

Anita Klavenes

Steady Laminar Flow over a Backwards Facing Step solved by the Finite Volume Method

Master's thesis in Chemical Engineering and Biotechnology

Supervisor: Hugo Atle Jakobsen

July 2020

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering



Norwegian University of
Science and Technology

Anita Klavenes

Steady Laminar Flow over a Backwards Facing Step solved by the Finite Volume Method

Master's thesis in Chemical Engineering and Biotechnology
Supervisor: Hugo Atle Jakobsen
July 2020

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering



Preface

This master thesis was written in the spring of 2020, and marks the end of the five year long integrated masters program in Chemical Engineering and Biotechnology, with specialisation in Environmental Engineering and Reactor Technology. The thesis work is a continuation of the specialisation project from the autumn of 2019.

Thank you to my supervisor professor Hugo Atle Jakobsen for the opportunity to work on this topic for my specialisation project and master's thesis and for always keeping the (virtual) door to your office open whenever I had questions. Thank you also to my co-supervisors Suat Canberk Ozan and professor Jannike Solsvik for the much appreciated guidance and support.

I would like to sincerely thank my parents and brother for all the encouragement over many years, and Sander for all the love and constant support.

Declaration of Compliance

I declare that this is an independent work according to the exam regulations of the Norwegian University of Science and Technology (NTNU).

Trondheim, 3rd July 2020

Summary

Laminar, steady flow with no heat transfer in a straight channel and over a backwards facing step has been solved by the Finite Volume Method. The SIMPLE-algorithm and the Upwind Differencing Scheme were used and the discretised governing equations formulated in Cartesian coordinates were solved in **MATLAB**. The pressure and velocities have been solved simultaneously. The backwards facing step domains had two different expansion ratios of $H/h = 1.5$ and 2, and both a constant inlet velocity and a parabolic inlet velocity profile were used. A known pressure was used for the outlet boundary condition.

The thesis is a continuation from the specialisation project of the fall of 2019, and the models created in this project were improved. The governing equations were solved on their dimensionless form, and the results for the backwards facing step domains were obtained for a range of low Reynolds numbers between 0.0001 and 400. The reattachment lengths of the recirculation zones were found to be in agreement with results found in literature, but the resolution of the grid was not high enough to show the recirculation at the lowest Reynolds numbers. The flow into the expanded section did not resemble the results found in literature, which likely was due to the choice of discretisation scheme, since using the Upwind Differencing Scheme for the convective terms can lead to some errors related to false diffusion.

A transfinite interpolation technique was used to obtain an algebraic grid for use when solving the fluid flow problem formulated in generalised curvilinear coordinates. A code for an elliptic grid using the algebraic grid as an initial guess was made, but the code did not yield the satisfactory grid, most likely due to a mistake in the discretised elliptic grid generation equations or in the code.

Sammendrag

Laminær, stasjonær strømning uten varmetransport i en rett kanal og i en kanal utvidet over et trinn (backwards facing step) har blitt løst ved bruk av Finite Volume Method. SIMPLE-algoritmen og Upwind Differencing ble brukt, og de diskretiserte strømningslikningene formulert i kartesiske koordinater ble løst i MATLAB. Trykk og hastighet ble beregnet samtidig. Trinnet i den utvidede kanalen hadde to høyder på $H/h = 1.5$ og 2 relativt til høyden på innløpet. På innløpet ble en konstant hastighet og en parabolisk hastighetsprofil brukt, mens på utløpet ble et kjent trykk brukt som grensebetingelse.

Denne oppgaven er en videreføring av arbeid gjort i forbindelse med fordypningsprosjektet høsten 2019, og modellene som ble utviklet i fordypningsprosjektet har blitt forbedret i denne oppgaven. Strømningslikningene har blitt løst på sin dimensjonsløse form, og for den utvidede kanalen ble strømmingen modellert for ulike lave Reynoldstall mellom 0.0001 og 400 . Lengen på resirkulasjonssonene etter steget stemmer overens med resultater fra litteraturen, men grunnet det relativt lave antallet celler brukt i beregningene er ikke resirkulasjonen synlig for de laveste Reynoldstallene. Strømningsmønsteret over steget skiller seg fra litteraturen, noe som kan forklares med valget av teknikk for diskretisering av konveksjonsleddene, siden Upwind Differencing kan gi unøyaktigheter som likner diffusjon.

