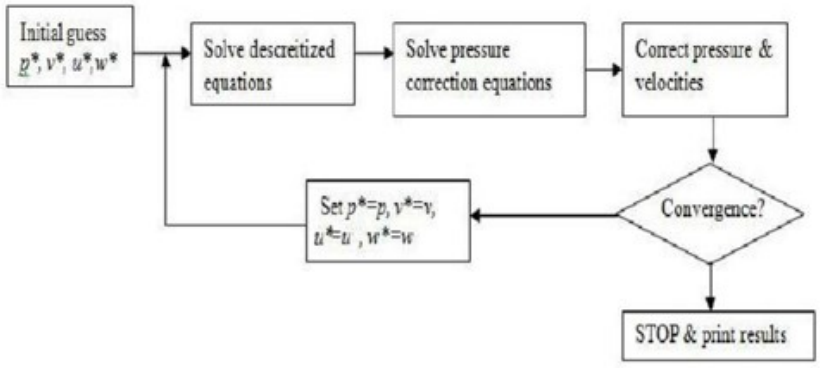


Figure 5: Flow-chart of SIMPLE algorithm

2.5 Pseudo velocity

2.5.1 x direction

By integrating the Navier Stokes equation and an assumption of steady state, we get,

$$\int_s^n \int_w^e [\frac{\partial uu}{\partial x} + \frac{\partial uv}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2})]\, dx\, dy$$

the equation can be further rewritten into

$$u^* = \frac{a_E u_E + a_W u_W + a_N u_N + a_S u_S + h(P_{i,j} - P_{i,j+1})}{a_u}$$

where

$$a_E = -0.5 \times u_E \times h + \nu; \quad a_W = 0.5 \times u_W \times h + \nu$$

$$a_N = -0.5 \times v_N \times h + \nu; \quad a_S = 0.5 \times u_S \times h + \nu$$

$$a_u = 0.5 \times u_E \times h - 0.5 \times u_W \times h + 0.5 \times v_N \times h - 0.5 \times u_S \times h + 4\nu$$

and the velocity at the cell face will be later determined by the implementation of flux-limiter, introduced in the following section.

2.5.2 y direction

By integrating the Navier Stokes equation and an assumption of steady state, we get,

$$\int_s^n \int_w^e [\frac{\partial vu}{\partial x} + \frac{\partial vv}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2})]\, dx\, dy$$

the equation can be further rewritten into

$$v^* = \frac{a_E v_E + a_W v_W + a_N v_N + a_S v_S + h(P_{i,j} - P_{i+1,j})}{a_v}$$

where

$$a_E = -0.5 \times u_E \times h + \nu; \quad a_W = 0.5 \times u_W \times h + \nu$$

$$a_N = -0.5 \times v_N \times h + \nu; \quad a_S = 0.5 \times u_S \times h + \nu$$

$$a_v = 0.5 \times u_E \times h - 0.5 \times u_W \times h + 0.5 \times v_N \times h - 0.5 \times u_S \times h + 4\nu$$

and the velocity at the cell face will be later determined by the implementation of flux-limiter, introduced in the following section.

2.6 Pressure Correction

The relation between of velocity correction and pressure correction is established based on the following two equation:

$$u = \frac{\sum a_{nb} u_{nb} + h(P_{i,j} - P_{i,j+1})}{a_u}$$

$$u^* = \frac{\sum a_{nb} u^*_{nb} + h(P_{i,j} - P_{i,j+1})}{a_u}$$

$$v = \frac{\sum a_{nb} v_{nb} + h(P_{i,j} - P_{i+1,j})}{a_u}$$

$$v^* = \frac{\sum a_{nb} v^*_{nb} + h(P_{i,j} - P_{i+1,j})}{a_u}$$

and since

$$u' = u - u^*$$

$$v' = v - v^*$$

with Partankar's assumption, we then obtain the relation between of velocity correction

$$u' = \frac{\Delta y}{a_u}(P'_{i,j} - P'_{i,j+1})$$

$$v' = \frac{\Delta x}{a_v}(P'_{i+1,j} - P'_{i,j})$$

We then derive the equation for pressure correction through continuity

$$\int_s^n \int_w^e (\frac{\delta u}{\delta x} + \frac{\delta v}{\delta y})\, dx\, dy = 0$$

after the integration, the equation becomes

$$(u_e - u_w)\Delta y + (v_n - v_s)\Delta x = 0$$

$$\rightarrow ((u_e^* + u_e') - (u_w^* + u_w'))\Delta y + ((v_n^* + v_n') - (v_s^* + v_s'))\Delta x = 0$$

