2D Simulation of a Flow Past a Rectangle Near a Wall

BSc Thesis

Delft University of Technology
Faculty of Applied Sciences
Department of Chemical Engineering

Starting date: 6th of March 2018
Defense date: 11th of January 2019

Author:
D.J.H. Den Beer Poortugael
Student number: 4371178

Supervisor:
L. Portela

Reviewers:
S. Kenjeres

Abstract

A numerical study has been done on a rectangle in a 2D pipe flow. Therefore a Matlab code is made from scratch on basis of the SIMPLE algorithm. In the appendix is explained step by step exactly how all equations are implemented in Matlab. Focusing on low Reynolds numbers (1<Re<80) are different simulations made of the flow with the rectangle. Different heights of the rectangle in the pipe in a pressure driven flow are considered. Drag, lift and torque coefficients that were obtained from these simulations are plotted against the Reynolds number. Furthermore are shear flows as a consequence of a moving wall considered. Plots and other data of the dimensionless coefficients are included. The goal to make a good simulation of the flow past a rectangle is achieved. Different possible improvements to the code have been opposed, so it can be improved in the future to make also simulations of more complex geometries than a rectangle.

Contents

Chapter 1

Introduction

In many processes in industry occurs the transportation of particulate solids by a fluid. For instance transporting gold ores in a fluid in mining industry or for more practical use as increasing the contact area between a solid and a fluid in a fluidized bed reactor. A main difficulty in designing such processes is modeling the behaviour of motion of the solids and the fluid. Particles move constantly in all directions dependent on their shape and are affected by other particles or solid boundaries as the wall of a pipe. The geometries of the particles are of real importance and make it a lot harder to correctly model such flows. Often are assumptions made that the particles in the flow are spherical for making it easier to model. In practice however will most particles be non-spherical and a lot of extra dynamic properties come in play such as a pitching torque acting on the particle. If we could exactly model the behaviour of such particles, then the designing of such processes could be optimized.

In this project we tried to make a start in modeling the behaviour of these non-spherical particles. This is done by developing a Matlab code that could simulate a pipe flow with an obstacle in it. Because we started from scratch is in this project the obstacle for simplicity chosen to be a square. Many different numerical studies have already been done on this subject. For instance on the wake that is created behind the obstacle. It has been shown that the wake increases for increasing Reynolds number [Bhattacharyya and Maiti (2004)]. In the same research is found that downstream of the square the boundary layer along the wall separates in the vicinity of the vortex formation region and a recirculation region forms on the wall at all Reynolds numbers. All this information among with other studies on this subject found in the literature is used in this project to refer to if our code yields appropriate output.

In the future could this code be optimized and extended. We achieved to write a code and the data that it produces is convenient with background information. Although it costs a lot of time to actually produce it. There can be a lot optimized on the code, which will be discussed as well. First thing that could be extended is changing the geometry of the obstacle. In our short term goal of making a start of a model is the obstacle kept a square. For trying to achieve the long term goal of modeling the behaviour of non-spherical particles, should the geometry be adjusted. If the code could be run with obstacle with different shapes, then behaviour of actual non-spherical particles could be studied. The research question that comes from our short term goal and is tried to be solved during this project is:

"Can we simulate a flow past a rectangle and does it show realistic behaviour?"

So in order to achieve this goal is first a Matlab code written. Gradually is the SIMPLE algorithm build and then is tried to process the data which was yielded. In the Appendix is exactly worked out step by step how the different parts of the SIMPLE algorithm should be implemented in Matlab.

In this report will first the relevant theory for the used formulae be explained and then how they are implemented in the Matlab code. Considering the amount of time it took to write a working code, little time was spent on actually obtaining results of it. The results that were yielded are compared to literature and could then more be used to verify whether the code is correct. The achieved results are simulations of the rectangle in the flow at different heights in the pipe for a pressure driven flow. Further are a few simulations made of the pipe with moving wall(s) resulting in a shear flow.

Chapter 2

Theory

In this chapter, the relevant theory which was used to build the Matlab code will be summarized. For simulation of the flow is the pipe divided into small squares and then the finite volume method is used. An illustration of a staggered grid is depicted in figure 2.1. More information about buildup of the grid and assumptions made for boundary conditions as ghost points for instance are in the Appendix.

After defining the grid is the flow simulated iteratively via the SIMPLE algorithm. SIMPLE is an acronym for Semi-Implicit Method for Pressure Linked Equations. The first step here is the initiation. In this step will be defined the properties and boundary conditions of the flow and the initial guesses for the pressure field and the velocity fields are made. Using these guessed fields will solve the momentum equation yielding new velocity fields, u^* and v^*. The equation that is solved is the Navier-Stokes equation:

$$\frac{\partial}{\partial t}(\rho\vec{U}) = -\nabla \cdot (\rho\vec{U} \otimes \vec{U}) + \nabla \cdot \tau - \nabla p \tag{2.1}$$

Next the pressure correction field is determined by solving the continuity equation:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2.2}$$

With the pressure correction field can the velocity corrections be considered. The guessed pressure field and velocity fields are updated with their corresponding correction fields subjected to an under-relaxation factor α. It is chosen that for this project that $\alpha = 0.6$. Derivations for the Navier-Stokes and continuity equation are in the Appendix. How these equations are implemented in Matlab will be described in chapter 3. The flow is continuous which means that after the last column of nodes the first column comes. So the simulated part of the pipe is in fact repeated multiple times after each other.

After updating all fields will these fields be used as initial conditions to start the same process again. This is done until the fields converge towards a steady state solution. Every iteration works with a change in time dt. With smaller dt is more accurate data obtained as the error gets smaller [Garcia and Wagner (2000)]. Due to the smaller increase in time are more iterations required so this costs more computation time. As a matter of time is in this project chosen for $dt = 10^{-3}$. As shown by Garcia et al. is the error scaling with dt^2 so for a lower dt would the error decrease fast.

The grid that is used in this project is 60×60. Larger grid sizes would again increase the accuracy of the data but cost more computation time. The nodes on the square obstacle are 6×6. In a study for grid sizes has a body with 80 nodes an error of 2% on the drag force and increasing the number of nodes on the body to 160 reduces the error to 1% [Harichandan and Roy (2012)]. The nodes on the body in this project are thus little and the error that comes therefrom can be very large if the data of Harichandan and Roy would be extrapolated.

After convergence is reached could with the output of the code be looked at force distribution throughout the channel and forces acting on the block. The flow is divided into multiple different smaller volumes and therefore it is possible to calculate for every position in the pipe which forces are dominant. Three kinds of forces are considered: inertial, viscous and pressure forces. The inertial forces are the convection of momentum. They depend on density and velocity. For denser fluids and at higher flow velocities

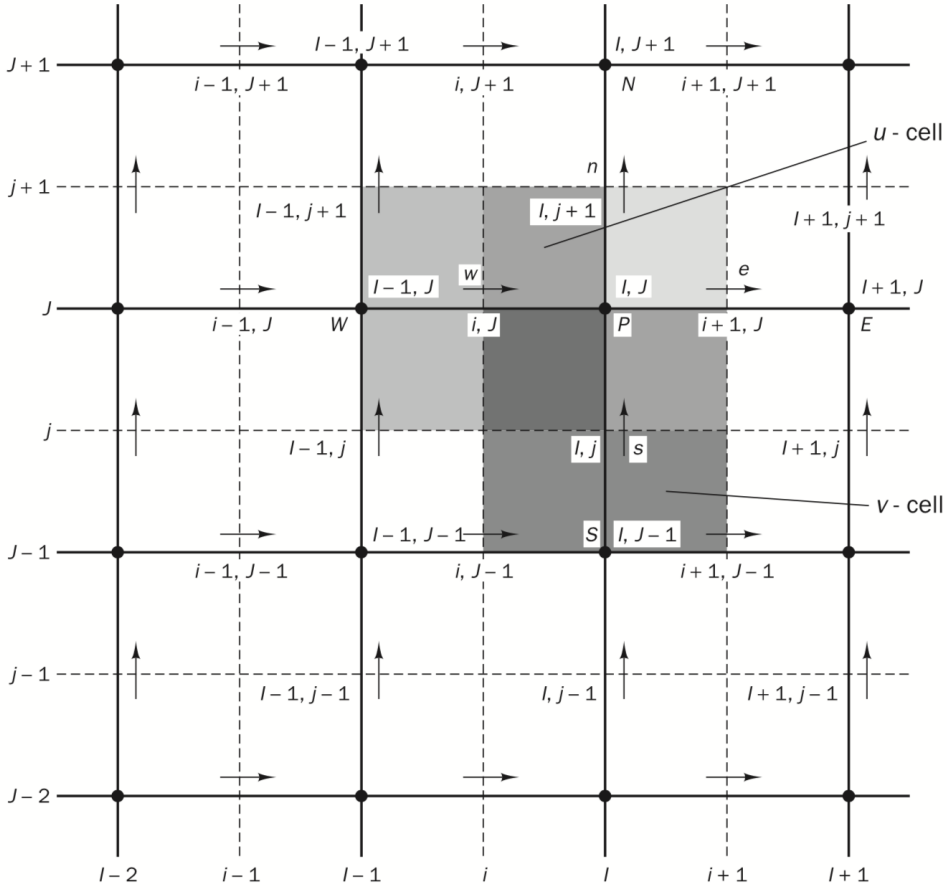

Figure 2.1: Staggered grid

the convection becomes higher. Viscous forces depend on the viscosity and on velocity gradients. For higher viscosity and larger velocity gradient grows the stress higher and become viscous forces more important. Closer to the wall are flow velocities lower and are viscous forces of more importance. Moving further away from the wall reduces the shear stress and flow velocities increase. Here inertial forces grow more dominant. Pressure forces arise from pressure gradients. In a pressure driven flow is a pressure difference present over the pipe which results in a constant pressure force towards the lower pressure end. At higher velocity the pressure decreases as Bernoulli states. Therefore could, when the rectangle comes closer to the wall and velocities change, different pressure forces be created.

Even as the force distribution in the flow can the forces on the rectangle be considered. The drag force is a consequence of the fluid that exerts a force on the block in the direction of the flow. If a flow without walls would be considered, an unbounded flow, then there would not be a lift on the rectangle. The flows over and under the block are identical and a net force of zero would be the result. This changes when the flow gets bounded by a wall. The flow is not identical anymore and a lift would be created. When the obstacle would move towards the wall, the flow under it would move faster than the flow over it due to continuity. The result of this is a pressure difference and a lift towards the wall is created. When the particle is pushed towards the wall, will the viscous stresses in the flow under the block become more important. At a certain point they become so high that the viscous forces slow down the under flow and a lift away from the wall will be created.

The forces from a cell that can act on the rectangle are the inertial, viscous and pressure forces. They results in drag, lift and a torque on the particle. The torque τ is the rotational force and is computed with formula:

$$\tau = r * F \tag{2.3}$$

with r is the arm that is made and F the force applied on the block. The midpoint of the block is taken as reference point and so is the distance to the middle of every side the arm r made for every force.

With the obtained drag, lift and torque can now the dimensionless coefficients be computed. These coefficients are calculated as follows:

$$C_D = 2\frac{F_{drag}}{p_{grad}HH_{part}dy} \tag{2.4}$$

$$C_L = 2\frac{F_{lift}}{p_{grad}HL_{part}dx} \tag{2.5}$$

$$C_T = 2\frac{torque}{p_{grad}HH_{part}dyL_{part}dx} \tag{2.6}$$

In these equations is H the height of the pipe and H_{part} and L_{part} the amount of grid cells facing the block. H_{part} and L_{part} multiplied with dy and dx respectively yields the surface of that side of the block. With these dimensionless coefficients can the forces be put more into perspective of flow quantities and can be compared for different cases.

The Reynolds number is computed via:

$$Re = \frac{\rho v_{cl} H_{part} dy}{\mu} \tag{2.7}$$

The reference point for the velocity is here chosen in the central line.

Chapter 3

Numerical method

3.1 Flow simulation

In this chapter will be explained how the theory of chapter 2 is used numerically. After initiation of the SIMPLE algorithm the first thing that has to be solved is the Navier-Stokes equation for every grid point. This is done by discretizing it. The part by part discretization can be found in the Appendix. The momentum balance for the x-direction is

$$\frac{\partial}{\partial t}\rho u = -\frac{\partial}{\partial x}(\rho uu) - \frac{\partial}{\partial y}(\rho vu) + \frac{\partial}{\partial x}\left[2\mu\frac{\partial u}{\partial x}\right] + \frac{\partial}{\partial y}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] + p_{grad} - \frac{\partial p'}{\partial x} \tag{3.1}$$

which is discretized to

$$\begin{aligned}\frac{\rho}{dt}\left(u^*_{i,j} - vx_{i,j}\right) = \frac{\rho}{dx}\left((0.5(vx_{i,j-1} + vx_{i,j}))^2 - (0.5(vx_{i,j} + vx_{i,j+1}))^2\right) \\ + 0.25\frac{\rho}{dy}((vy_{i-1,j-1} + vy_{i-1,j}) * (vx_{i-1,j} + vx_{i,j}) - (vy_{i,j-1} + vy_{i,j}) * (vx_{i,j} + vx_{i+1,j})) \\ + \frac{2\mu}{dx^2}(vx_{i,j+1} - 2vx_{i,j} + vx_{i,j-1}) \\ + \frac{\mu}{dy}\left(\frac{vx_{i+1,j} - 2vx_{i,j} + vx_{i-1,j}}{dy} + \frac{vy_{i,j} - vy_{i,j-1} - vy_{i-1,j} + vy_{i-1,j-1}}{dx}\right) \\ + pgrad + \frac{p_{i-1,j-1} - p_{i-1,j}}{dx}\end{aligned} \tag{3.2}$$

The momentum balance for the y-direction is

$$\frac{\partial}{\partial t}\rho v = -\frac{\partial}{\partial x}(\rho uv) - \frac{\partial}{\partial y}(\rho vv) + \frac{\partial}{\partial y}\left[2\mu\frac{\partial v}{\partial y}\right] + \frac{\partial}{\partial x}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] - \frac{\partial p'}{\partial y} \tag{3.3}$$

which is discretized to

$$\begin{aligned}\frac{\rho}{dt}\left(v^*_{i,j} - vy_{i,j}\right) = 0.25\frac{\rho}{dx}((vy_{i,j-1} + vy_{i,j}) * (vx_{i,j} + vx_{i+1,j}) - (vy_{i,j} + vy_{i,j+1}) * (vx_{i,j+1} + vx_{i+1,j+1})) \\ + \frac{\rho}{dx}\left((0.5(vy_{i-1,j} + vy_{i,j}))^2 - (0.5(vy_{i,j} + vy_{i+1,j}))^2\right) \\ + \frac{2\mu}{dy^2}(vy_{i+1,j} - 2vy_{i,j} + vx_{i-1,j}) \\ + \frac{\mu}{dx}\left(\frac{vx_{i+1,j+1} - vx_{i,j+1} - vx_{i+1,j} + vx_{i,j}}{dy} + \frac{vy_{i,j+1} - 2vy_{i,j} + vy_{i,j+1}}{dx}\right) \\ + \frac{p_{i-1,j} - p_{i,j}}{dy}\end{aligned} \tag{3.4}$$

When these momentum balances are solved for every grid cell, u^* and v^* are yielded. With these values can the continuity be checked. The continuity equation as stated in equation 2.2 can be written as:

$$\rho \mathrm{d}y(u_1 - u_2) + \rho \mathrm{d}x(v_1 - v_2) = 0 \tag{3.5}$$

As worked out in the Appendix, can equation 3.5 be derived to:

$$2dt\left(\frac{dy}{dx} + \frac{dx}{dy}\right)p_{i,j} - \frac{dt * dy}{dx} * (p_{i,j-1} + p_{i,j+1}) - \frac{dt * dx}{dy}(p_{i-1,j} + p_{i+1,j}) \\ = \rho dy(u^*_{i+1,j+1} - u^*_{i+1,j}) + \rho dx(v^*_{i+1,j} + v^*_{i,j}) \tag{3.6}$$

Equation 3.6 is now a Poisson equation for pressure correction. Every pressure correction in a grid cell is dependent on the pressure corrections, u^* and v^* form the cells surrounding it. The Poisson equation for every cell is put in two matrices and yields a matrix with all pressure corrections. How this is done exactly is explained in the Appendix.