Transfinite Interpolation ble brukt til å generere et algebraisk nett som kan brukes til beregning av strømningslikningene formulert med generelle kurvilineære koordinater. Det ble også laget en kode som genererer et elliptisk nett med det algebraiske nettet som initialbetingelse, men denne koden ga ikke et tilfredsstillende resultat. Mest sannsynlig er dette relatert til en feil i diskretiseringen av de elliptiske likningene, eller en feil i koden.

Table of Contents

Preface	i
Summary	iii
Sammendrag	v
List of Symbols	xi
1 Introduction	1
1.1 Previous Project Work	3
1.2 Objective of the Thesis	3
1.3 Assumptions	4
1.4 Survey of the Thesis	4
2 Theoretical Background	5
2.1 Fluid Flow	5
2.2 The Finite Volume Method	8
2.3 Properties of Numerical Schemes	16
2.4 Discretisation of the Domain	17
2.5 Non-Dimensional Equations	19
2.6 Solving Systems of Linear Algebraic Equations in MATLAB	20
3 Discretisation	21
3.1 Continuity Equation	21
3.2 Momentum Equation	22
3.3 SIMPLE-Equations	28
3.4 Dimensionless Equations	30
4 Implementation	39
4.1 Properties of the Flow and the Domain	39
4.2 Model Settings	40
4.3 Initial Guesses	42
4.4 Boundary Conditions	43
4.5 Backwards Facing Step	51
4.6 Dimensionless Equations For Comparison	56
4.7 Convergence Criteria	58
4.8 Plotting	60
4.9 Composition and Working Principle of the Code	61
5 Results	65
5.1 Two Dimensional Straight Channel	65

5.2	Backwards Facing Step Model	76
6	Discussion	91
6.1	Straight Channel Model	91
6.2	Backwards Facing Step Model	91
6.3	Model Improvements from Specialisation Project	94
7	Grid Generation	97
7.1	Theory	97
7.2	Governing Equations in General Coordinates	100
7.3	Discretisation of the Grid Generation Equations	101
7.4	Implementation	114
7.5	Results and Discussion	118
8	Conclusion	121
8.1	Recommendations for Future Work	121
	Bibliography	123
	Appendix	127

List of Symbols

Symbols

Symbol	Unit	Description
A	m^2	Surface area of control volume
\mathbf{A}	m^2	Face area vector
a	kg/s	Coefficient in velocity equation
β	Pa	Vector of source terms for pressure correction
b	m/s	Vector of source terms for velocities
c	-	Coefficient in elliptic grid equation
χ	-	Arbitrary variable
D	$\text{Pa}\cdot\text{s/m}$	Diffusion conductance
δ	-	Kronecker delta
δx	m	Width of control volume in x -direction
δy	m	Width of control volume in y -direction
δz	m	Width of control volume in z -direction
\mathbf{e}_x	-	Unit vector in x -direction
\mathbf{e}_y	-	Unit vector in y -direction
\mathbf{e}_z	-	Unit vector in z -direction
ε	-	Permutation symbol
F	kg/sm^2	Convective mass flux per unit area
F_s	$\text{Pa}\cdot\text{m}^2$	Shear force
ϕ	-	Arbitrary node or property
ϕ	-	Lagrange interpolation polynomial
g	m/s^2	Gravitational acceleration
g		Contravariant tensor component
\mathbf{g}		General base vector
Γ	var.	Diffusion coefficient
H	m	Channel height
h	m	Channel height
J	-	Jacobi determinant
L	m	Channel length
l	m	Channel length
M	-	Number of scalar nodes in y -direction
m	-	Number of v -velocity nodes in y -direction
μ	$\text{Pa}\cdot\text{s}$	Viscosity
N	-	Number of scalar nodes in x -direction
\mathbf{n}	-	Direction vector normal to surface
ν	$\text{s}\cdot\text{m}$	Coefficient in pressure equation
ρ	kg/m^3	Density