$$\rightarrow (u_e' - u_w')\Delta y + (v_n' - v_s')\Delta x + b = 0, \qquad b = (u_e^* - u_w^*)\Delta y + (v_n^* - v_s^*)\Delta x$$

by substituting the equation with the previous derived relation of pressure correction and velocity correction, we obtain

$$P' = \frac{\sum a_{nb} P_{nb} + b}{a_p}$$

where

$$a_E = \frac{h^2}{a_{u,e}}; \quad a_W = \frac{h^2}{a_{u,w}}; \quad a_N = \frac{h^2}{a_{v,n}}; \quad a_S = \frac{h^2}{a_{v,s}};$$

$$a_P = a_E + a_W + a_N + a_S$$

2.7 Velocity Correction

The velocity correction is determined by pressure correction, which was derived previously:

$$u' = \frac{\Delta y}{a_u}(P'_{i,j} - P'_{i,j+1})$$

$$v' = \frac{\Delta x}{a_v}(P'_{i+1,j} - P'_{i,j})$$

2.8 Convergence Criterion

The criterion of convergence is the reach of divergence free, if the summation of the divergence of flow rate of every cell is lower then 10^{-9}, then it is regard successfully reaching the convergence criterion.

2.9 Scheme for cell wall

To more generally apply schemes into our computation, the flux-limiter operator (ϕ) is implemented, where ϕ is the ratio of the slope between the cell wall and its previous node to the slope between its previous node to the node further ahead.

$$\phi_e = 2(\frac{\phi_e - \phi_P}{\phi_P - \phi_W})$$

We obtain r from the grid value, where r is the ratio of the slope between the following node of the cell wall and its previous node to the slope between its previous node to the node further ahead.

$$r_e = (\frac{\phi_E - \phi_P}{\phi_P - \phi_W})$$

where ϕ is can be represented as a function of r (e.g $\phi(r)$). By the given definition or design, we are capable of obtaining the value of cell wall from the known information from nodes. Where

Central Differencing scheme gives,

$$\phi(r) = r$$

QUICK scheme gives,

$$\phi(r) = \max[0, min(2r, 2, \frac{2(r+|r|)}{r+3})]$$

MUSCL scheme gives,

$$\phi(r) = \max[0, min(r, 1)]$$

2.10 Convergence Improvement

As for the results to not diverge with iterations, modification is required to be added to. It is the under relaxation that will be implemented in order to deal with the task. For this task, there are two place we apply under relaxation: the updating of velocity and pressure correction, and they are independent from another.

$$u_{n+1} = u_n + \omega_{vel}(u_{n+1} - u_n)$$

$$v_{n+1} = v_n + \omega_{vel}(v_{n+1} - v_n)$$

$$p'_{n+1} = p'_n + \omega_p(p_{n+1} - p_n)$$

For this task, ω_p is set 0.8 for all cases; at Re=100 ω_{vel} is set 0.8 under all schemes; at Re=1000 ω_{vel} is set 0.25 for CDS to converge, and 0.05 for QUICK and MUSCL to converge; at Re=5000 ω_{vel} is set 0.01 for CDS to converge, and QUICK and MUSCL is incapable of convergence.