With the known pressure corrections can the velocity and pressure fields be updated. The velocity corrections for u and v are respectively calculated as

$$u'_{i,j} = \frac{dt}{\rho dx}(p'_{i,j-1} - p'_{i,j}) \tag{3.7}$$

and

$$v'_{i,j} = \frac{dt}{\rho dy}(p'_{i,j} - p'_{i,j+1}) \tag{3.8}$$

The convergence can now be checked. If the corrections are smaller then a preassigned convergence factor, then the flow is converged and the algorithm stops. If the flow is not yet converged, then the guessed velocities and pressures are updated and one iteration is done. The process is repeated until convergence is reached. The velocities are updated with under-relaxation α as:

$$u^{new}_{i,j} = u^*_{i,j} + \alpha u'_{i,j} \tag{3.9}$$

and

$$v^{new}_{i,j} = v^*_{i,j} + \alpha v'_{i,j} \tag{3.10}$$

The under-relaxation factor is also involved in the pressure update, but as $1 - \alpha$:

$$p_{i,j} = p_{i,j} + (1 - \alpha)p'_{i,j} \tag{3.11}$$

3.2 Processing data

When the flow is converged, analysis can be done on it. First thing that is done on every set of data is plotting the stream lines. The build in function 'streamline' of Matlab yields from the velocity fields, a plot with the stream lines in the pipe. Such a plot clearly illustrates the flow pattern around the rectangle. This can be beneficial to see if for instance a wake is formed behind the rectangle. The position and the size are then evident.

Eventually are the forces acting on the rectangle considered. The drag force on the rectangle is computed by summing up the forces in the x-direction on the block of all grid cells surrounding it. The lift force is computed in the same way, only for the y-direction. The forces from a cell that can act on the rectangle are the inertial, viscous and pressure forces. The torque τ is the rotational force and is computed with formula:

$$\tau = r * F \tag{3.12}$$

with r is the arm that is made and F the force applied on the block. To compute this torque the force taken is the force acting in every grid cell. These forces are multiplied with the distance to the middle of the side on which they act. The midpoint of the block is taken as reference point and so is the distance to the middle of every side the arm r made for every force.

Chapter 4

Results

In this chapter will be discussed all the results obtained with the code. The code has been run for different scenarios. The distance between the wall and block have been varied for different Reynolds numbers in the case of a pressure driven flow. The case with shear flow has also been simulated. With moving one or two walls at different velocities, different flow profiles have been obtained. For all simulations are the drag, lift and torque coefficient considered.

4.1 $h/H = 1/2$

The relation h/H stands for the position of the block compared to the wall. h is the distance from the wall to the middle of the block and H is the total height of the pipe or the distance between both walls. So $h/H = 0$ the block is against the bottom wall and $h/H = 1$ means that the block is against the upper wall. $h/H = 1/2$ means that the block is exactly in the middle of the flow and in $h/H = 1/4$ and $h/H = 1/8$, which will be clarified in section 4.2 and 4.3 respectively, is the block closer to the bottom wall.

The streamlines around the block at $h/H = 1/2$ are with different Reynolds numbers depicted in figures 4.1 and 4.2.

It can be seen that the gap that is created behind block gets bigger for higher Reynolds numbers. Moreover are vortexes observed for increasing Reynolds number. These get also bigger for higher Reynolds numbers. In the research of Bhattacharyya and Maiti is the increasing gap and more vortexes also observed for

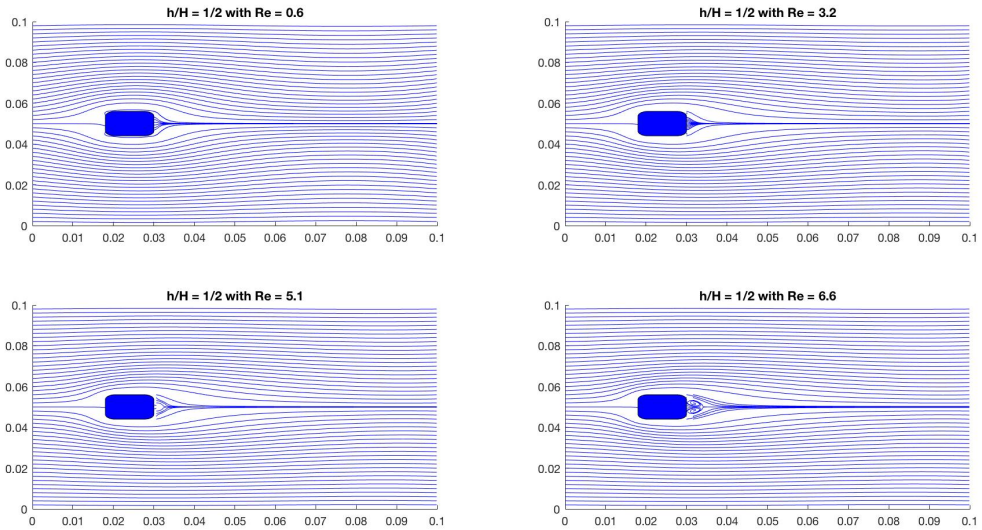

Figure 4.1: Flow simulation with $h/H = 1/2$ (1)

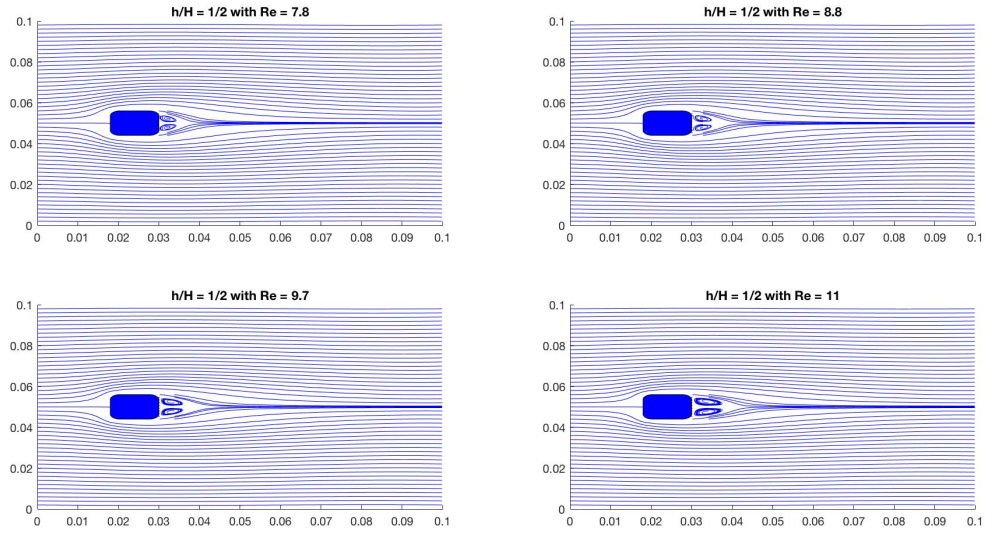

Figure 4.2: Flow simulation with $h/H = 1/2$ (2)

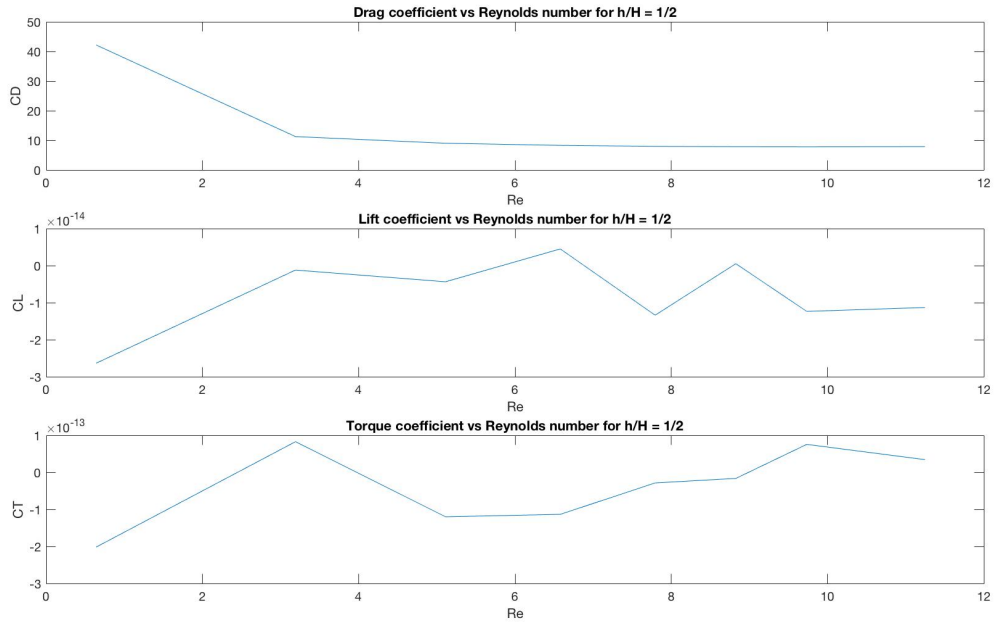

Figure 4.3: Dimensionless coefficients for $h/H = 1/2$

Re	0.6	3.2	5.1	6.6	7.8	8.8	9.7	11
CD	42	11	9.0	8.3	8.0	7.8	7.8	7.9
CL	-2.6e-14	-1.2e-15	-4.4e-15	4.5e-15	-1.3e-14	5.2e-16	-1.2e-14	-1.1e-14
CT	-2.0e-13	8.3e-14	-1.2e-13	-1.1e-13	-2.8e-14	-1.6e-14	7.6e-14	3.4e-14

Table 4.1: Dimensionless coefficients with Reynolds number at $h/H = 1/2$

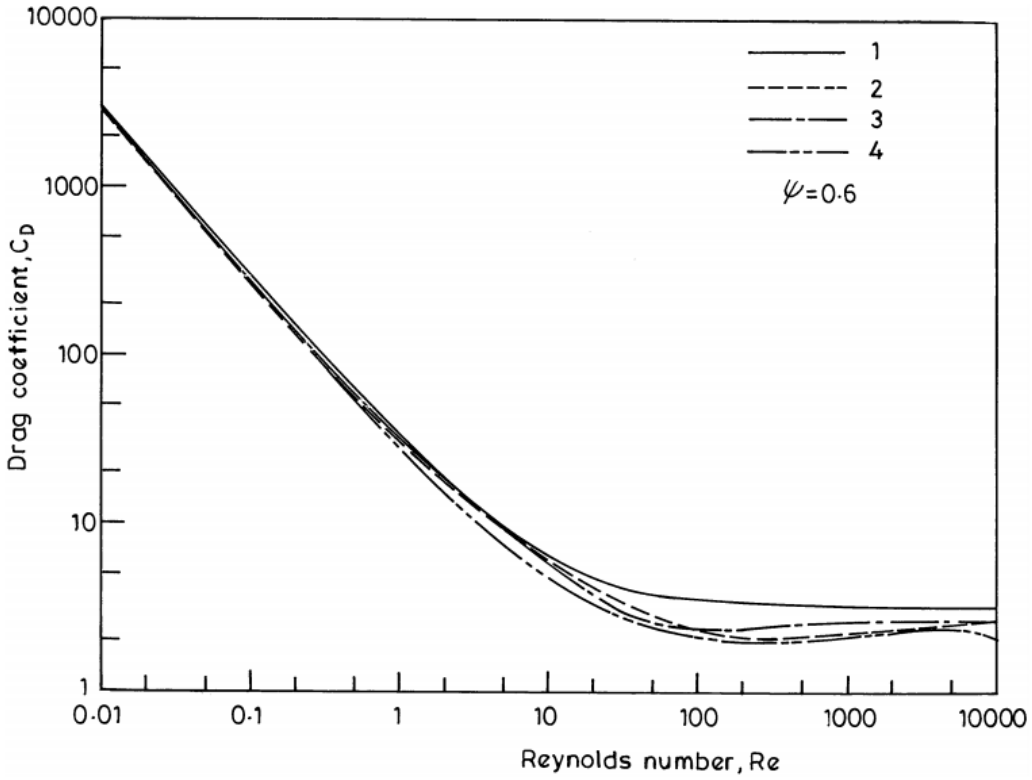

Figure 4.4: Plot of drag coefficient from Chabbra and Agarwal

increasing the Reynolds number [Bhattacharyya and Maiti (2004)]. The drag coefficient decreases for increasing Reynolds number and looks like to reach a certain constant value of approximately 7.8. In the research of Chabbra and Agarwal the Drag coefficient decreases in the same pattern for increasing the Reynolds number as for our results [Chhabra and Agarwal (1999)]. As can be seen in figure 4.4 is at $Re \sim 1$ the $C_D \sim 40$ and decreases to a constant value of $C_D \sim 5$. In our results is at $Re < 1$ the C_D also decreasing drastically when the Reynolds number increases. From $Re \sim 10$ it reaches a constant value of $C_D \sim 5$, but this is rectangle with smoother edges. As the article of Chabbra suggests increases the constant value that is reached, when the edges become less smooth, so towards a rectangle.

The lift and torque coefficient are in the order of 10^{-13} - 10^{-15} and can be considered 0. This is expected because the flow is simultaneously and the block is exactly in the middle. The flow over and under the block are the same so neither lift nor torque is expected. The drag, lift and torque coefficients are computed as explained in section 3.1 and are plotted against the corresponding Reynolds numbers in figure 4.3. The values of the data plotted in figure 4.3 are stated in table 4.1.

4.2 $h/H = 1/4$

The block has been moved closer to the wall to a quarter of the height of the pipe and the results of this simulation are written in this section. In figures 4.5 and 4.6 are the streamlines around the block at $h/H = 1/4$ plotted for different Reynolds numbers.

The dimensionless coefficients associated with these different flow-streams are given in table 4.2 and they have been plotted against the Reynolds number in figure 4.7.