Continued on next page

Continued from previous page

Symbol	Unit	Description
p	Pa	Pressure
P	-	Poisson control function
Pe	-	Péclet number
ψ	-	Lagrange interpolation polynomial
q	-	Curvilinear coordinate
\mathbf{qr}	-	Position vector
T	-	Matrix of coefficients for pressure
τ	Pa	Shear stress
U	-	Matrix of coefficients for x -velocity
u	m/s	Velocity in x -direction
V	-	Matrix of coefficients for y -velocity
V	m ³	Volume
v	m/s	Velocity in y -direction
x	-	x -direction coordinate
y	-	y -direction coordinate
z	-	z -direction coordinate

Diacritics

Diacritic	Description
~	Adjusted variable
^	Dimensionless variable

Superscripts

Superscript	Description
*	Intermediate obtained after matrix inversion
◦	Initial guess
'	Correction value
c	Continuity coefficient
i	Coordinate index
j	Coordinate index
p	Coordinate index
q	Coordinate index

Subscripts

Subscript	Description
E	Eastern node
e	Eastern control volume face
I	Scalar (pressure) node index in x -direction
i	Velocity node index in x -direction
J	Scalar (pressure) node index y -direction

Continued on next page

Continued from previous page

Subscript	Description
j	Velocity node index in y -direction
k	Coordinate index
l	Coordinate index
M	Maximum index of scalar nodes in y -direction
m	Coordinate index
m	Maximum index of v -velocity nodes in y -direction
N	Northern node
N	Maximum index of scalar nodes in x -direction
n	Northern control volume face
nb	Neighbouring coefficient
P	Current node
S	Southern node
s	Southern control volume face
W	Western node
w	Western control volume face
x	x -direction
y	y -direction
z	z -direction

Abbreviations

Abbreviation	Description
1D	One dimension
2D	Two dimensions
BFS	Backwards Facing Step
CFD	Computational Fluid Dynamics
CV	Control volume
FVM	Finite Volume Method
LHS	Left hand side
PDE	Partial Differential Equation
RHS	Right hand side
TFI	Transfinite interpolation

1

Introduction

In this thesis, laminar, steady flow with no heat transfer will be solved by the Finite Volume Method. The Continuity equation and the Momentum equation for fluid motion will be the starting point for calculating the pressure and the velocities in x - and y -direction. The pressure will be calculated using a semi-implicit equation derived from the Continuity equation, and this equation and the Momentum equation will be solved simultaneously.

The Finite Volume method is a numerical method for solving partial differential equations by expressing them as algebraic equations [1]. The appropriate equations for the problem of interest are integrated over a control volume drawn around each computational node in the domain [2]. Finite differences are used to approximate the derivative terms yielding a system of algebraic equations before the discretised equations are iterated until convergence. For the system in this thesis, the algebraic equations are linear and can be solved by matrix operations in `MATLAB`.

The fluid property ϕ is conserved across each control volume of the domain when using the Finite Volume method, which is a clear advantage. Conservation of ϕ can be achieved across the entirety of the domain by using consistent flux relations in the discretisation of the governing equations. The Finite Volume method is a variant of a Finite Difference method and is a common numerical method to use in Computational Fluid Dynamics (CFD) software, where mass and heat transfer problems are solved using computer simulations [2].

The flow domains will be various simple and complex geometries. Figure 1.1 shows a straight channel with two different lengths, which will be the domains in use for developing a two dimensional fluid flow model. The left channel is a short channel with the length corresponding to the length of the short channel before the backwards facing step in figure 1.2. The right channel is an extended channel corresponding to the full length of the backwards facing step domain. Figures 1.2 and 1.3 show two channel domains with an expansion of the channel, a backwards facing step. The first domain in figure 1.2 is used by Melaaen [3] and the second domain is used by Biswas et al. [4].