3 Results and discussions

3.1 Central Differencing Scheme

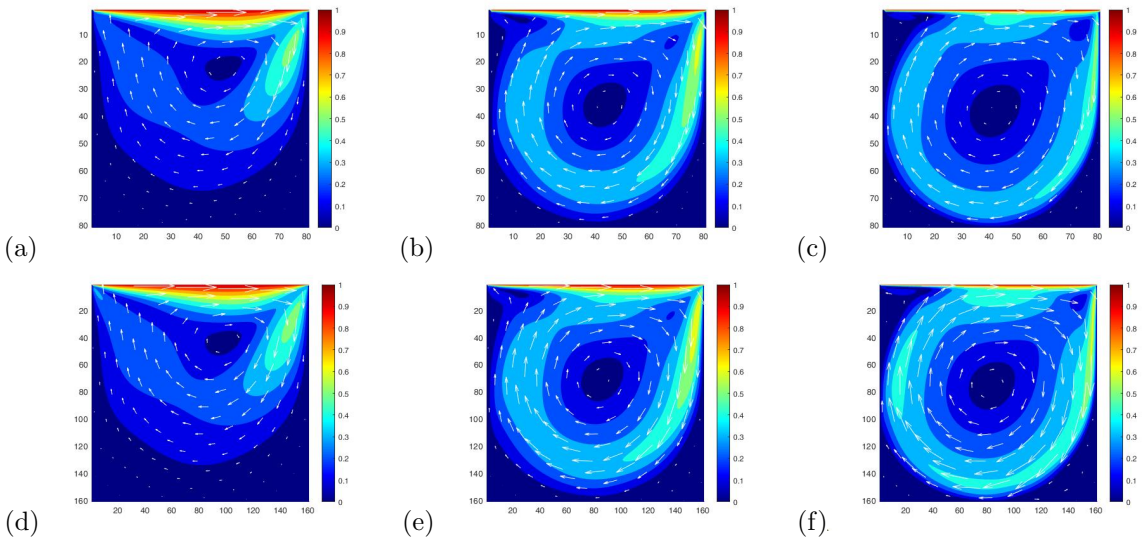


Figure 6: Result of (a) Re=100 81x81 (b) Re=1000 81x81 (c) Re=5000 81x81 (d) Re=100 161x161 (e) Re=1000 161x161(f) Re=5000 161x161

3.2 QUICK

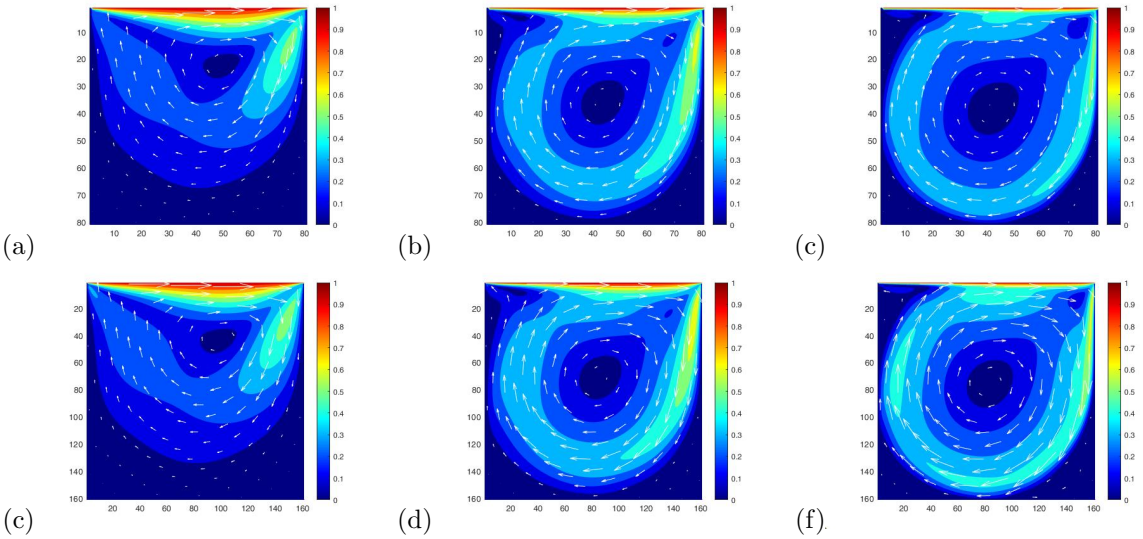


Figure 7: Result of (a) Re=100 81x81 (b) Re=1000 81x81 (c) Re=5000 81x81 (d) Re=100 161x161 (e) Re=1000 161x161(f) Re=5000 161x161