The gap after the rectangle is as with $h/H = 1/2$ getting bigger for higher Reynolds number. Due to the flow moving faster on the upper side of the block than on the lower side, the flow moves straight forward on the upper side and the lower side gradually moves to higher velocity. The drag seems to move slower towards a constant value of $C_D \sim 0.3$. The lift forces changes direction around $Re \sim 30$. The lift on a sphere changes direction at $Re \sim 50$ [Holzer and Sommerfeld (2009)], so the obtained value seems to be

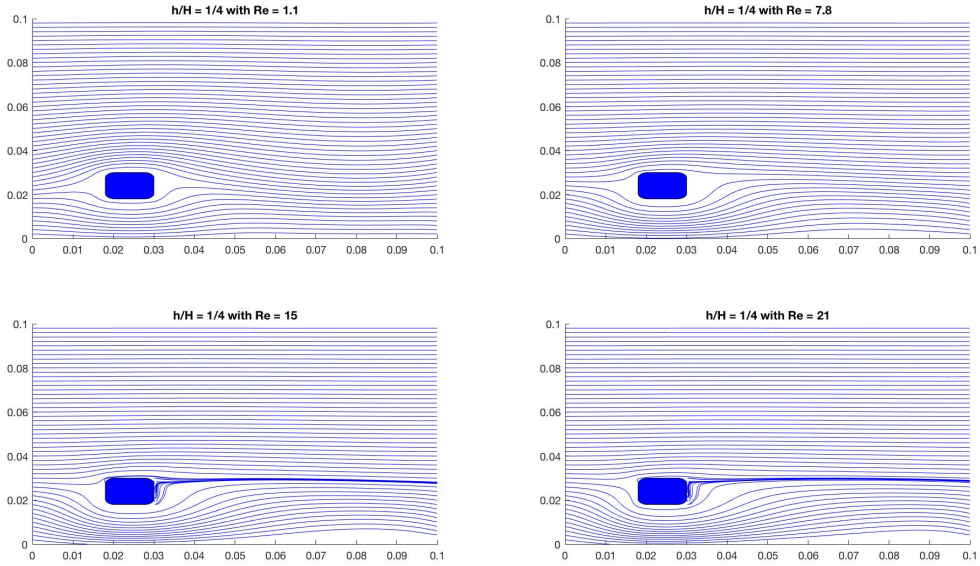

Figure 4.5: Flow simulation with $h/H = 1/4$ (1)

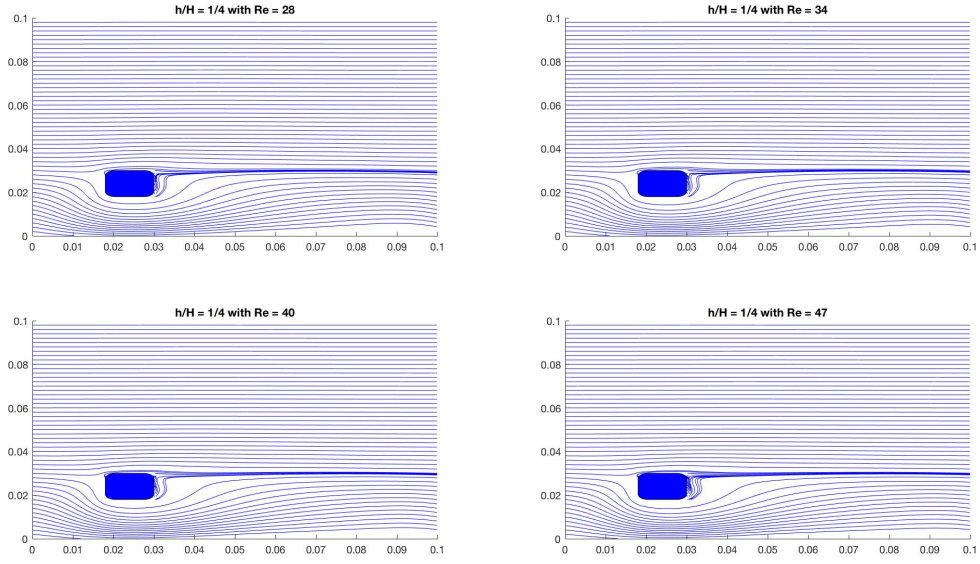

Figure 4.6: Flow simulation with $h/H = 1/4$ (2)

Re	1.1	7.8	15	21	28	34	40	47
CD	12	1.9	1.1	0.79	0.62	0.52	0.44	0.39
CL	1.5	0.39	0.14	5.3e-2	1.2e-2	-1.1e-2	-2.3e-2	-3.1e-2
CT	3.7e2	34	14	8.6	6.1	4.8	4.0	3.5

Table 4.2: Dimensionless coefficients with Reynolds number at $h/H = 1/4$

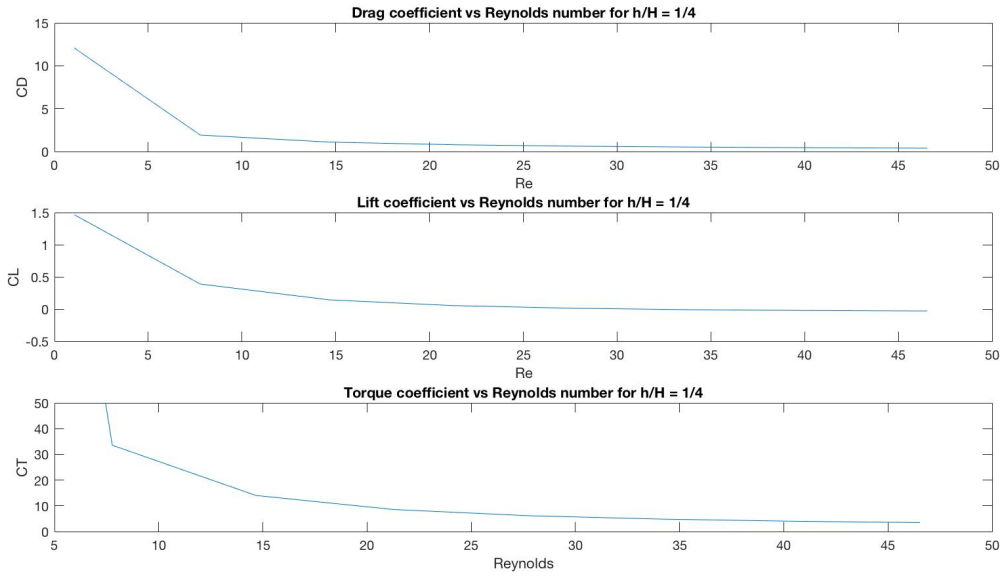

Figure 4.7: Flow simulation with $h/H = 1/4$

Re	1.2	11	21	31	41	51	61
CD	7.0	0.96	0.54	0.39	0.30	0.25	0.21
CL	0.82	0.34	0.21	0.15	0.11	8.4e-2	6.7e-2
CT	4.2e2	43	20	12	8.5	6.4	5.1

Table 4.3: Dimensionless coefficients with Reynolds number at $h/H = 1/8$

of the same order. As can be seen in figure 4.8 is the pattern of decreasing of our torque coefficient in the same way between a fibre and ellipsoid 1 as in research of Zastawny et al. [et al. (2012)].

4.3 $h/H = 1/8$

The results of the block at an eighth of the pipe height are put in this section. The flow profiles are depicted in figures 4.9 and 4.10 for different Reynolds numbers. The dimensionless coefficients are plotted against the Reynolds number in figure 4.11 and the corresponding values can be found in table 4.3.

Something different is happening now than at the other two positions. The vortex shredding is here not directly behind the block but it is created a bit further away. At $Re = 1.2$ this is barely noticeable but when the Reynolds number increases the circulation also increases. As shown by Bhattacharyya and Maiti is this the recirculation region that forms on the wall more downstream of the obstacle [Bhattacharyya and Maiti (2004)]. This happens at all Reynolds numbers, although at higher Reynolds number is the region larger. The Drag coefficient is again decreasing in the same way as at precious positions for increasing the Reynolds number. Now it seems to go to a constant of $C_D \sim 0.2$. The rectangle is now closer to the wall than at $h/H = 1/4$ and the C_L should thus be higher [Holzer and Sommerfeld (2009)]. This is not the case for $Re \sim 1$, but at higher Reynolds it holds. The torque coefficient is decreasing in the same manner as at $h/H = 1/4$, only a fraction higher values.

4.4 Moving top wall

The driving force of the flow was in sections 4.1 - 4.3 the pressure difference present in the pipe. In this section and section 4.5 a shear flow will be considered, the flow will move as a consequence of one or two moving walls. In this section will the upper wall move. The flow has been simulated with different velocities of the wall. To be able to compare the different velocities, the block has been on the same height

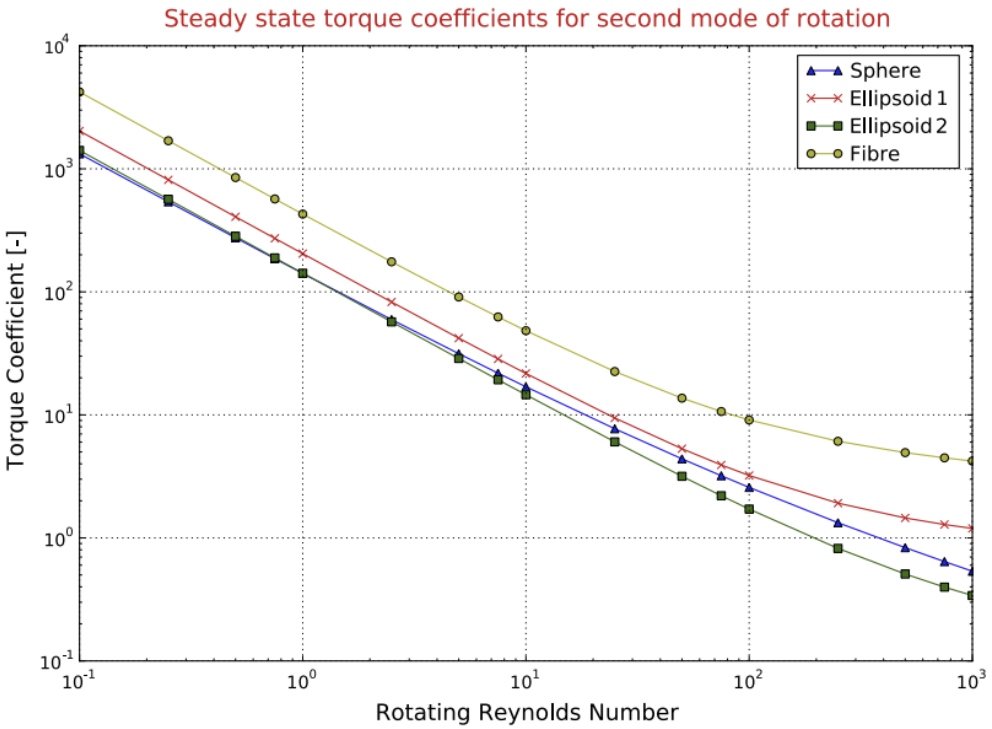

Figure 4.8: Results for torque coefficient from Zastawny et al.

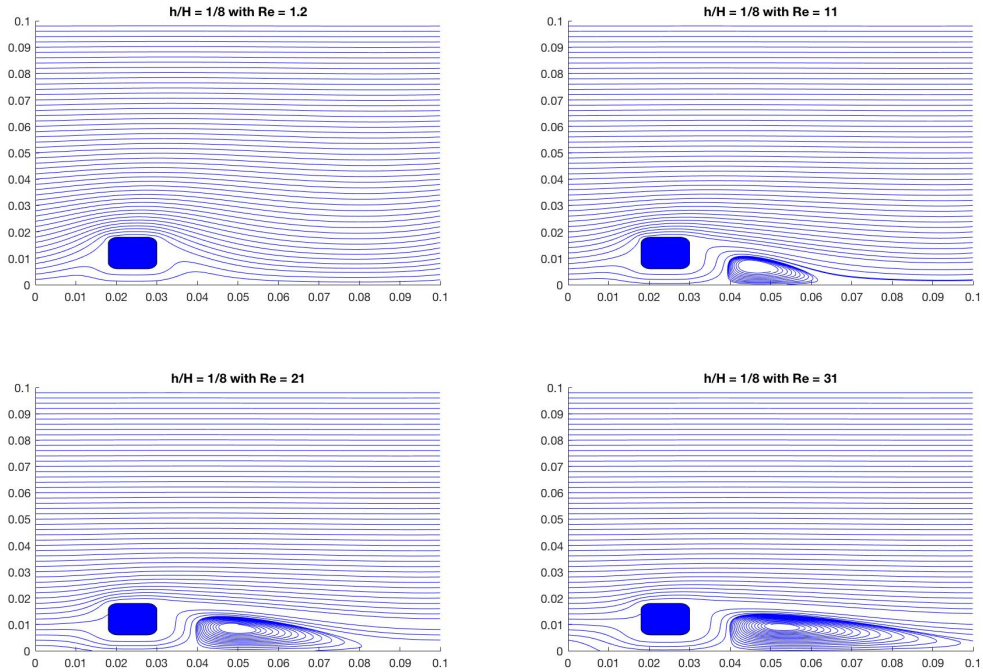

Figure 4.9: Flow simulation with $h/H = 1/8$ (1)

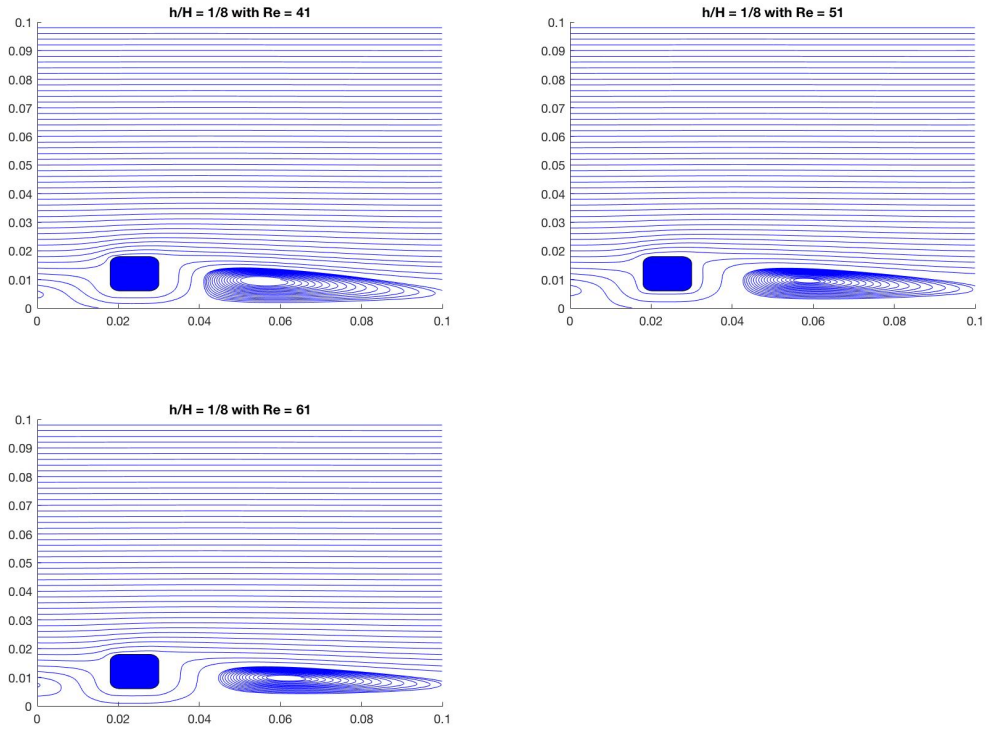

Figure 4.10: Flow simulation with $h/H = 1/8$ (2)