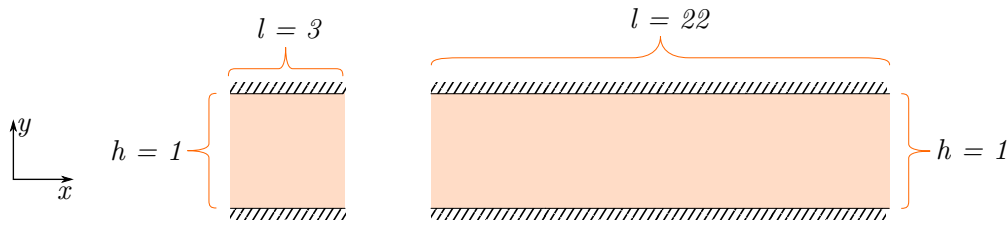


Figure 1.1: Straight channel domains.

Flow over a backwards facing step is an interesting topic in fluid mechanics [4][5], often because it is fairly simple and it has one fixed separation point where separation of the flow into layers can be observed [6].

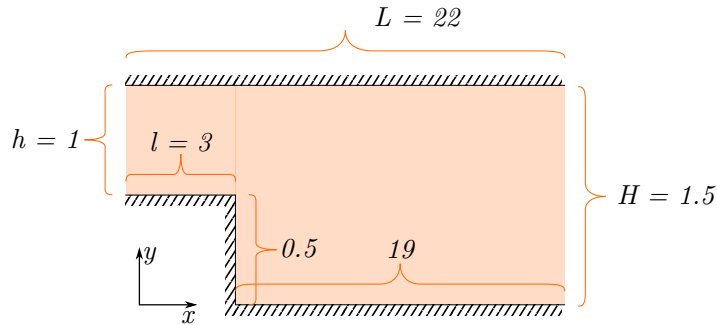


Figure 1.2: Domain as used by Melaaen [3], used to develop the two dimensional model for fluid flow over a backwards facing step.

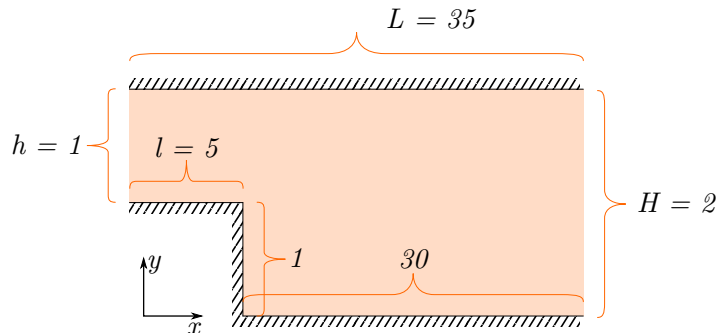


Figure 1.3: Domain as used by Biswas et al. [4], used in the backwards facing step model with a variation of Reynolds numbers for comparison to the results given by Biswas et al. [4].

A separation of the flow is expected around the step with a circulation zone under the step before the flow is reattached. Armaly et al. [7] also observed a secondary circulation zone after the first one on the northernmost wall for Reynolds numbers higher than around 400. This separation when the fluid flows over a sharp change of geometry is important within many fields of engineering, and has been a topic of study since the seventies, for example by Goldstein et al. [8] and Denham and Patrick [9] [5]. Flow separation of this sort can for example resemble the one over airfoils at large angles of attack, flow in turbines, heat-exchangers and compressors and flow in pipes with a rapid expansion [5][6][10]. The backwards facing step is also much used as a quite simple but also complex enough geometry for modelling of turbulent flow [5]. It is also a well established test geometry in CFD.

Several studies have been conducted on flow over the backwards facing step where velocity is calculated along with the reattachment length of the flow after the separation for large varieties of Reynolds numbers. Examples are Biswas et al. [4], Armaly et al. [7], Barton [11], Lee and Mateescu [12], and Nie and Armaly [13].

Building a model for the flow over the backwards facing step can work as a stepping stone for extending the model to new applications. Formulation of the model equations in generalised curvilinear coordinates around complex geometries is an interesting topic for which the backwards facing step is a good test geometry. With this method, a grid with different shape than a regular Cartesian coordinate grid is used, meaning that a dense number of computational points can be placed where accuracy is needed [3][14]. This would mean that the recirculation zone after the backwards facing step could be very well represented, while fewer nodes may be placed in the rest of the domain close to the edges, where the results are more trivial and not of great interest.

In this thesis, all the channels are rectangular like the channel seen in figure 1.4. A simplification was made by assuming that the channel is laying like in figure 1.4, and gravity is acting in z -direction.