3.3 MUSCL

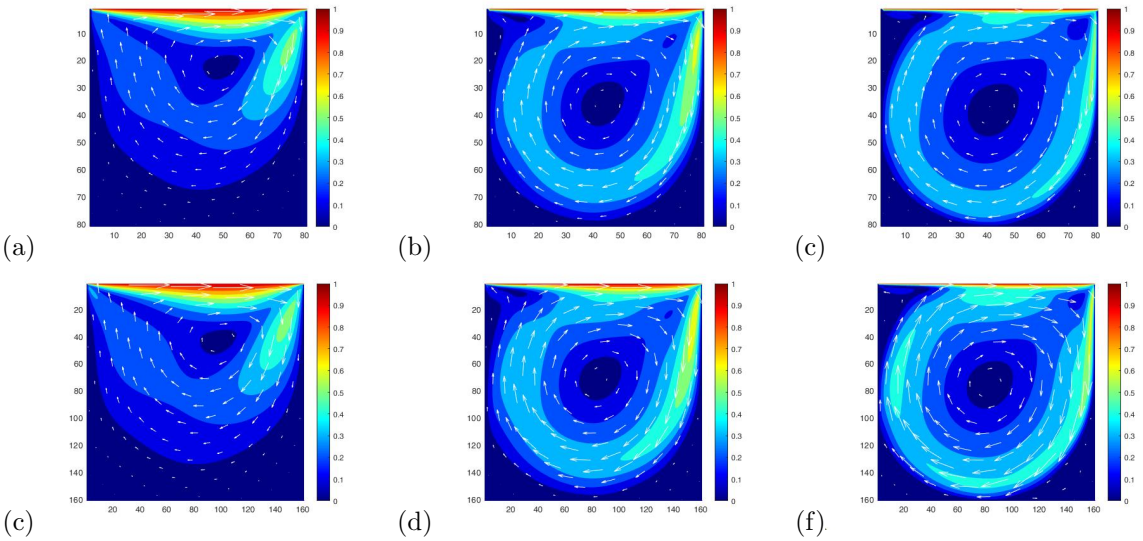
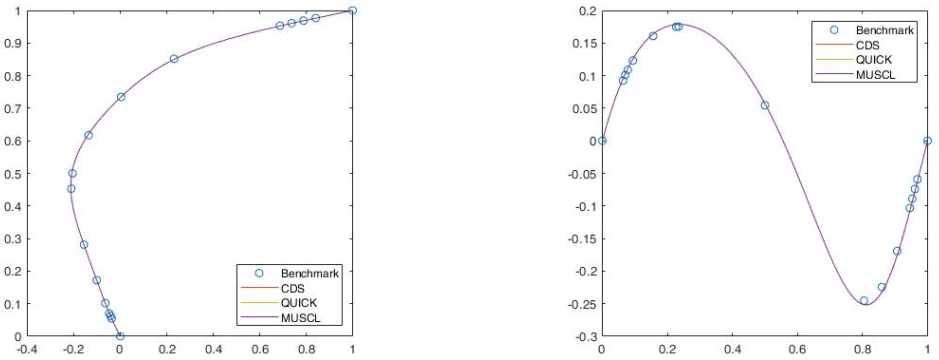


Figure 8: Result of (a) Re=100 81x81 (b) Re=1000 81x81 (c) Re=5000 81x81 (d) Re=100 161x161 (e) Re=1000 161x161(f) Re=5000 161x161

3.4 Comparison with Benchmark solution

In this section, we compare our results under different scheme with the benchmark solutions from Gihia et al. (1982) for u(y) at x=0.5 and v(x) at y=0.5.

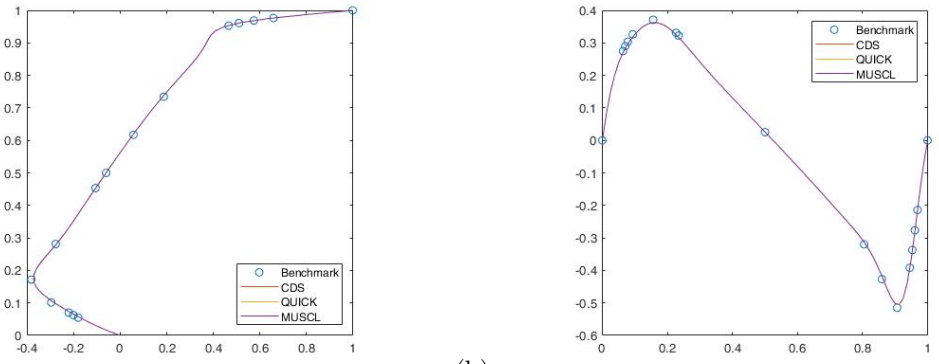
Re=100



(a) (b)

Figure 9: Comparison with Benchmark in (a) u at x=0.5 (b) v at y=0.5

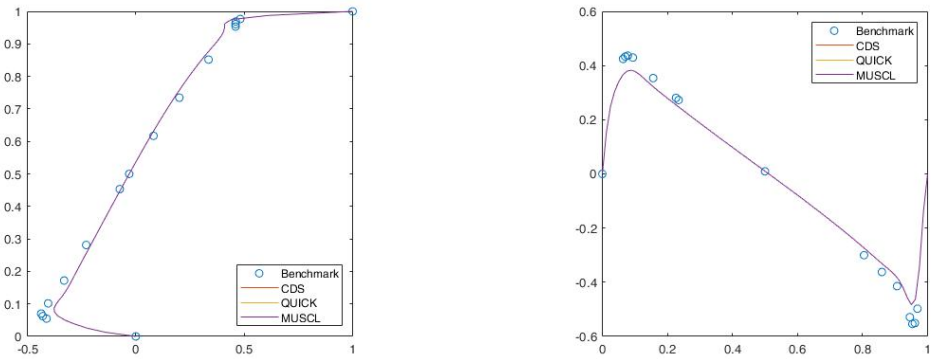
Re=1000



(a) (b)