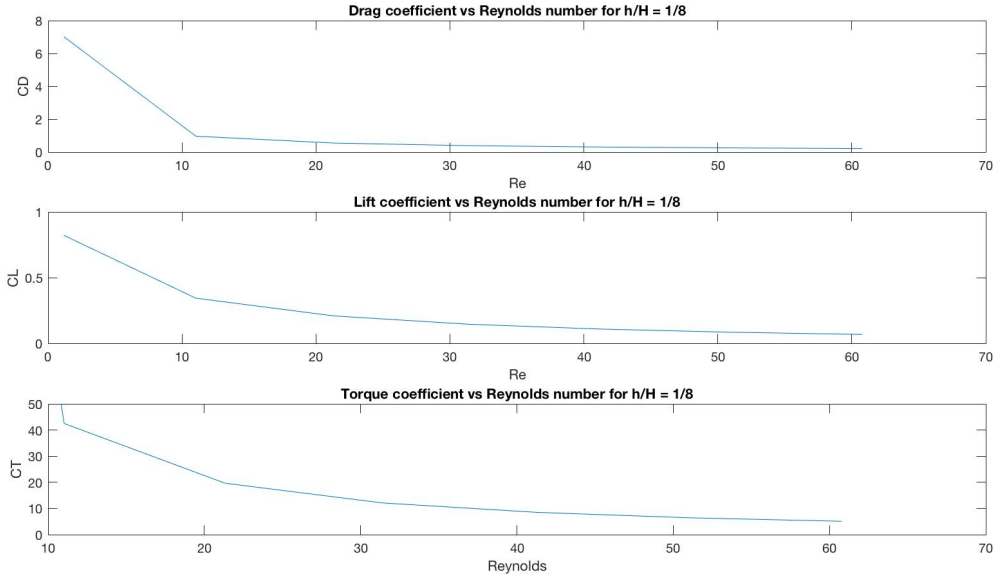

Figure 4.11: Dimensionless coefficients for $h/H = 1/8$

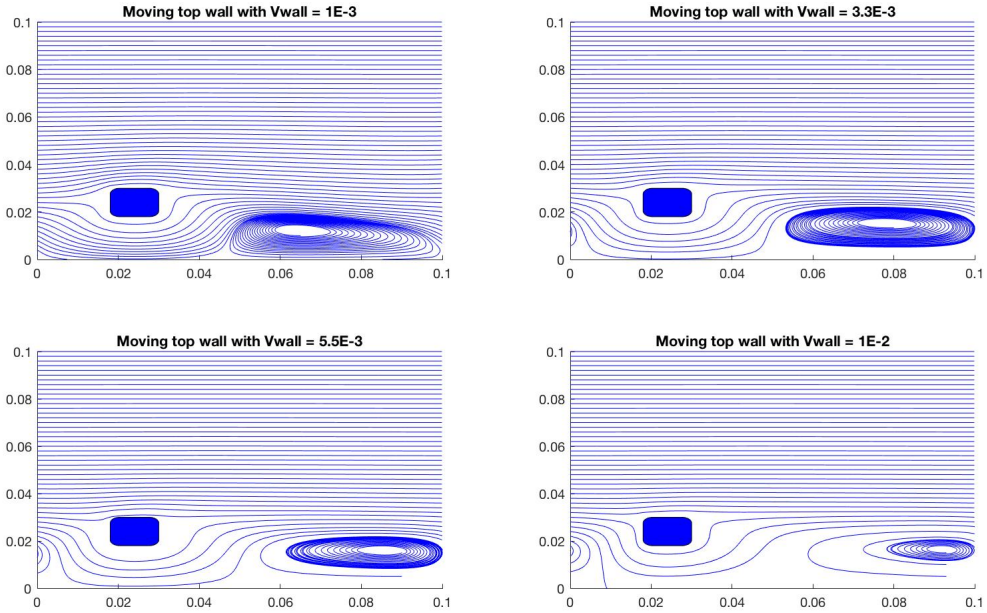

Figure 4.12: Flow simulations with moving top wall

Vwallt	1.0e-3	3.3e-3	5.5e-3	1.0e-2
CD	2.2	0.76	0.47	0.28
CL	0.46	0.15	6.7e-2	1.5e-2
CT	93	25	14	7.2

Table 4.4: Dimensionless coefficients with moving top wall

ratio	0.10	0.33	0.55	1.0
CD	6.9	4.6	3.1	1.2
CL	1.9	1.2	0.73	0.20
CT	-1.6e2	-89	-53	-13

Table 4.5: Dimensionless coefficients with moving both walls

in the pipe, at $h/H = 1/4$. The streamlines of the simulations made for these cases are depicted in figure 4.12. The dimensionless coefficients have been plotted against the velocity of the top wall. This is plotted in figure 4.13 and the data for this graph can be found in table 4.4.

The streamlines here look very much like the case of the rectangle at an eighth of the pipe height. There is re-circulation a little further from the block happening at the wall. This shifts further from the block for increasing velocity of the moving wall. Drag, lift and torque coefficients have the same pattern of decreasing. The order of C_L for shear flow is found in the results of Lee et al. as can be seen in figure 4.14 [Lee and Balachandar (2010)].

4.5 Moving both walls

In this section the results of a simulation with the two walls moving will be given. The ratio between both velocities is considered as reference. The ratio taken is $\frac{V_{wallt}}{V_{wallb}}$. The streamlines for the different ratios are plotted in figure 4.15. The dimensionless coefficients are plotted against the ratio of velocities and these graphs are plotted in figure 4.16 with corresponding data in table 4.5.

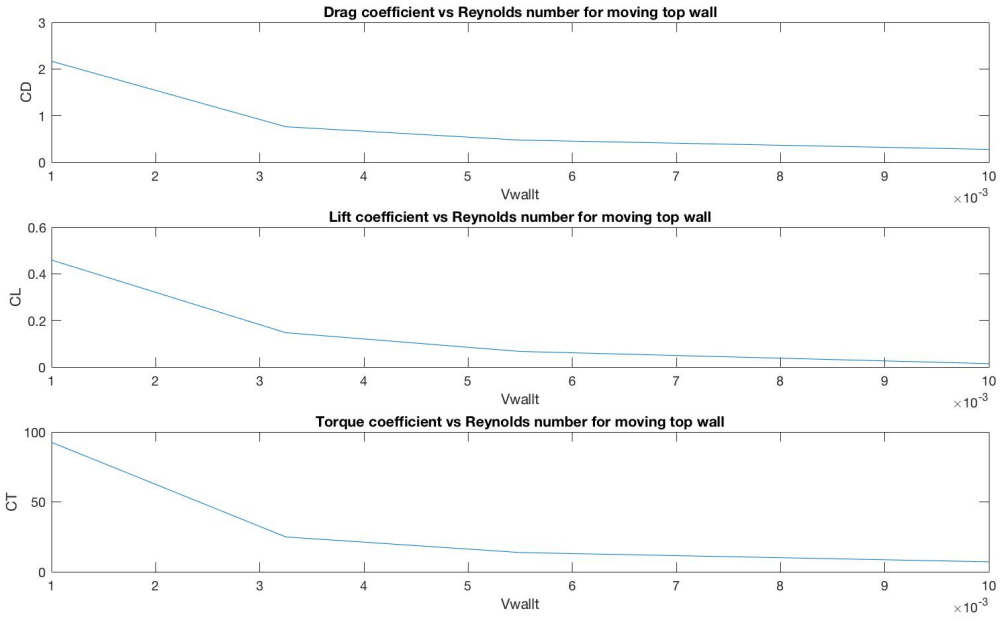

Figure 4.13: Dimensionless coefficients for moving top wall

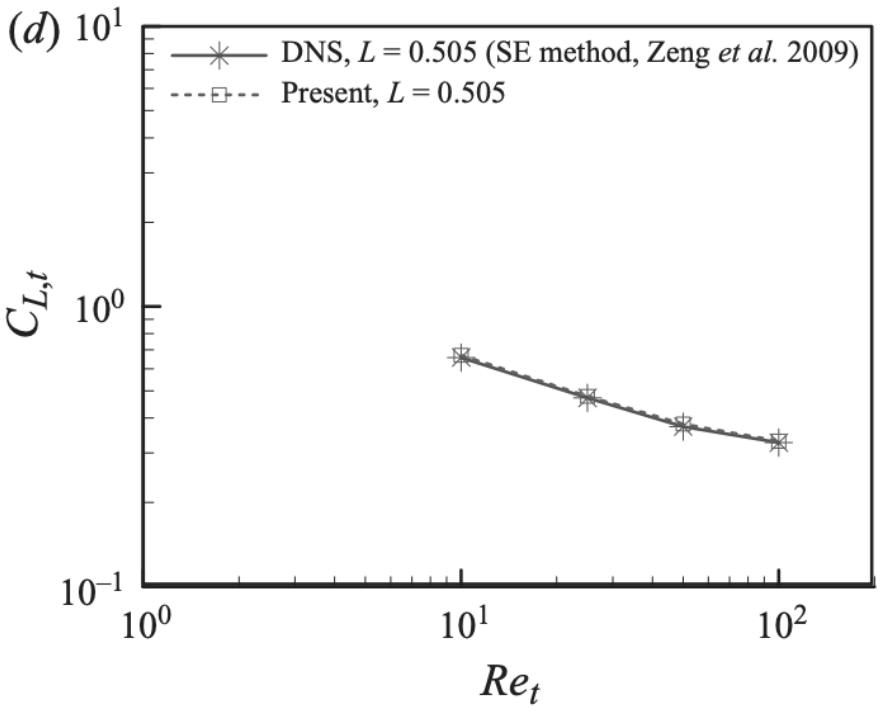

Figure 4.14: C_L decreasing in a shear flow
Lee and Balachandar (2010)

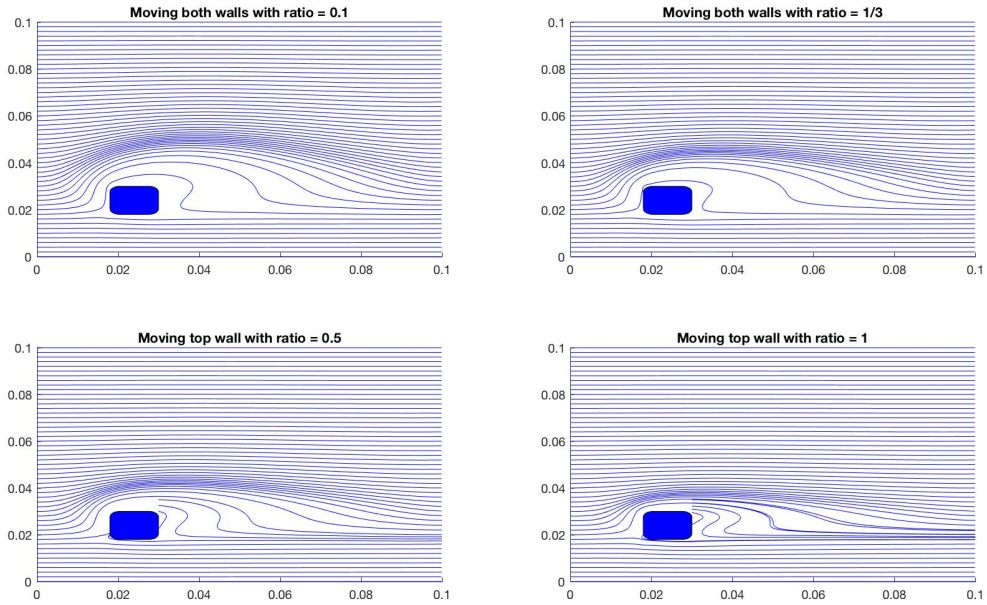

Figure 4.15: Flow simulations with moving both walls

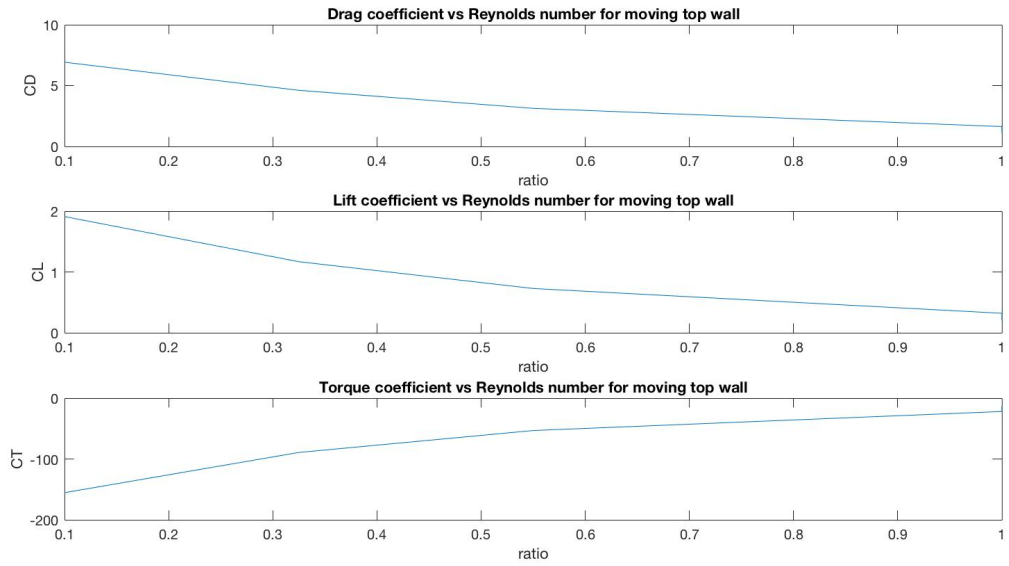

Figure 4.16: Dimensionless coefficients for moving both walls

Because of the lower wall moving faster than the upper wall, the flow is going straight forward at the lower part of the block and gets at the upper side more vertical velocity. Quite a big area with less flow velocity is created. With the ratio moving towards 1, the velocity on the upper side of the rectangle becomes higher and the gap behind the block with less velocity shrinks. Drag and lift coefficient decrease for increasing velocity. The lift coefficient is an order higher than for the shear flow with one wall moving. A remarkable thing is that now the torque coefficient is negative so rotating anti-clockwise. This is due to the fact that the bottom wall is moving faster than the upper wall. As a consequence is on the down side of the block the flow moving faster than on the upper side. This gives on the down side a stronger force to the right, which results in the observed anticlockwise torque.

Chapter 5

Conclusion and recommendations

5.1 Conclusions

The goal of this project was to model a flow past a rectangle near a wall. This is done by writing a Matlab code from scratch which could simulate the pipe flow. Then the rectangle is placed inside the pipe and the flow around it could be studied. Forces present throughout the flow and acting on the rectangle could be calculated and dimensionless coefficients could be determined. The results of the different simulations that ran with the code are stated in chapter 4 and will be discussed in this chapter.

The drag coefficients decrease in the same way to a constant level for increasing Reynolds number, conform the literature that was found. The constant level until where it goes, seems to decrease when the rectangle is placed closer to the wall. The lift and torque were zero when the rectangle was placed in the middle of the pipe. This is evident because of simultaneous flow, but is proof that the code is working correctly. The lift at $h/H = 1/4$ decreases and changes direction in a way like it was found in literature. The height of the rectangle at $h/H = 1/8$ which should be compared to $L/d = 0.67$ was not considered in literature. As was found should lift increase when the rectangle approaches the wall. This is conform the results that were obtained, except from lift at $Re \sim 1$. The torque is for $h/H = 1/4$ and $h/H = 1/8$ decreasing in the same way as in the literature. Closer to the wall the values stay higher, which should be the result of higher lift on the rectangle.

The cases of shear flow that were investigated are with one and two walls moving. The drag is decreasing in a way that was expected for increasing Reynolds number. Lift also decreases as supposed, except that with both walls moving the lift coefficient is an order higher than expected. The bottom wall is moving faster than the top wall and exerts a high lift on the rectangle.

Then we can finally turn back to the research question we were trying to answer in this project:

"Can we simulate a flow past a rectangle and does it show realistic behaviour?"