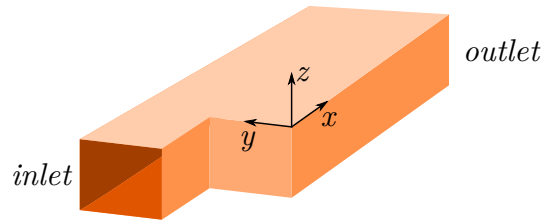


Figure 1.4: Example backwards facing step channel in three dimensions.

1.1 Previous Project Work

This thesis is a continuation of work that was done in a specialisation project in the fall of 2019 [15]. In this specialisation project, the main concepts of the finite volume method were studied, and a model was made for a one-dimensional and two-dimensional system as well as a backwards facing step model. These models had severe issues, and worked only for specific settings and parameter values. The models would not work for any inlet velocity far away from 1 m/s and the viscosity had to be kept to 1 Pa·s. The backwards facing step model was modelled by splitting the domain in two sections exactly at the step, and using the two-dimensional model for a square channel to solve the two domains. The computational time for these models were very long, and the backwards facing step model took approximately 14 hours to solve with a relatively coarse grid size.

The discretised equations in the fall project had some mistakes and the algorithm used in the **MATLAB** models was wrongly implemented and therefore slow. The algorithm used the velocities from the previous iteration for calculating the pressure correction, which acted as an extra under-relaxation step. This made all the models converge very slowly, and increasing the under-relaxation factors was not possible.

1.2 Objective of the Thesis

The objective of this thesis is to model laminar fluid flow in channels of regular and complex geometries using the Finite Volume Method. Furthermore, the objective is to cover the basic theory of grid generation for use when solving the same complex geometries using curvilinear coordinates, and to obtain an algebraic and an elliptic grid.

1.3 Assumptions

The fluid flow equations will be solved in one dimension and two dimensions in **MATLAB**. The flow is laminar and at steady state and will be solved using Cartesian coordinates. The modelled fluid is water and the fluid properties will be taken to be constant with the values given in equation (4.1.1). Heat transfer will not be calculated, and gravity will not be taken into account, meaning the gravitational force is in z -direction.

1.4 Survey of the Thesis

Chapter 2 covers the theory behind the models. Chapter 3 provides all the discretisations of the fluid flow equations. Implementation of the models in **MATLAB** as well as initial guesses and composition of the **MATLAB** models are given in chapter 4. Chapter 5 contains the resulting profiles and plots for the different flow parameters, as well as the results for the Reynolds number comparison. The results are discussed in chapter 6, and a discussion of the changes done to the models from the specialisation project is also given. Chapter 7 contains theory, derivation, implementation and results for grid generation for use when modelling the same domain in curvilinear generalised coordinates. Conclusions and recommendations for future work are given in chapter 8.

2

Theoretical Background

This chapter describes the underlying theory behind building of the fluid flow models used in this thesis. The covered theory includes fluid flow, the Finite Volume method, discretisation of the domain, and the solution of the equations in `MATLAB`.

2.1 Fluid Flow

For modelling fluid flow, a set of governing equations that describe the behaviour of the flow is used. The central equations for modelling fluid flow are the Continuity equation, the Equation of Motion and the Heat equation. For the case of this project, convective fluid flow with no heat transfer, the Continuity equation and the Equation of Motion are sufficient to model the domain. All the derivations of the model equations are given in chapter 3.

Equation (2.1.1) is the Mass Based Equation of Continuity [16][17].

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \quad (2.1.1)$$

where ρ is the density and \mathbf{u} is the velocity vector. Since the density is constant, the flow is incompressible, and the Continuity equation reduces to equation (2.1.2). In the derivation to yield the model equations in chapter 3, this simplification is used.

$$\nabla \cdot \mathbf{u} = 0 \quad (2.1.2)$$

The Equation of Motion in vector form is given in equation (2.1.3) [16][17]. It is also known as the Momentum Equation.

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \mathbf{u}) = -\nabla p - \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} \quad (2.1.3)$$

where ρ is the fluid density, \mathbf{u} is a vector of velocities, p is the pressure, $\boldsymbol{\sigma}$ is the shear stress and \mathbf{g} is a vector of gravity constants.