Figure 10: Comparison with Benchmark in (a) u at x=0.5 (b) v at y=0.5

Re=5000



(a) (b)

Figure 11: Comparison with Benchmark in (a) u at x=0.5 (b) v at y=0.5

3.5 Effect of mesh size on the result

From the results, mesh size do not influence the result much at Re=100, the difference in between 81x81 and 161x161 is nonobvious; mesh size do influence the result at Re=1000, we can see there's difference in between 81x81 and 161x161; and mesh size influence the result at Re=5000 as well, we can see there's an obvious difference in between 81x81 and 161x161. We may as this point conclude that with the growth of Reynolds number, the mesh number is required to be high enough as to get a accurate result.

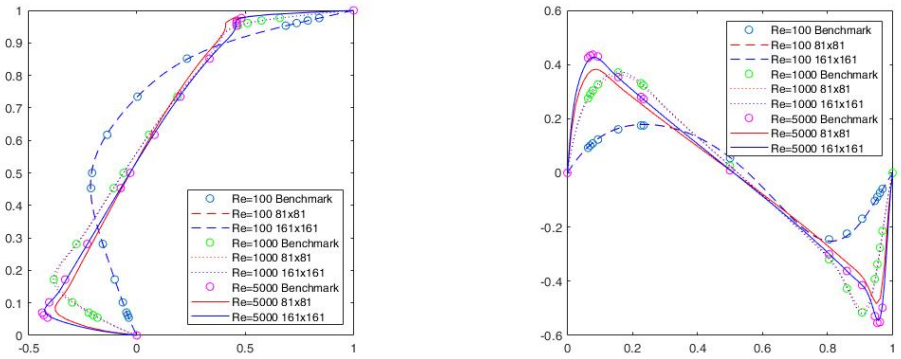


Figure 12: Comparison with Benchmark in (a) u at x=0.5 (b) v at y=0.5

4 Conclusion

As to prevent problems that would possibly take place(e.g checkerboard), some arrangements were done beforehand, including the implementation of Finite Volume method(FVM) and staggered grid. For this task, SIMPLE algorithm is our choice of coupling velocity and pressure. Furthermore, as for the simplification in applying various schemes, flux-limiter-operator is implemented. Also, under-relaxation greatly improve the wellness of convergence in this task. Under a high Reynolds number, the difficulty of obtaining a predicted result is higher than that of the low Reynolds number. Not only the time required for the result to converge, but also the certain modification required for the result to converge. From the comparison with Benchmark, we learn how mesh size would affect the outcome, mesh number is required to be high enough as to support a accurate result, yet, the execution time prolongs with the growth of mesh number, it is a commonly-seen trade-off in between. In other word, dealing a certain scenario, it would be better if we carefully arrange the mesh size beforehand, . Targeting on the result, the slope of the velocity profile is steeper at the wall/lid at high Reynolds number, which from the physics' point of view is reasonable, for the kinetic force dominating the flow-field instead of viscous force.