The code that is developed simulates a flow through a pipe. Different parameters can be varied so the code can be used for many different cases. The output for all different cases is very similar to what in the literature could be found. Plots with flow streams that were made have all aspects that real flows would have. Everything that is found points towards that our Matlab code is working correctly. Hence the answer on the research question is: yes we can and yes it does.

5.2 Recommendations

The code can be improved in many ways. With my chemical background I have never learned seriously programming. So getting the right answer was my prior task when writing the code. This came clearly back in the run time of the code. With a mesh of $50 * 50$ grid cells a single simulation until convergence took already quite a day. Increasing the number of grid points was paid with a lot of extra simulation time. A simulation with more than $100 * 100$ grid cells takes more than a week to converge. In the little time available for obtaining results, a smaller grid was chosen to be able to investigate multiple scenarios. With less grid cells the accuracy of the obtained numbers decreases[Harichandan and Roy (2012)].

Major improvement to the code is to use less loops and more matrix multiplications. Matlab is a lot faster in using matrix and vector operations than in loop-based code [The MathWorks (2018)]. In some cases I only knew how to get the right answer by using a loop, where probably a matrix or vector could also be used. In these cases this was at the expense of the run time.

A good example where run time can be improved is the creation of the matrix A for the Poisson equation. I came to the conclusion when writing this thesis that some unnecessary matrix operations are executed. The deleting of rows, transforming and reshaping could be canceled out by just calling the specific rows and columns all 0.

If the code would be improved and the running time for convergence would decrease, then could be experimented with finer grids. In a study for grid sizes has a body with 80 nodes an error of 2% on the drag force and increasing the number of nodes on the body to 160 reduces the error to 1% [Harichandan and Roy (2012)]. The body in this project only consists of 36 nodes. The paper does not have errors for less than 80 nodes on the body, but if the error would be extrapolated to 36 could the error be huge. So improving the grid size would result in much more accurate values.

Besides all changes in the code that could be done, are improvements on the results which the code produces. In the whole project was the block in the flow a rectangle. Our short term goal was to simulate that. Now that is reached could be looked at our long term of investigating the behaviour of non-spherical particles. If the rectangle would be changed to different configurations, then could data be produced which is more related to actual events.

Bibliography

S. Bhattacharyya and D. Maiti. Shear flow past a square cylinder near a wall. *International Journal of Engineering Science*, 42:2119–2134, 2004. doi: https://doi.org/10.1016/j.ijengsci.2004.04.007.

R.P. Chhabra and L. Agarwal. Drag on non-spherical particles: an evaluation of available methods. *Powder Technology*, 101:288–295, 1999. doi: https://doi.org/10.1016/S0032-5910(98)00178-8.

M. Zastawny et al. Derivation of drag and lift force and torque coefficients for non-spherical particles in flows. *International Jounral of Multiphase Flow*, 39:227–239, 2012. doi: https://doi.org/10.1016/j.ijmultiphaseflow.2011.09.004.

A.L. Garcia and W. Wagner. Time step truncation error in direct simulation monte carlo. *Physics of Fluids*, 12:2621–2633, 2000. doi: https://doi.org/10.1063/1.1289691.

A.B. Harichandan and A. Roy. Numerical investigation of flow past single and tandem cylindrical bodies in the vicinity of a plane wall. *Journal of Fluids and Structures*, 33:19–43, 2012. doi: https://doi.org/10.1016/j.jfluidstructs.2012.04.006.

A. Holzer and M. Sommerfeld. Lattice boltzmann simulations to determine drag, lift and torque acting on non-spherical particles. *Computers and fluids*, 38:572–589, 2009. doi: https://doi.org/10.1016/j.compfluid.2008.06.001.

T. Holzmann. *Mathematics, Numerics, Derivations and OpenFOAM(R)*. www.holzmann-cfd.de, 2016.

H. Lee and S. Balachandar. Drag and lift forces on a spherical particle moving on a wall in a shear flow at finite re. *Journal of Fluid Mechanics*, 657:89–125, 2010. doi: https://doi.org/10.1017/S0022112010001382.

R. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

Inc. The MathWorks. Techniques to improve performance, 2018. URL `https://nl.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html`.

E.F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer-Verlag, 1999.

H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics*. Pearson Education Limited, 2007.

Wikipedia. Finite volume method, 2018. URL `https://en.wikipedia.org/wiki/Finite_volume_method`.

Appendix A

Appendices

A.1 Finite volume method

To calculate the forces acting on a particle in a pipe flow, then the flow needs to be simulated first. Therefor the geometry of the pipe is divided into small squares, the mesh. Via the finite volume method the flow can be simulated.

> "The finite volume method (FVM) is a method for representing and evaluating partial differential equations in the form of algebraic equations [LeVeque (2002) ; Toro (1999)]. Similar to the finite difference method or finite element method, values are calculated at discrete places on a meshed geometry. "Finite volume" refers to the small volume surrounding each node point on a mesh." [Wikipedia (2018)]

A scalar variable is stored in every node point, in this case the pressure. To prevent wrong influence of the pressure in case of a 'checker-board' pressure field, a staggered grid is used for the velocity components. The x-velocity (u) and y-velocity (v) are placed at the face of each cell between two horizontally and vertically aligned nodes respectively. In this way there is a pressure gradient $\delta p/\delta x$ over all x-velocities and pressure gradient $\delta p/\delta y$ over all y-velocities. Now, the discretized momentum equation can be solved for every cell which will be explained in Section A.2. The visualization of this mesh is given in Figure 2.1[Versteeg and Malalasekera (2007)], where the big dots are the central nodes containing the pressure, the horizontal arrows the x-velocity and the vertical arrow the y-velocity.

For the simulation of the flow it is necessary to solve the momentum equation in every cell in the pipe. For solving a cell, the pressures and velocities of the surrounding cells are required. Since for the cells at the wall there would be a problem, ghost points are used. These ghost points are one row of cells on the other side of the wall. Here can velocities and pressure nodes be stored. In this code only the x-velocity is stored in the ghost points at the walls. The velocity in both x and y direction is 0 at the wall. The v at the wall lies in case of the top and bottom wall at the North and South side of each cell respectively. It is obvious that the value of all those v's should be 0 since it lies in the wall. It is different for the u because it does not lie exactly at the wall. Here is where the ghost points are used for. Since the value of u at the wall should be equal to zero, the value of u in the ghost points is exactly chosen so that the average of the two u's at both side of the wall is zero. So,

$$\frac{u_{wall} + u_{ghostpoint}}{2} = 0 \tag{A.1}$$

which can an also be stated as:

$$u_{wall} = -u_{ghostpoint} \tag{A.2}$$

where u_{wall} is the x-velocity in a cell at a wall and $u_{ghostpoint}$ the x-velocity in the ghost point at the other side of the wall.

The pressure gradient at the wall is also zero. Therefore could the ghost points be used, but in this code this is done in another way as will be explained in Section A.2. Ghost points are not only used at the walls, but also for the cells lying in the block. The block in this project counts for the particle moving through the pipe. As at the walls, the velocities and pressure gradients at the edges of the block are zero. How this is used will be further clarified in Section A.2.

A.2 SIMPLE algorithm

In this project, the SIMPLE algorithm is used for simulating the flow in the pipe. SIMPLE is an acronym for Semi-Implicit Method for Pressure Linked Equations. This is an iterative method which starts with guessing a pressure field p^* and guessing velocity fields u^* and v^*, this is explained in Subsection A.2.1. With the guessed pressure field, the discretized momentum equation can be solved yielding a new velocity field. This is described is Subsection A.2.2. Because continuity must be satisfied, after updating the velocity fields, the pressure field needs to be updated as well, which will be clarified in Subsection A.2.3. With the pressure correction p' that follows from the previous step, the velocity field can be updated. All these steps are repeated until convergence is reached.

A.2.1 Initiation

First thing that needs to be done is define the parameters and the boundary conditions. After putting in the desired numbers, the code has to be initiated. This is done by creating a matrix with a guessed: pressure field p^* , x-velocity field u^* and y-velocity field v^*. With defining the parameters, a certain grid size is defined. The matrix for p^* is exact as large as the number of gridpoint chosen, the matrix for u^* is as large as the grid points with addition of two rows for the ghost points at both walls and the matrix for v^* is as large as the grid points with addition of one row because the y-velocities are defined at the top and bottom side of each cell. The initial guesses in this project are for all fields always zero, because the outcomes of these fields change for different parameters is this the best guess.

A.2.2 Momentum equation

First thing that needs to be done after the initial guesses are made, is updating the velocity fields. This is done via solving the conserved momentum equation. The conservation of momentum states that for a specific volume[Holzmann (2016)]:

$$\begin{aligned}[\text{ rate of momentum accumulation }] = [\text{ rate of momentum entering the volume }] \\ - [\text{ rate of momentum leaving the volume }] + [\text{ sum of forces that act on the volume }]\end{aligned} \tag{A.3}$$

So, besides momentum entering and leaving the volume there is change in momentum caused by forces acting on the surface of the volume. These forces consists of pressure and viscous forces. The entering and leaving of momentum is considered as convective acceleration. When equation A.3 is applied to a small volume, in this 2D case a little square, a more detailed equation for momentum conservation can be obtained. The rate of momentum accumulation is given by the change of momentum in this volume over time:

$$\frac{\partial}{\partial t}\rho\vec{U}\mathrm{d}x\mathrm{d}y \tag{A.4}$$

where ρ is the density of the fluid and U the velocity of the fluid. The flow used in this project is considered as incompressible, so ρ is assumed to be a constant. $\mathrm{d}x$ is the width and $\mathrm{d}y$ the height of the 2D volume.

The convective acceleration is the momentum entering and leaving the volume. As the accumulation is considered in equation A.3 the entering of momentum contributes to the total and is positive. The leaving of momentum decreases the total and is thus negative. The convection happens through the surface of the volume. For this 2D case, the entering of momentum at the left and bottom side of the square. Leaving of momentum, and thus negative, happens at right and top side. X-momentum is the mass times the velocity in the x-direction u. The rate at which momentum is entering the square is divided into the rate of mass entering the volume and u at that place. This gives for the left side of the square

$$(\rho\vec{u}\vec{u})|_x\,\mathrm{d}y \tag{A.5}$$

and for leaving at the right side

$$-\,(\rho\vec{u}\vec{u})|_{x+\mathrm{d}x}\,\mathrm{d}y \tag{A.6}$$

Combining A.5 and A.6 gives the convection of x-momentum in the x-direction:

$$(\,(\rho\vec{u}\vec{u})|_x - (\rho\vec{u}\vec{u})|_{x+\mathrm{d}x})\mathrm{d}y \tag{A.7}$$

For the x-momentum entering en leaving through the bottom and top side, it is still about the rate of mass entering with velocity u. The rate of mass entering the bottom face of the square is

$$(\rho\vec{v})|_y \, \mathrm{d}x \tag{A.8}$$

so the density multiplied with the velocity in y-direction v and the surface area $\mathrm{d}x$.
This gives for the convection of momentum through the south face of the square:

$$(\rho\vec{v}\vec{u})|_y \, \mathrm{d}x \tag{A.9}$$

and through the north face:

$$-\ (\rho\vec{v}\vec{u})|_{y+\mathrm{d}y} \, \mathrm{d}x \tag{A.10}$$

The combination of A.9 and A.10 yields the convection of x-momentum in the y-direction:

$$(\,(\rho\vec{v}\vec{u})|_y - (\rho\vec{v}\vec{u})|_{y+\mathrm{d}y})\mathrm{d}x \tag{A.11}$$

For the y-momentum the mass should be multiplied with v, the velocity in the y-direction. The rate of mass flowing is here the same as with the x-momentum. So like the derivations for A.7 and A.11 the equations for the convection of y-momentum through the surfaces of the volume can be derived. For convection in the x-direction this is

$$(\,(\rho\vec{u}\vec{v})|_x - (\rho\vec{u}\vec{v})|_{x+\mathrm{d}x})\mathrm{d}y \tag{A.12}$$

and in the y-direction:

$$(\,(\rho\vec{v}\vec{v})|_y - (\rho\vec{v}\vec{v})|_{y+\mathrm{d}y})\mathrm{d}x \tag{A.13}$$

Then, the last part of equation A.3, the sum of forces that act on the volume, can be clarified. On the volume act two kind of forces, viscous forces and pressure forces. First will the viscous forces be explained and then the pressure forces. Viscous forces on the volume are the result of velocity gradients in the fluid and near the block. The viscous forces on the square in the x-direction are τ_{xx} perpendicular to the left en right side and τ_{yx} parallel to the top and bottom face. Because in this project an incompressible flow is considered [Holzmann (2016)]:

$$\tau_{xx} = 2\mu\frac{\partial u}{\partial x} \tag{A.14}$$

and

$$\tau_{yx} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \tag{A.15}$$

The τ_{xx} on the right side and τ_{yx} on the top side are considered positive to the right and as a matter of definition τ_{xx} on the left side and τ_{yx} on the bottom side are then to the left and considered negative. This gives for the east and west face:

$$(\,\tau_{xx}|_{x+\mathrm{d}x} - \tau_{xx}|_x)\mathrm{d}y \tag{A.16}$$

For the north and south face of the square holds that

$$(\,\tau_{yx}|_{y+\mathrm{d}y} - \tau_{yx}|_y)\mathrm{d}x \tag{A.17}$$

Substituting equation A.14 into equation A.16 yields:

$$2\mu\left(\left.\frac{\partial u}{\partial x}\right|_{x+\mathrm{d}x} - \left.\frac{\partial u}{\partial x}\right|_x\right)\mathrm{d}y \tag{A.18}$$

Filling in equation A.15 in A.17 gives:

$$\mu\left(\left.\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right|_{y+\mathrm{d}y} - \left.\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right|_y\right)\mathrm{d}x \tag{A.19}$$

In the same way the viscous forces in y-direction can be derived. For the tensor perpendicular to the north and south face it holds that

$$\tau_{yy} = 2\mu \frac{\partial v}{\partial y} \tag{A.20}$$

so that like equation A.16 the total force on surface $\mathrm{d}x$ is:

$$2\mu \left(\left. \frac{\partial v}{\partial y} \right|_{y+\mathrm{d}y} - \left. \frac{\partial v}{\partial y} \right|_{y} \right) \mathrm{d}x \tag{A.21}$$

Tensor τ_{xy} parallel to the west and east side of the volume is exactly in the same way worked out as equation A.15, which gives the following force on surface $\mathrm{d}y$:

$$\mu \left(\left. \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right|_{x+\mathrm{d}x} - \left. \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right|_{x} \right) \mathrm{d}y \tag{A.22}$$

Eventually the pressure force will be considered. In this project two special types of flows are used or a combination of both, namely shear flow and pressure driven flow. In the pressure driven flow is a certain pressure gradient present which causes the flow to move. This pressure gradient p_{grad} is import for the computation of the pressure force exerted on a specific volume in the flow or on the block positioned in the flow. p_{grad} is constant decrease in pressure between two grid points. In this project the pressure difference is over a horizontal pipe and so is the p_{grad} only present between two horizontal aligned grid point. p_{grad} has a constant value everywhere found in the pipe. To compute the pressure force exerted on the 2D volume, the pressure gradient acting on the surface must be considered. For the pressure force in the x-direction this is the p_{grad} acting on area $\mathrm{d}y$ plus the pressure correction p'. So this is

$$\left(p|_{x} - p|_{x+\mathrm{d}x} \right) \mathrm{d}y \tag{A.23}$$

which leads to:

$$\left(p_{grad}\mathrm{d}x + p'|_{x} - p'|_{x+\mathrm{d}x} \right) \mathrm{d}y \tag{A.24}$$