5 List of programs

```
//0.8 0.25 0.01
//0.8 0.05

#include <iostream>
#include <math.h>
#include "write.h"
#include "math.h"

#define PI 3.1415926
#define N 161
#define Re 5000
#define SCHEME 1
#define SOR_s 0.02
#define SOR_p 0.8
#define tol 1E-7
#define P 0
#define U 1
#define V 2
```

```
using namespace std;

//int p->0; u->1; v->2
void setBC(int);
void pseudo(int);
void scheme(int,int,int,int);
void corr();
double residual();
void result();

double u[N + 1][N] = { 0.0 };
double v[N][N + 1] = { 0.0 };
double p[N + 1][N + 1] = { 0.0 }, p_corr[N + 1][N] = { 0.0 };
double nu = (double)1 / (Re - 1);
double h = (double)1 / (N - 1);
double dm = 0;
double u_final[N][N], v_final[N][N], p_final[N][N], velocity_final[N][N];
char name1[] = "velocity.csv", name2[] = "u.csv", name3[] = "v.csv", name4[] = "p.csv";

int main() {
int i, iteration=0;
setBC(U);
setBC(V);
setBC(P);

do{
iteration += 1;
pseudo(U);
setBC(U);
pseudo(V);
setBC(V);
corr();
if (iteration % 50000 == 0) {
  cout << "iteration: " << iteration << " residual: " << dm << endl;
result();

}
} while (residual() > tol);//
result();
return 0;
}


/*Functions definition*/
void setBC(int k) {
int j;
switch (k) {
case 0:
//set boundary for p
for (j = 0; j < N + 1; j++) {
p[0][j] = p[1][j];
p[N][j] = p[N-1][j];
p[j][0] = p[j][1];
p[j][N] = p[j][N-1];
}
break;
case 1:
//set boundary for u
for (j = 0; j < N; j++) {
u[0][j] = 2 - u[1][j];
u[N][j] = -u[N - 1][j];
u[j][0] = 0;
u[j][N - 1] = 0;
}
break;
case 2:
//set boundary for v
for (j = 0; j < N; j++) {
v[0][j] = 0;
v[N-1][j] = 0;
v[j][0] = -v[j][1];
v[j][N] = -v[j][N - 1];
}
break;
}
};

double d_u[N + 1][N] = { 0.0 };
double d_v[N][N + 1] = { 0.0 };
double a_u[N + 1][N] = { 0.0 };
double a_v[N][N + 1] = { 0.0 };
double u_E, u_W, v_N, v_S;
double a_E, a_W, a_N, a_S, a_P;
```

```
double temp = 0;
void pseudo(int k) {
int i, j;
switch (k) {
case 1:
//Pre-cal for u
for (i = 1; i < N; i++) {
for (j = 1; j < N - 1; j++) {
scheme(U, SCHEME,i,j);
a_E = -u_E * h / 2.0 + nu;
a_W = u_W * h / 2.0 + nu;
a_N = -v_N * h / 2.0 + nu;
a_S = v_S * h / 2.0 + nu;
a_u[i][j] = (u_E - u_W + v_N - v_S) * h/2.0 + 4 * nu;
d_u[i][j] = h / a_u[i][j];
temp = u[i][j];
u[i][j] = (a_E * u[i][j + 1] + a_W * u[i][j - 1] + a_N * u[i - 1][j] + a_S * u[i + 1][j] - h * (p[i][j + 1] - p[i][j]))
/ a_u[i][j];
u[i][j] = temp + SOR_s * (u[i][j] - temp);
}
}
break;

case 2:
//Pre-cal for v
for (i = 1; i < N - 1; i++) {
for (j = 1; j < N; j++) {
scheme(V, SCHEME,i,j);
a_E = -u_E * h *0.5 + nu;
a_W = u_W * h * 0.5 + nu;
a_N = -v_N * h * 0.5 + nu;
a_S = v_S * h * 0.5 + nu;
a_v[i][j] = (u_E - u_W + v_N - v_S) * h * 0.5 + 4 * nu;
d_v[i][j] = h / a_v[i][j];
temp = v[i][j];
v[i][j] = (a_E * v[i][j + 1] + a_W * v[i][j - 1] + a_N * v[i - 1][j] + a_S * v[i + 1][j] - h * (p[i][j] - p[i + 1][j]))
/ a_v[i][j];//
v[i][j] = temp + SOR_s * (v[i][j] - temp);
}
}
break;
}

};

void corr() {
int i, j;
double u_corr, v_corr;
double b;
for (i = 1; i < N; i++) {
for (j = 1; j < N; j++) {
p_corr[i][j] = { 0.0 };
}
}