The pressure force in the vertical direction only consists of the gradient of p' and no extra p_{grad} because that pressure gradient only works in the horizontal direction. That being the case will lead to a vertical pressure force of

$$\left(p'|_{y} - p'|_{y+\mathrm{d}y} \right) \mathrm{d}x \tag{A.25}$$

Now all parts of equation A.3 are considered, this equation can be written down in more detail. Using equations: A.4 for rate of accumulation, A.7, A.11, A.12 and A.13 for rate of entering and leaving of momentum, A.18, A.19, A.21, A.22, A.24 and A.25 for sum of forces acting on the volume, this yields for the x-momentum:

$$\begin{aligned} \frac{\partial}{\partial t}\rho u \mathrm{d}x\mathrm{d}y = {} & \left((\rho \vec{u}\vec{u})|_{x} - (\rho \vec{u}\vec{u})|_{x+\mathrm{d}x} \right)\mathrm{d}y + \left((\rho \vec{v}\vec{u})|_{y} - (\rho \vec{v}\vec{u})|_{y+\mathrm{d}y} \right)\mathrm{d}x \\ & + 2\mu \left(\left. \frac{\partial u}{\partial x} \right|_{x+\mathrm{d}x} - \left. \frac{\partial u}{\partial x} \right|_{x} \right) \mathrm{d}y + \mu \left(\left. \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right|_{y+\mathrm{d}y} - \left. \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right|_{y} \right) \mathrm{d}x \\ & + \left(p_{grad}\mathrm{d}x + p'|_{x} - p'|_{x+\mathrm{d}x} \right) \mathrm{d}y \end{aligned} \tag{A.26}$$

and for the y-momentum:

$$\begin{aligned} \frac{\partial}{\partial t}\rho v \mathrm{d}x\mathrm{d}y = {} & \left((\rho \vec{u}\vec{v})|_{x} - (\rho \vec{u}\vec{v})|_{x+\mathrm{d}x} \right)\mathrm{d}y + \left((\rho \vec{v}\vec{v})|_{y} - (\rho \vec{v}\vec{v})|_{y+\mathrm{d}y} \right)\mathrm{d}x \\ & + 2\mu \left(\left. \frac{\partial v}{\partial y} \right|_{y+\mathrm{d}y} - \left. \frac{\partial v}{\partial y} \right|_{y} \right) \mathrm{d}x + \mu \left(\left. \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right|_{x+\mathrm{d}x} - \left. \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right|_{x} \right) \mathrm{d}y \\ & + \left(p'|_{y} - p'|_{y+\mathrm{d}y} \right) \mathrm{d}x \end{aligned} \tag{A.27}$$

Further working out gives then for the momentum in the x-direction:

$$\frac{\partial}{\partial t}\rho u = -\frac{\partial}{\partial x}(\rho\vec{u}\vec{u}) - \frac{\partial}{\partial y}(\rho\vec{v}\vec{u}) + \frac{\partial}{\partial x}\left[2\mu\frac{\partial u}{\partial x}\right] + \frac{\partial}{\partial y}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] + p_{grad} - \frac{\partial p'}{\partial x} \tag{A.28}$$

and for momentum in the y-direction:

$$\frac{\partial}{\partial t}\rho v = -\frac{\partial}{\partial x}(\rho\vec{u}\vec{v}) - \frac{\partial}{\partial y}(\rho\vec{v}\vec{v}) + \frac{\partial}{\partial y}\left[2\mu\frac{\partial v}{\partial y}\right] + \frac{\partial}{\partial x}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] - \frac{\partial p'}{\partial y} \tag{A.29}$$

Combination of A.28 and A.46 is the Navier-Stokes equation:

$$\frac{\partial}{\partial t}(\rho\vec{U}) = -\nabla\cdot(\rho\vec{U}\otimes\vec{U}) + \nabla\cdot\tau - \nabla p \tag{A.30}$$

How these equations are implemented and used in the code will be further clarified in chapter 3.

A.2.3 Continuity equation

Besides solving the momentum balance, the mass balance should also be taken into account. This mass balance is also known as the continuity equation. In the SIMPLE algorithm is first the momentum balance solved, with the continuity equation follows an pressure correction field. With this field can velocities be updated, which all will be explained in chapter 3. The mass balance over an arbitrary volume is

$$[\text{ rate of mass entering the volume }] - [\text{ rate of mass leaving the volume }] = 0 \tag{A.31}$$

This gives for a 2D volume that

$$\frac{\partial}{\partial t}\Big(m|_x - m|_{x+\mathrm{d}x} + m|_y - m|_{y+\mathrm{d}y}\Big) \tag{A.32}$$

which can be further worked out to:

$$\rho\ u|_x\,\mathrm{d}y - \rho\ u|_{x+\mathrm{d}x}\,\mathrm{d}y + \rho\ v|_y\,\mathrm{d}x - \rho\ v|_{y+\mathrm{d}y}\,\mathrm{d}x = 0 \tag{A.33}$$

This gives that

$$-\rho\mathrm{d}u\mathrm{d}y - \rho\mathrm{d}v\mathrm{d}x = 0 \tag{A.34}$$

which eventually leads to the continuity equation:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{A.35}$$

A.3 Discretized momentum equation

In the initiating step a field for the x-velocity and a field for the y-velocity with on every grid point a guessed velocity is created. This guessed velocity is then called vx for the x-direction and vy for the y-direction. With solving the discretized momentum balance these guessed velocities will be updated to u^* and v^* for x- and y-direction respectively. The momentum balance as in equation A.28 will be discretized for a random place on the grid as in figure 2.1. The nodes in this figure for x-velocities, y-velocities and for pressure are denoted with small and/or with capital letter 'i' and 'j'. In the figure u is with small i and capital J, v with capital I and small j, P with both capital I and J. In the code the three different fields are put in three different matrices. The places of all values are in the same way positioned as in figure 2.1. Because this chapter is about explanation of the code, the notation as in the code will be used here. So it is taken into account that the matrix with x-velocities has two extra rows for storing the velocities of the ghost points and the matrix with y-velocities has one row extra. For clarity will equation

A.28 here be stated once again and will then be discretized part by part. So momentum balance for the x-direction is

$$\frac{\partial}{\partial t}\rho u = -\frac{\partial}{\partial x}(\rho\vec{u}\vec{u}) - \frac{\partial}{\partial y}(\rho\vec{v}\vec{u}) + \frac{\partial}{\partial x}\left[2\mu\frac{\partial u}{\partial x}\right] + \frac{\partial}{\partial y}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] + p_{grad} - \frac{\partial p'}{\partial x} \tag{A.36}$$

In the first part is the update of the guessed velocity vx to the new velocity u^* present:

$$\frac{\partial}{\partial t}\rho u \rightarrow \frac{\rho}{dt}(u^*(i,j) - vx(i,j)) \tag{A.37}$$

Then the two convective parts can be discretized. For the convection in the horizontal direction this is

$$-\frac{\partial}{\partial x}(\rho\vec{u}\vec{u}) \rightarrow \frac{\rho}{dx}\Big((0.5(vx(i,j-1) + vx(i,j)))^2 - (0.5(vx(i,j) + vx(i,j+1)))^2\Big) \tag{A.38}$$

and for vertical direction this gives:

$$\begin{aligned}-\frac{\partial}{\partial y}(\rho\vec{v}\vec{u}) \rightarrow \frac{\rho}{dy}(&0.5(vy(i-1,j-1) + vy(i-1,j)) * 0.5(vx(i-1,j) + vx(i,j)) \\ &- 0.5(vy(i,j-1) + vy(i,j)) * 0.5(vx(i,j) + vx(i+1,j)))\end{aligned} \tag{A.39}$$

Taking the 0.5 out yields the discretized form for the convection in vertical direction:

$$\begin{aligned}-\frac{\partial}{\partial y}(\rho\vec{v}\vec{u}) \rightarrow 0.25\frac{\rho}{dy}(&(vy(i-1,j-1) + vy(i-1,j)) * (vx(i-1,j) + vx(i,j)) \\ &- (vy(i,j-1) + vy(i,j)) * (vx(i,j) + vx(i+1,j)))\end{aligned} \tag{A.40}$$

The viscous forces can be discretized as

$$\frac{\partial}{\partial x}\left[2\mu\frac{\partial u}{\partial x}\right] \rightarrow \frac{2\mu}{dx}\left(\frac{vx(i,j+1) - vx(i,j)}{dx} - \frac{vx(i,j) - vx(i,j-1)}{dx}\right) \tag{A.41}$$

and

$$\begin{aligned}\frac{\partial}{\partial y}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] \rightarrow \frac{\mu}{dy}(&\left(\frac{vx(i+1,j) - vx(i,j)}{dy} + \frac{vy(i,j) - vy(i,j-1)}{dx}\right) \\ &- \left(\frac{vx(i,j) - vx(i-1,j)}{dy} + \frac{vy(i-1,j) - vy(i-1,j-1)}{dx}\right))\end{aligned} \tag{A.42}$$

Working out equations A.41 and A.42 further gives respectively:

$$\frac{\partial}{\partial x}\left[2\mu\frac{\partial u}{\partial x}\right] \rightarrow \frac{2\mu}{dx^2}(vx(i,j+1) - 2vx(i,j) + vx(i,j-1)) \tag{A.43}$$

and

$$\begin{aligned}\frac{\partial}{\partial y}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] \rightarrow \frac{\mu}{dy}(&\frac{vx(i+1,j) - 2vx(i,j) + vx(i-1,j)}{dy} \\ &+ \frac{vy(i,j) - vy(i,j-1) - vy(i-1,j) + vy(i-1,j-1)}{dx})\end{aligned} \tag{A.44}$$

The pressure force is the last part of the Navier-Stokes equation that has to be considered. The pressure is stored in the center nodes and is thus applicable for the x- and y-direction. The pressure force consist

of the pressure gradient p_{grad} and the pressure correction. The p_{grad} is a constant and will not change. However the pressure correction will change and therefore is chosen to store the pressure gradient in the center of the cells as p. The pressure force in the x-direction on a 2D volume can be discretized as

$$p_{grad} - \frac{\partial p'}{\partial x} \rightarrow pgrad + \frac{p(i-1,j-1) - p(i-1,j)}{dx} \tag{A.45}$$

The discretization of the Navier-Stokes equation for the y-direction goes in the same way as for the x-direction. The differential equation will be stated again first and then part by part the discretizations. So the equation was

$$\frac{\partial}{\partial t}\rho v = -\frac{\partial}{\partial x}(\rho \vec{u}\vec{v}) - \frac{\partial}{\partial y}(\rho \vec{v}\vec{v}) + \frac{\partial}{\partial y}\left[2\mu\frac{\partial v}{\partial y}\right] + \frac{\partial}{\partial x}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] - \frac{\partial p'}{\partial y} \tag{A.46}$$

The first part can be discretized as

$$\frac{\partial}{\partial t}\rho v \rightarrow \frac{\rho}{dt}(v^*(i,j) - vy(i,j)) \tag{A.47}$$

the convective parts as

$$\begin{aligned} -\frac{\partial}{\partial x}(\rho \vec{u}\vec{v}) \rightarrow 0.25\frac{\rho}{dx}((vy(i,j-1) + vy(i,j)) * (vx(i,j) + vx(i+1,j)) \\ - (vy(i,j) + vy(i,j+1)) * (vx(i,j+1) + vx(i+1,j+1))) \end{aligned} \tag{A.48}$$

and

$$-\frac{\partial}{\partial y}(\rho \vec{v}\vec{v}) \rightarrow \frac{\rho}{dx}\left((0.5(vy(i-1,j) + vy(i,j)))^2 - (0.5(vy(i,j) + vy(i+1,j)))^2\right) \tag{A.49}$$

the viscous forces as

$$\frac{\partial}{\partial y}\left[2\mu\frac{\partial v}{\partial y}\right] \rightarrow \frac{2\mu}{dy^2}(vy(i+1,j) - 2vy(i,j) + vx(i-1,j)) \tag{A.50}$$

and

$$\begin{aligned} \frac{\partial}{\partial x}\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right] \rightarrow \frac{\mu}{dx}(\frac{vx(i+1,j+1) - vx(i,j+1) - vx(i+1,j) + vx(i,j)}{dy} \\ + \frac{vy(i,j+1) - 2vy(i,j) + vy(i,j+1)}{dx}) \end{aligned} \tag{A.51}$$

and at last the pressure force as

$$-\frac{\partial p'}{\partial y} \rightarrow \frac{p(i-1,j) - p(i,j)}{dy} \tag{A.52}$$

In the code will now first all the x-velocities be updated. All the velocities that are in a boundary position will be updated in the way as mentioned in section **??**. So for the continuity of the flow the first and last column are dependent on each other. Also will the values in the ghost points of the walls and the block in the flow be updated so that the velocity at the boundary becomes 0. After updating all u, the y-velocities are updated. Again first for all places not in a boundary position, then first and last row for continuity and eventually making values in the walls or block 0.

A.4 Poisson equation

Besides solving the momentum balances for x- and y-direction the mass balance must also be satisfied. In section A.2.3 the mass balance is already been derived to the continuity equation. Nevertheless this equation is still not convenient for numerical usage. The way it is adapted is characteristic for the SIMPLE

method. How this is done will be explained in this section. The continuity equation as stated in A.34 can be written as:

$$\rho \mathrm{d}y(u_1 - u_2) + \rho \mathrm{d}x(v_1 - v_2) = 0 \tag{A.53}$$

Further can the discretized momentum equations form section A.3 be used here too. The total equation for x-momentum is:

$$\begin{aligned}\frac{\rho}{dt}(u^*(i,j) - vx(i,j)) = \frac{\rho}{dx}\Big((0.5(vx(i,j-1) + vx(i,j)))^2 - (0.5(vx(i,j) + vx(i,j+1)))^2\Big) \\ + 0.25\frac{\rho}{dy}((vy(i-1,j-1) + vy(i-1,j)) * (vx(i-1,j) + vx(i,j)) \\ - (vy(i,j-1) + vy(i,j)) * (vx(i,j) + vx(i+1,j))) \\ + \frac{2\mu}{dx^2}(vx(i,j+1) - 2vx(i,j) + vx(i,j-1)) \\ + \frac{\mu}{dy}(\frac{vx(i+1,j) - 2vx(i,j) + vx(i-1,j)}{dy} \\ + \frac{vy(i,j) - vy(i,j-1) - vy(i-1,j) + vy(i-1,j-1)}{dx}) \\ + pgrad + \frac{p(i-1,j-1) - p(i-1,j)}{dx}\end{aligned} \tag{A.54}$$

and for the y-momentum the total discretized equation is:

$$\begin{aligned}\frac{\rho}{dt}(v^*(i,j) - vy(i,j)) = \frac{\rho}{dx}\Big((0.5(vy(i-1,j) + vy(i,j)))^2 - (0.5(vy(i,j) + vy(i+1,j)))^2\Big) \\ + 0.25\frac{\rho}{dx}((vy(i,j-1) + vy(i,j)) * (vx(i,j) + vx(i+1,j)) \\ - (vy(i,j) + vy(i,j+1)) * (vx(i,j+1) + vx(i+1,j+1))) \\ + \frac{2\mu}{dy^2}(vy(i+1,j) - 2vy(i,j) + vx(i-1,j)) \\ + \frac{\mu}{dx}(\frac{vx(i+1,j+1) - vx(i,j+1) - vx(i+1,j) + vx(i,j)}{dy} \\ + \frac{vy(i,j+1) - 2vy(i,j) + vy(i,j+1)}{dx}) \\ + \frac{p(i-1,j) - p(i,j)}{dy}\end{aligned} \tag{A.55}$$

Now will the main approximation of the SIMPLE algorithm be used. This is that the velocity correction u' and v' are not dependent on surrounding velocities and so only dependent on the pressure correction p'. Hence in the discretized Navier-Stokes equations A.54 and A.55 will all velocity involved parts drop out and this gives

$$\frac{\rho}{dt}(u^*(i,j) - vx(i,j)) = \frac{p(i-1,j-1) - p(i-1,j)}{dx} \tag{A.56}$$

for the x-direction, and for the y-direction this gives:

$$\frac{\rho}{dt}(v^*(i,j) - vy(i,j)) = \frac{p(i-1,j) - p(i,j)}{dy} \tag{A.57}$$

Now these equations can be rewritten for x- and y-velocity respectively:

$$vx(i,j) = u^*(i,j) - \frac{dt * (p(i-1,j-1) - p(i-1,j))}{\rho * dx} \tag{A.58}$$

$$vy(i,j) = v^*(i,j) - \frac{dt * (p(i-1,j) - p(i,j))}{\rho * dy} \tag{A.59}$$

Equations A.58 and A.59 can be substituted into A.53 which yields the following:

$$\rho \mathrm{d}y \left(u^*(i+1,j) - \frac{dt*(p(i,j-1)-p(i,j))}{\rho * dx} - u^*(i+1,j+1) - \frac{dt*(p(i,j)-p(i,j+1))}{\rho * dx} \right) + \rho \mathrm{d}x \left(v^*(i,j) - \frac{dt*(p(i-1,j)-p(i,j))}{\rho * dy} - v^*(i+1,j) - \frac{dt*(p(i,j)-p(i+1,j))}{\rho * dy} \right) = 0 \quad \text{(A.60)}$$

This can be rearranged to:

$$2dt \left(\frac{dy}{dx} + \frac{dx}{dy} \right) p_{i,j} - \frac{dt*dy}{dx} * (p_{i,j-1} + p_{i,j+1}) - \frac{dt*dx}{dy}(p_{i-1,j} + p_{i+1,j}) = \rho dy(u^*_{i+1,j+1} - u^*_{i+1,j}) + \rho dx(v^*_{i+1,j} + v^*_{i,j}) \quad \text{(A.61)}$$

Equation A.61 is now the continuity equation as a Poisson equation for the pressure correction. Every pressure correction is dependent on the pressure corrections surrounding it and the guessed velocities u^* and v^*. For a 'normal' cell, meaning one not next to the wall or block, is dependent on the four cells surrounding it. For a cell in a boundary position this is different. At the face which is touching the wall or block is no pressure difference over both cells. In the code it is programmed that in this case only the values of the other three surrounding cells are used. Meaning that the velocity in the boundary position is 0 and the pressure gradient is also 0.

Because all the pressure corrections are dependent on each other, they have to be solved simultaneously. This is done by creating two matrices with as result after processing a third matrix. The first matrix is the biggest and in the code it is called A. The size of the matrix is dependent on the grid size that is taken. Because every node has a pressure correction and there are $M * N$ squares, the size of A is also $M * N$. The task is now filling in the matrix with correct numbers for the pressure correction. The coefficients standing before the pressure values in equation A.61 are constants. In the code is $2dt \left(\frac{dy}{dx} + \frac{dx}{dy} \right)$ called a, $-\frac{dt*dy}{dx}$ called b and $-\frac{dt*dx}{dy}$ called c. a, b and c are put in matrix A so that the coefficients are convenient with the left part of equation A.61. The second matrix that is made, is a column vector with containing the right hand side of equation A.61. This matrix is called D. After creating the two matrices, the third can be made via:

$$A * p = D \quad \text{(A.62)}$$

The matrix p is like D a column vector, but now containing the values of all the nodes for the pressure correction of the flow. The vector will then be reshaped so that a the pressure field is made again with every p' at the right node.

A.5 Updating variables

After the pressure correction field is obtained, the guessed pressure field and guessed velocity fields can be updated. As of the fact that with the pressure correction the velocity correction can be calculated. This will be explained in this section. Updating the fields is due to under-relaxation. In this project an under-relaxation factor α of 0.6 is chosen. The correction for velocities u' and v' are obtained as follows:

$$u'_{i,j} = \frac{dt}{\rho dx}(p'_{i,j-1} - p'_{i,j}) \quad \text{(A.63)}$$

and

$$v'_{i,j} = \frac{dt}{\rho dy}(p'_{i,j} - p'_{i,j+1}) \quad \text{(A.64)}$$

These velocity corrections are calculated for every point and are then used for updating the velocities as:

$$u^{new}_{i,j} = u^*_{i,j} + \alpha u'_{i,j} \quad \text{(A.65)}$$

$$v_{i,j}^{new} = v_{i,j}^{*} + \alpha v_{i,j}' \tag{A.66}$$

Updating the pressure also involves the under-relaxation factor but with 1-α :

$$p_{i,j} = p_{i,j} + (1-\alpha) p_{i,j}' \tag{A.67}$$

After updating these fields, the convergence is checked. If the flow is not yet converged the program will start over, as it is an iterative process. The updated fields will at the start of the new iteration be the first guessed fields. So that these updated fields will get corrected and then updated again. This process is done until convergence is finally reached.

Appendix B

Code

```
function [CD , CL , CT , reynolds] = bephtoH14(rho)

H = 0.1; L = 0.1;
mu = 1E-3;
%rho = 1E3;
pgrad = 1E-2;
N = 50; M = 50; dt = 10^-2;
niter = 0; nmax = 2E5; alpha = 0.6; dx = L/N; dy = H/M;
err = 1; eps = 10^-3; startconvergencecheck = 2000;

x = linspace(0,L,N+1);
y = linspace(0,H,M+1);

p_prime = zeros(M,N);
u_prime = zeros(M,N);
v_prime = zeros(M-1,N);

% determine particle size and place
L_part = 6; L_dist = 10;
H_part = 6;
H_dist = 10;

% initial gueses
vx = zeros(M+2,N);    %(M+2 , N)
vy = zeros(M+1,N);    %(M+1 , N)
p  = zeros(M,N);


% first determine u-/v- star.
u_star = zeros(M+2,N);
v_star = zeros(M+1,N);

vtoconvergence = zeros(nmax,1);
convergu = zeros((nmax-startconvergencecheck),1);

%% calculate u_star at every point in stag grid
while (niter < nmax & err > eps)
   for j = 2:M+1          % every row without rows in ghost points

       for i = 2:N-1    % every column between two pressure points
       a1 = 2*(vx(j,i+1) - 2*vx(j,i) + vx(j,i-1))/dx; % viscous forces
       a2 = (vx(j+1,i) - 2*vx(j,i) + vx(j-1,i))/dy + (vy(j,i)- vy(j,i-1) - vy(j-1,i)
           + vy(j-1,i-1))/dx; % viscous forces
       a3 = (((0.5*(vx(j,i-1)+vx(j,i)))^2) - ((0.5*(vx(j,i)+vx(j,i+1)))^2))/dx; %
           convective acc
```

```
44        a4 = (0.25*(vx(j,i)+vx(j-1,i))*(vy(j-1,i)+vy(j-1,i-1)) -
              0.25*(vx(j+1,i)+vx(j,i))*(vy(j,i)+vy(j,i-1)))/dy;
45
46        A = (mu/rho)*(a1/dx + a2/dy) + a3 + a4;
47        u_star(j,i) = vx(j,i) + dt*(A + pgrad/rho + (1/dx/rho)*(p(j-1,i-1) -
              p(j-1,i))); % velocity + dt * (A - alpha - pressure gradient)
48        end
49
50     %boundaries vx
51     %start
52
53        a1 = 2*(vx(j,2) - 2*vx(j,1) + vx(j,end))/dx; % viscous forces
54        a2 = (vx(j+1,1) - 2*vx(j,1) + vx(j-1,1))/dy + (vy(j,1)- vy(j,end) - vy(j-1,1)
              + vy(j-1,end))/dx; % viscous forces
55        a3 = (((0.5*(vx(j,end)+vx(j,1)))^2) - ((0.5*(vx(j,1)+vx(j,2)))^2))/dx; %
              convective acc
56        a4 = (0.25*(vx(j,1)+vx(j-1,1))*(vy(j-1,1)+vy(j-1,end)) -
              0.25*(vx(j+1,1)+vx(j,1))*(vy(j,1)+vy(j,end)))/dy;
57
58        A = (mu/rho)*(a1/dx + a2/dy) + a3 + a4;
59        u_star(j,1) = vx(j,1) + dt*(A + pgrad/rho + (1/dx/rho)*(p(j-1,end) -
              p(j-1,1))); % velocity + dt * (A - alpha - pressure gradient)
60        %end
61
62        a1 = 2*(vx(j,1) - 2*vx(j,end) + vx(j,end-1))/dx; % viscous forces
63        a2 = (vx(j+1,end) - 2*vx(j,end) + vx(j-1,end))/dy + (vy(j,end)- vy(j,end-1) -
              vy(j-1,end) + vy(j-1,end-1))/dx; % viscous forces
64        a3 = (((0.5*(vx(j,end-1)+vx(j,end)))^2) - ((0.5*(vx(j,end)+vx(j,1)))^2))/dx;
              % convective acc
65        a4 = (0.25*(vx(j,end)+vx(j-1,end))*(vy(j-1,end)+vy(j-1,end-1)) -
              0.25*(vx(j+1,end)+vx(j,end))*(vy(j,end)+vy(j,end-1)))/dy;
66
67        A = (mu/rho)*(a1/dx + a2/dy) + a3 + a4;
68        u_star(j,end) = vx(j,end) + dt*(A + pgrad/rho + (1/dx/rho)*(p(j-1,end-1) -
              p(j-1,end))); % velocity + dt * (A - alpha - pressure gradient)
69
70     end
71
72     % boundaries vx
73     u_star(1,:) = -u_star(2,:);       % vx in ghost points
74     u_star(end,:) = -u_star(end-1,:); % vx in ghost points
75
76
77     %particle
78  %left and right side
79    u_star(H_dist+1:H_dist+H_part, L_dist:L_dist+L_part) = 0;
80  %down and top side
81    u_star(H_dist+1 , L_dist +1 : L_dist+L_part-1) = -u_star(H_dist-1 , L_dist +1 :
          L_dist+L_part-1);
82    u_star(H_dist+H_part , L_dist +1 : L_dist+L_part-1) = -u_star(H_dist+H_part+1 ,
          L_dist +1 : L_dist+L_part-1);
83
84
85  % calculate v_star at every point in stag grid
86
87  for i = 2:M    % every row between two pressure points
88
89     for j = 2:N-1     % every column without inlet and outlet points
90        b1 = (vy(i,j+1) - 2*vy(i,j) + vy(i,j-1))/dx + (vx(i+1,j+1) - vx(i,j+1) -
              vx(i+1,j) + vx(i,j))/dy; % viscosity
91        b2 = 2*(vy(i+1,j) - 2*vy(i,j) + vy(i-1,j))/dy; % viscosity
```

```
92        b3 = ((0.5*(vy(i,j)+vy(i-1,j)))^2 - (0.5*(vy(i+1,j)+vy(i,j)))^2)/dy; %
              convective
93        b4 = (0.25*(vy(i,j)+vy(i,j-1))*(vx(i+1,j)+vx(i,j)) -
              0.25*(vy(i,j+1)+vy(i,j))*(vx(i+1,j+1)+vx(i,j+1)))/dx;
94
95        B = (mu/rho)*(b1/dx + b2/dy) + b3 + b4;
96        v_star(i,j) = vy(i,j) + dt*(B + (1/dy/rho)*(p(i-1,j) - p(i,j))); % old
              velocity + dt*(B - pressure gradient)
97    end
98
99    % boundaries vy
100
101       % start
102       b1 = (vy(i,2) - 2*vy(i,1) + vy(i,end))/dx + (vx(i+1,2) - vx(i,2) - vx(i+1,1)
              + vx(i,1))/dy; % viscosity
103       b2 = 2*(vy(i+1,1) - 2*vy(i,1) + vy(i-1,1))/dy; % viscosity
104       b3 = ((0.5*(vy(i,1)+vy(i-1,1)))^2 - (0.5*(vy(i+1,1)+vy(i,1)))^2)/dy; %
              convective
105       b4 = (0.25*(vy(i,1)+vy(i,end))*(vx(i+1,1)+vx(i,1)) -
              0.25*(vy(i,2)+vy(i,1))*(vx(i+1,2)+vx(i,2)))/dx;
106
107       B = (mu/rho)*(b1/dx + b2/dy) + b3 + b4;
108       v_star(i,1) = vy(i,1) + dt*(B + (1/dy/rho)*(p(i-1,1) - p(i,1))); % old
              velocity + dt*(B - pressure gradient)
109
110       % end
111       b1 = (vy(i,1) - 2*vy(i,end) + vy(i,end-1))/dx + (vx(i+1,1) - vx(i,1) -
              vx(i+1,end) + vx(i,end))/dy; % viscosity
112       b2 = 2*(vy(i+1,end) - 2*vy(i,end) + vy(i-1,end))/dy; % viscosity
113       b3 = ((0.5*(vy(i,end)+vy(i-1,end)))^2 - (0.5*(vy(i+1,end)+vy(i,end)))^2)/dy;
              % convective
114       b4 = (0.25*(vy(i,end)+vy(i,end-1))*(vx(i+1,end)+vx(i,end)) -
              0.25*(vy(i,1)+vy(i,end))*(vx(i+1,1)+vx(i,1)))/dx;
115
116       B = (mu/rho)*(b1/dx + b2/dy) + b3 + b4;
117       v_star(i,end) = vy(i,end) + dt*(B + (1/dy/rho)*(p(i-1,end) - p(i,end))); %
              old velocity + dt*(B - pressure gradient)
118
119   end
120
121   %boundaries vy
122
123   v_star(1,:) = 0; % vy at wall
124   v_star(end,:) = 0; % vy at wall
125
126   %particle
127   %down and top side
128    v_star(H_dist:H_dist+H_part , L_dist : L_dist+L_part-1) = 0;
129
130   %left and right side
131    v_star(H_dist+1:H_dist+H_part-1, L_dist) = -v_star(H_dist+1:H_dist+H_part-1,
           L_dist-1);
132    v_star(H_dist+1:H_dist+H_part-1, L_dist+L_part-1) =
           -v_star(H_dist+1:H_dist+H_part-1 , L_dist+L_part);
133
134
135   % continuity equation
136
137   % solve poisson equation for pressure correction
138   a = 2*dt*((dy/dx) + (dx/dy));
139   b = -dt*(dy/dx);
```

```
c = -dt*(dx/dy);

A = zeros(M*N,N*M);
D = zeros(N,M);

%matrix for coefficients poisson equation

for k = 1:M

   A(1+(k-1)*N,1+(k-1)*N) = a;
   A(1+(k-1)*N,2+(k-1)*N) = b;
   A(1+(k-1)*N,N+(k-1)*N) = b;

   for i = 2:N-1
   A(i+(k-1)*N,i+(k-1)*N) = a;
   A(i+(k-1)*N,i-1+(k-1)*N) = b;
   A(i+(k-1)*N,i+1+(k-1)*N) = b;
   end

   A(N+(k-1)*N,N+(k-1)*N) = a;
   A(N+(k-1)*N,N-1+(k-1)*N) = b;
   A(N+(k-1)*N,1+(k-1)*N) = b;

   if k == 1
      for i = 1+(k-1)*N : N+(k-1)*N
         A(i, i+N) = c;
      end

      A(1+(k-1)*N,1+(k-1)*N) = -2*b - c;
      for j = 2:N-1
         A(j+(k-1)*N,j+(k-1)*N) = -2*b - c;
      end
      A(N+(k-1)*N,N+(k-1)*N) = -2*b - c;

   elseif k == M
      for i = 1+(k-1)*N : N+(k-1)*N
         A(i, i-N) = c;
      end

      A(1+(k-1)*N,1+(k-1)*N) = -2*b - c;
      for j = 2:N-1
         A(j+(k-1)*N,j+(k-1)*N) = -2*b - c;
      end
      A(N+(k-1)*N,N+(k-1)*N) = -2*b - c;


   else
      for i = 1+(k-1)*N : N+(k-1)*N
         A(i, i-N) = c;
         A(i, i+N) = c;
      end
   end

%set p_prime(1,1) to certain value
  A(1,:) = 0;
  A(1,1) = 1;
end

% array with right hand sides of continuity equation
for j = 1:M
for i = 1:N
```

```
    if i == N
       D(i,j) = rho*(dy*(u_star(j+1,i) - u_star(j+1,1)) + dx*(v_star(j,i) -
           v_star(j+1,i)));
    else
       D(i,j) = rho*(dy*(u_star(j+1,i) - u_star(j+1,i+1)) + dx*(v_star(j,i) -
           v_star(j+1,i)));
    end
end
end

%set p_prime to value:
D(1,1) = 0;

d = reshape(D,[],1);

% particle
% determine position in matrix A of particle
L_p = zeros(1,L_part);
for i = L_dist:(L_dist+L_part-1)
    L_p(i-L_dist+1) = i;
end

H_p = zeros(H_part,1);
for i = H_dist:(H_dist+H_part-1)
    H_p(i-H_dist+1) = i;
end

N_p1 = zeros(H_part , L_part);
for i = 1:H_part
for j = 1:L_part
N_p1(i,j) = N*(H_p(i)-1) + L_p(j);
end
end

% change a of pressure correction next to block in matrix A

%left and right side
for i = 1:H_part
    A(N_p1(i,1)-1 , N_p1(i,1)-1) = -b - 2*c;
    A(N_p1(i,L_part)+1 , N_p1(i,L_part)+1) = -b - 2*c;
end

%bottom and top side
for i = 1:L_part
    A(N*(H_p(1)-2) + L_p(i) , N*(H_dist-2) + L_p(i)) = -2*b - c;
    A(N*H_p(end) + L_p(i) , N*H_p(end) + L_p(i)) = -2*b - c;
end

N_p1 = reshape(N_p1, [], 1);
N_p1 = sort(N_p1);

% sort in descending order and delete all rows and columns in A and d
N_p2 = sort(N_p1,'descend');
for i = 1:length(N_p2)
    A(N_p2(i), :) = [];
    A(:, N_p2(i)) = [];
    d(N_p2(i)) = [];
end

% solve pressure correction