//correct p
for (i = 1; i < N ; i++) {
for (j = 1; j < N ; j++) {
b = h * (u[i][j] - u[i][j - 1] +  v[i - 1][j]-v[i][j]);
a_P = h* d_u[i][j - 1] + h* d_u[i][j] + h* d_v[i - 1][j] + h* d_v[i][j];
p_corr[i][j] = (p_corr[i][j - 1] * h* d_u[i][j - 1] + p_corr[i][j + 1] * h* d_u[i][j] + p_corr[i - 1][j] * h* d_v[i - 1][j]
+ p_corr[i + 1][j] * h* d_v[i][j] - b) / a_P;
p[i][j] = p[i][j]+ SOR_p * p_corr[i][j];

}
}
setBC(P);
//correct u
for (i = 1; i < N ; i++) {
for (j = 1; j < N - 1; j++) {
u_corr= d_u[i][j] * (p_corr[i][j] - p_corr[i][j + 1]);
temp = u[i][j];
u[i][j] += u_corr;
  temp = u[i][j];
}
}
setBC(U);
//correct v
for (i = 1; i < N - 1; i++) {
for (j = 1; j < N ; j++) {
v_corr = d_v[i][j] * (p_corr[i+1][j] - p_corr[i][j]);
v[i][j] += v_corr;
}
}
setBC(V);
```

```
};

double residual() {
int i, j;
dm = 0;
for (i = 1; i < N - 1; i++) {
for (j = 1; j < N - 1; j++) {
dm += abs(u[i][j - 1] - u[i][j] + v[i][j]-v[i - 1][j]);
}
}
return dm;
}

double r_e, r_w, r_n, r_s;
double phi_e, phi_w, phi_n, phi_s;
void scheme(int k, int m, int i, int j) {
if (k == U) {
switch (m) {
case 1:

r_e = (u[i][j + 1] - u[i][j]) / (u[i][j] - u[i][j - 1]);
r_w = (u[i][j] - u[i][j - 1]) / (u[i][j - 1] - u[i][j - 2]);
r_n = (v[i - 1][j + 1] - v[i - 1][j]) / (v[i - 1][j] - v[i - 1][j - 1]);
r_s = (v[i][j + 1] - v[i][j]) / (v[i][j] - v[i][j - 1]);
//phi
phi_e = r_e;
phi_n = r_n;
phi_s = r_s;
phi_w = r_w;
//
u_E = u[i][j] + 0.5 * phi_e * (u[i][j] - u[i][j - 1]);
u_W = u[i][j - 1] + 0.5 * phi_w * (u[i][j - 1] - u[i][j - 2]);
v_N = v[i - 1][j] + 0.5 * phi_n * (v[i - 1][j] - v[i - 1][j - 1]);
v_S = v[i][j] + 0.5 * phi_s * (v[i][j] - v[i][j - 1]);

if (u[i][j] == u[i][j - 1])
u_E = 0;
if ((u[i][j - 1] == u[i][j - 2]))
u_W = 0;
if (v[i - 1][j] == v[i - 1][j - 1])
v_N = 0;
if ((v[i][j] == v[i][j - 1]))
v_S = 0;

case 2:
r_e = (u[i][j + 1] - u[i][j]) / (u[i][j] - u[i][j - 1]);
r_w = (u[i][j] - u[i][j - 1]) / (u[i][j - 1] - u[i][j - 2]);
r_n = (v[i - 1][j + 1] - v[i - 1][j]) / (v[i - 1][j] - v[i - 1][j - 1]);
r_s = (v[i][j + 1] - v[i][j]) / (v[i][j] - v[i][j - 1]);
//phi
phi_e = max(0.0, min(2 * r_e, min(2.0, 2 * (r_e + abs(r_e)) / (r_e + 3))));
phi_n = max(0.0, min(2 * r_n, min(2.0, 2 * (r_n + abs(r_n)) / (r_n + 3))));
phi_s = max(0.0, min(2 * r_s, min(2.0, 2 * (r_s + abs(r_s)) / (r_s + 3))));
phi_w = max(0.0, min(2 * r_w, min(2.0, 2 * (r_w + abs(r_w)) / (r_w + 3))));
//
u_E = u[i][j]+0.5*phi_e*(u[i][j] - u[i][j - 1]) ;
u_W = u[i][j - 1]+0.5*phi_w*(u[i][j - 1] - u[i][j - 2]);
v_N = v[i - 1][j]+0.5*phi_n*(v[i - 1][j] - v[i - 1][j - 1]);
v_S = v[i][j]+0.5*phi_s* (v[i][j] - v[i][j - 1]);
break;
case 3:
r_e = (u[i][j + 1] - u[i][j]) / (u[i][j] - u[i][j - 1]);
r_w = (u[i][j] - u[i][j - 1]) / (u[i][j - 1] - u[i][j - 2]);
r_n = (v[i - 1][j + 1] - v[i - 1][j]) / (v[i - 1][j] - v[i - 1][j - 1]);
r_s = (v[i][j + 1] - v[i][j]) / (v[i][j] - v[i][j - 1]);
//phi
phi_e = max(0.0, min(2 * r_e, min(2.0, (r_e + 1) / 2)));
phi_n = max(0.0, min(2 * r_n, min(2.0, (r_n + 1) / 2)));
phi_s = max(0.0, min(2 * r_s, min(2.0, (r_s + 1) / 2)));