```

```
260 p1 = A\d;
261
262 % fill array back with zeros at particle place for total flow
263 zer = zeros(L_part,1);
264
265 for i = 1:H_part
266 p1 = [p1(1:N_p1(1+L_part*(i-1))-1) ; zer ; p1(N_p1(1+L_part*(i-1)):end)];
267 end
268
269 % from vector make pressure correction field
270
271 p2 = reshape(p1, [N,M]);
272 p_prime = p2';
273
274
275 %compute u_prime at every point between two p_primes
276
277    for i = 1:M
278       for j = 2:N
279       u_prime(i,j) = (dt/dx/rho)*(p_prime(i,j-1) - p_prime(i,j));
280       end
281    u_prime(i,1) = (dt/dx/rho)*(p_prime(i,end) - p_prime(i,1));
282    end
283
284 %compute v_prime at every point between two p_primes
285
286    for j = 1:N
287    for i = 2:M
288       v_prime(i-1,j) = (dt/dy/rho)*(p_prime(i-1,j) - p_prime(i,j));
289    end
290    end
291
292 % update u and v star
293    for j = 1:N
294
295 % update u
296    for i = 1:M
297       vx(i+1,j) = u_star(i+1,j) + alpha*u_prime(i,j);
298    end
299 % update u in ghost points
300    vx(1,j) = -vx(2,j);
301    vx(end,j) = -vx(end-1,j);
302
303 % update v
304    for i = 1:M-1
305       vy(i+1,j) = v_star(i+1,j) + alpha*v_prime(i,j);
306    end
307    end
308
309 % update p
310    p = p + (1-alpha)*p_prime;
311
312
313 if niter > startconvergencecheck
314 % convergence
315
316    convergu(niter-startconvergencecheck) = (vtoconvergence(niter) -
          vtoconvergence(niter-startconvergencecheck))/vtoconvergence(niter);
317
318    err = convergu(niter-startconvergencecheck)
319 end
```

```
320
321
322     niter = niter + 1
323
324 vtoconvergence(niter) = max(vx(round(M/2),:));
325
326 %particle
327 %left and right side
328   vx(H_dist+1:H_dist+H_part, L_dist:L_dist+L_part) = 0;
329 %down and top side
330   vx(H_dist+1 , L_dist +1 : L_dist+L_part-1) = -vx(H_dist , L_dist +1 :
          L_dist+L_part-1);
331   vx(H_dist+H_part , L_dist +1 : L_dist+L_part-1) = -vx(H_dist+H_part+1 , L_dist +1
          : L_dist+L_part-1);
332
333 %particle
334 %down and top side
335   vy(H_dist:H_dist+H_part , L_dist : L_dist+L_part-1) = 0;
336 %left and right side
337   vy(H_dist+1:H_dist+H_part-1, L_dist) = -vy(H_dist+1:H_dist+H_part-1, L_dist-1);
338   vy(H_dist+1:H_dist+H_part-1, L_dist+L_part-1) = -vy(H_dist+1:H_dist+H_part-1 ,
          L_dist+L_part);
339
340 end
341
342
343
344
345 %% visualisation
346
347 u_vec = zeros(M,N);
348 for i = 1:M
349 for j = 1:N-1
350 u_vec(i,j) = 0.5*(vx(i+1,j)+vx(i+1,j+1));
351 end
352 u_vec(i,N) = 0.5*(vx(i+1,N)+vx(i+1,1));
353 end
354
355 v_vec = zeros(M,N);
356 for j = 1:N
357 for i = 1:M
358    v_vec(i,j) = 0.5*(vy(i,j)+vy(i+1,j));
359 end
360 end
361
362 %add starting column
363 u_vec = [vx(2:end-1,1) u_vec vx(2:end-1,end)];
364 v_vec1 = zeros(M,1);
365 for i = 1:M
366    v_vec1(i) = 0.5*(vy(i,1)+vy(i,end));
367
368 end
369 v_vec = [v_vec1 v_vec v_vec1];
370
371 %add bottom and top row
372 v_vec = [(0.5*(vy(1,1)+vy(1,end))) vy(1,1:end) (0.5*(vy(1,1)+vy(1,end))); v_vec ;
       (0.5*(vy(end,1)+vy(end,end))) vy(end,1:end) (0.5*(vy(end,1)+vy(end,end)))];
373 u_vec1 = zeros(1,N);
374 u_vec2 = zeros(1,N);
375 for i = 1:N
376    u_vec1(i) = 0.5*(vx(1,i) + vx(2,i));
```

```
377     u_vec2(i) = 0.5*(vx(M+1,i) + vx(M+2,i));
378 end
379 u_vec = [u_vec1(1) u_vec1 u_vec1(end) ; u_vec ; u_vec2(1) u_vec2 u_vec2(end)];
380
381
382 % try makeing new grid for streamline plot
383 X = zeros(1,N+2);
384 for i = 1:N
385     X(i+1) = 0.5*(x(i) + x(i+1));
386 end
387 X(end) = L;
388
389 Y = zeros(1,M+2);
390 for i = 1:M
391     Y(i+1) = 0.5*(y(i)+ y(i+1));
392 end
393 Y(end) = H;
394
395 [X,Y] = meshgrid(X,Y);
396
397
398 starty = linspace(0,H,101);
399 startx = zeros(size(starty));
400
401
402 streamline(X,Y,u_vec,v_vec,startx,starty)
403 hold on
404 quiver(X,Y,u_vec,v_vec)
405
406 rectangle('position',[(dx*(L_dist-1)) (dy*(H_dist-1)) (dx*L_part)
        (dy*H_part)],'facecolor','blue','Curvature',0.4)
407
408
409
410 %% y forces fluid on block
411
412 % forces xy east and west
413
414 tauxye = zeros(H_part,1);
415 tauxyw = zeros(H_part,1);
416
417 for i = 1:H_part
418     tauxye(i) = mu*0.5*(vy(H_dist+i , L_dist+L_part) + vy(H_dist-1+i ,
            L_dist+L_part) - vy(H_dist+i , L_dist+L_part-1) - vy(H_dist-1+i ,
            L_dist+L_part-1))/dx;
419     tauxyw(i) = -mu*0.5*(vy(H_dist+i , L_dist) + vy(H_dist-1+i , L_dist) -
            vy(H_dist+i , L_dist-1) - vy(H_dist-1+i , L_dist-1))/dx; %minus for flow on
            block
420 end
421
422 Ffobe = sum(tauxye*dy);
423 Ffobw = sum(tauxyw*dy);
424
425
426 % forces yy top and bottom
427
428 tauyyt = zeros(L_part,1);
429 tauyyb = zeros(L_part,1);
430
431 for i = 1:L_part
432     tauyyt(i) = 2*mu*vy(H_part+H_dist+1,L_dist-1+i)/dy;
```

```
433     tauyyb(i) = 2*mu*vy(H_dist-1,L_dist-1+i)/dy; %minus for flow on block
434 end
435 
436 Ffobt = sum(tauyyt*dx);
437 Ffobb = sum(tauyyb*dx);
438 
439 
440 % pressure on block
441 
442 pfobt = zeros(L_part,1);
443 pfobb = zeros(L_part,1);
444 
445 for i = 1:L_part
446     pfobb(i) = (-pgrad*dx*(L_dist-1.5+i) + p(H_dist-1,L_dist-1+i))*dx;
447     pfobt(i) = -(-pgrad*dx*(L_dist-1.5+i) + p(H_dist+H_part,L_dist-1+i))*dx;
448 end
449 
450 
451 Pfobt = sum(pfobt);
452 Pfobb = sum(pfobb);
453 
454 Fl = Ffobe + Ffobw + Ffobt + Ffobb + Pfobt + Pfobb;
455 
456 
457 %% x forces fluid on block
458 
459 % top and bottom side
460 
461 tauyxfobb = zeros(L_part,1);
462 tauyxfobt = zeros(L_part,1);
463 
464 for i = 1:L_part
465     tauyxfobb(i) = 0.5*mu*(vx(H_dist,L_dist-1+i) + vx(H_dist,L_dist+i) -
            vx(H_dist+1,L_dist-1+i) - vx(H_dist+1,L_dist+i))/dy; %min hierin verwerkt
466     tauyxfobt(i) = 0.5*mu*(vx(H_dist+H_part+1,L_dist-1+i) +
            vx(H_dist+H_part+1,L_dist+i) - vx(H_dist+H_part,L_dist-1+i) -
            vx(H_dist+H_part,L_dist+i))/dy;
467 end
468 
469 Fyxfobb = sum(tauyxfobb*dx);
470 Fyxfobt = sum(tauyxfobt*dx);
471 
472 
473 % east and west side
474 
475 tauxxfobw = zeros(H_part,1);
476 tauxxfobe = zeros(H_part,1);
477 
478 for i = 1:H_part
479     tauxxfobw(i) = 2*mu*vx(H_dist+i,L_dist-1)/dx; %min times min for block on flow
480     tauxxfobe(i) = 2*mu*vx(H_dist+i,L_dist+L_part+1)/dx;
481 end
482 
483 Fxxfobw = sum(tauxxfobw*dy);
484 Fxxfobe = sum(tauxxfobe*dy);
485 
486 
487 % pressure
488 
489 pfobw = zeros(H_part,1);
490 pfobe = zeros(H_part,1);
```

```
491
492 for i = 1:H_part
493    pfobw(i) = dy*(-pgrad*dx*(L_dist-1) + p(H_dist-1+i , L_dist-1));
494    pfobe(i) = -dy*(-pgrad*dx*(L_dist+L_part-1) + p(H_dist-1+i , L_dist+L_part));
495 end
496
497 Pfobw = sum(pfobw);
498 Pfobe = sum(pfobe);
499
500 Fdrag = Fyxfobb + Fyxfobt + Fxxfobw + Fxxfobe + Pfobw + Pfobe;
501
502
503 %% torque
504 % clockwise is positive, anticlockwise is negative
505
506
507 % distance to middle
508
509 rytb = linspace((L_part-1)/2,-(L_part-1)/2,L_part);
510 rxew = linspace(-(H_part-1)/2,(H_part-1)/2,H_part);
511 ryw = abs(rxew);
512 rye = -abs(rxew);
513 rxt = abs(rytb);
514 rxb = -abs(rytb);
515
516 % forces acting on block
517
518 torytb = zeros(L_part,1);
519 toryew = zeros(H_part,1);
520 torxtb = zeros(L_part,1);
521 torxew = zeros(H_part,1);
522
523 for i = 1:L_part
524    torytb(i) = dx*(tauyyb(i) + tauyyt(i) + pfobb(i) + pfobt(i))*rytb(i);
525    torxtb(i) = dx*(tauyxfobb(i)*rxb(i) + tauyxfobt(i)*rxt(i));
526 end
527
528 for i = 1:H_part
529    toryew(i) = dy*(tauxye(i)*rye(i) + tauxyw(i)*ryw(i));
530    torxew(i) = dy*rxew(i)*(tauxxfobe(i) + tauxxfobw(i) + pfobe(i) + pfobw(i));
531 end
532
533
534 torque = sum(torytb) + sum(toryew) + sum(torxtb) + sum(torxew);
535
536
537
538 % try making vector arrow
539
540
541 % start arrow
542 starrx = dx*(L_dist-1 + L_part/2);
543 starry = dy*(H_dist-1 + H_part/2);
544
545 % slope arrow
546 slox = Fdrag*10^3;
547 sloy = Fl*10^4;
548
549
550
551 %quiver(starrx , starry , slox , sloy)
```

```
552 quiver(starrx , starry , 0 , sloy)
553 quiver(starrx , starry , slox , 0)
554 xlim([0 L])
555 ylim([0 H])
556
557 %% non dimensionalise forces
558
559 vcl = max(vx(round(M/2),:));
560
561 CD = 2*Fdrag/(rho*(vcl^2)*H_part*dy);
562 CL = 2*Fl/(rho*(vcl^2)*L_part*dx);
563 CT = 2*torque/(rho*(vcl^2)*H_part*dy*L_part*dx);
564
565 % Reynolds number
566
567 reynolds = (rho/mu)*vcl*H_part*dy;
568
569
570 %% save
571
572 save(['bephtoH14_for_Re_is_' num2str(reynolds) '.mat'])
573
574 end
```