phi_w = max(0.0, min(2 * r_w, min(2.0, (r_w + 1) / 2)));
//
u_E = u[i][j] + 0.5 * phi_e * (u[i][j] - u[i][j - 1]);
u_W = u[i][j - 1] + 0.5 * phi_w * (u[i][j - 1] - u[i][j - 2]);
v_N = v[i - 1][j] + 0.5 * phi_n * (v[i - 1][j] - v[i - 1][j - 1]);
v_S = v[i][j] + 0.5 * phi_s * (v[i][j] - v[i][j - 1]);
break;

}
}
else if (k == V) {
switch (m) {
case 1:
u_E = (u[i][j] + u[i + 1][j]) * 0.5;
u_W = (u[i][j - 1] + u[i + 1][j - 1]) * 0.5;
```

```
v_N = (v[i - 1][j] + v[i][j]) * 0.5;
v_S = (v[i][j] + v[i + 1][j]) * 0.5;
break;
case 2:
r_e = (u[i][j] - u[i + 1][j]) / (u[i + 1][j] - u[i + 2][j]);
r_w = (u[i][j - 1] - u[i + 1][j - 1]) / (u[i + 1][j - 1] - u[i + 2][j - 1]);
r_n = (v[i - 1][j] - v[i][j]) / (v[i][j] - v[i + 1][j]);
r_s = (v[i][j] - v[i + 1][j]) / (v[i + 1][j] - v[i + 2][j]);
//phi
phi_e = max(0.0, min(2 * r_e, min(2.0, 2 * (r_e + abs(r_e)) / (r_e + 3))));
phi_n = max(0.0, min(2 * r_n, min(2.0, 2 * (r_n + abs(r_n)) / (r_n + 3))));
phi_s = max(0.0, min(2 * r_s, min(2.0, 2 * (r_s + abs(r_s)) / (r_s + 3))));
phi_w = max(0.0, min(2 * r_w, min(2.0, 2 * (r_w + abs(r_w)) / (r_w + 3))));
//
u_E = u[i + 1][j] + 0.5 * phi_e * (u[i + 1][j] - u[i + 2][j]);
u_W = u[i + 1][j - 1] + 0.5 * phi_w * (u[i + 1][j - 1] - u[i + 2][j - 1]);
v_N = v[i][j] + 0.5 * phi_n * (v[i][j] - v[i + 1][j]);
v_S = v[i + 1][j] + 0.5 * phi_s * (v[i + 1][j] - v[i + 2][j]);
break;
case 3:
r_e = (u[i][j] - u[i + 1][j]) / (u[i + 1][j] - u[i + 2][j]);
r_w = (u[i][j - 1] - u[i + 1][j - 1]) / (u[i + 1][j - 1] - u[i + 2][j - 1]);
r_n = (v[i - 1][j] - v[i][j]) / (v[i][j] - v[i + 1][j]);
r_s = (v[i][j] - v[i + 1][j]) / (v[i + 1][j] - v[i + 2][j]);
//phi
phi_e = max(0.0, min(2 * r_e, min(2.0, (r_e + 1) / 2)));
phi_n = max(0.0, min(2 * r_n, min(2.0, (r_n + 1) / 2)));
phi_s = max(0.0, min(2 * r_s, min(2.0, (r_s + 1) / 2)));
phi_w = max(0.0, min(2 * r_w, min(2.0, (r_w + 1) / 2)));
//
u_E = u[i + 1][j] + 0.5 * phi_e * (u[i + 1][j] - u[i + 2][j]);
u_W = u[i + 1][j - 1] + 0.5 * phi_w * (u[i + 1][j - 1] - u[i + 2][j - 1]);
v_N = v[i][j] + 0.5 * phi_n * (v[i][j] - v[i + 1][j]);
v_S = v[i + 1][j] + 0.5 * phi_s * (v[i + 1][j] - v[i + 2][j]);
break;
}
}

}



void result() {
int i, j;
for (i = 0; i < N; i++) {
for (j = 0; j < N; j++) {
u_final[i][j] = (u[i][j] + u[i + 1][j]) * 0.5;
v_final[i][j] = (v[i][j] + v[i][j + 1]) * 0.5;
p_final[i][j] = (p[i][j] + p[i][j + 1] + p[i + 1][j] + p[i + 1][j + 1]) * 0.25;
velocity_final[i][j] = sqrt(u_final[i][j] * u_final[i][j] + v_final [i] [j] * v_final[i][j]);
}
}
write(&velocity_final[0][0], N, N, name1);
write(&u_final[0][0], N, N, name2);
write(&v_final[0][0], N, N, name3);
write(&p_final[0][0], N, N, name4);
}
```

6 Reference

Snyder, W. T., Goldstein, G. A. (1965). An analysis of fully developed laminar flow in an eccentric annulus. AIChE Journal, 11(3), 462–467. Gihia et al. (1982)High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method*

T. GanesanM. Awang (2015). Large Eddy Simulation (LES) for Steady-State Turbulent Flow Prediction