The Momentum Equation can also be noted in component form for each spatial coordinate. These equations are shown in appendix A.2 along with the expressions for the shear stress σ .

2.1.1 Developed Flow Profile

For fully developed flow, the v -velocity and the u -velocity gradient $\frac{\partial u}{\partial x}$ are zero, meaning that the u -velocity is only dependent on the y -position [18]. The fully developed flow takes a parabolic shape, and this profile is known as the *Hagen-Poiseuille law* and is given in equation (2.1.4) [16]. u_{\max} is located at $y = 0$.

$$u(y) = u_{\max} \left(1 - \left(\frac{y}{h} \right)^2 \right) \quad (2.1.4)$$

where h is the height of the channel. u_{\max} is the maximum velocity and is given by equation (2.1.5).

$$u_{\max} = 2u_{avg} \quad (2.1.5)$$

where u_{avg} is the average velocity which appears as u in the expression for the Reynolds number in equation (2.1.12). Equation (2.1.6) shows equation (2.1.4) altered to place u_{\max} at $y = \frac{h}{2}$.

$$u(y) = u_{\max} \left(1 - \left(\frac{y - \frac{h}{2}}{\frac{h}{2}} \right)^2 \right) \quad (2.1.6)$$

Figure 2.1 shows the parabolic profile at the inlet of the narrow channel, represented with 10 computational nodes in y -direction.

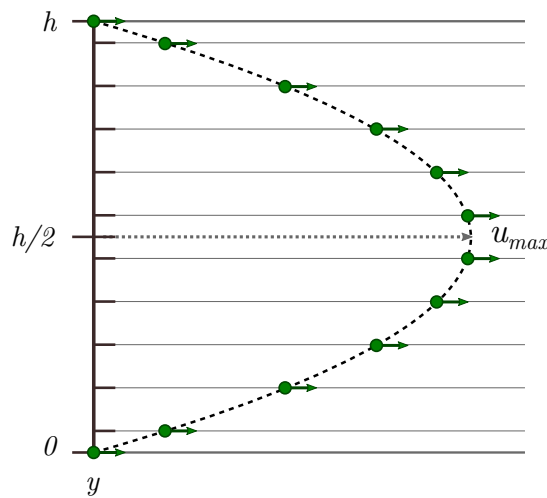


Figure 2.1: A parabolic velocity profile with u_{\max} located at $y = \frac{h}{2}$.

2.1.2 Wall Boundary

It is widely acknowledged that when approaching a wall, the fluid velocity goes to zero relative to the wall, as can be seen in figure 2.1 where there are walls at $y = 0$ and $y = h$. This is known as the no-slip condition and is caused by viscous effects close to the wall [19]. This condition requires that the tangential component of the velocity must be

zero at the surface. The no-penetration condition applies to the normal component of the velocity, which must be zero at the surface if the fluid can not move through the wall [20]. Hence, both the u - and the v -velocity are zero at the walls.

2.1.3 Reynolds Number

The Reynolds number is a dimensionless number that gives an indication of how large the viscous terms in the Momentum equation are compared to the rest of the terms [16][21]. The Reynolds number is defined by equation (2.1.7)[17].

$$Re = \frac{\rho u D}{\mu} \quad (2.1.7)$$

where ρ is the density of the fluid, u is the average velocity defined as the volumetric flow rate divided by cross-sectional area, D is the diameter of the tube and μ is the fluid viscosity. For non-circular tubes, there is no intuitive diameter, and the hydraulic diameter D_{hyd} is used instead [19]. Equation (2.1.7) becomes equation (2.1.8).

$$Re = \frac{\rho u D_{hyd}}{\mu} \quad (2.1.8)$$

where D_{hyd} is the hydraulic diameter. The hydraulic diameter for a rectangular duct is defined by equation (2.1.9) [19].

$$D_{hyd} = \frac{2hw}{h+w} \quad (2.1.9)$$

where h is the height of the channel in y -direction and w is the width of the channel in z -direction as can be seen in figure 2.2. For the two-dimensional system, w is the system depth and is equal to the unit length in z -direction which is 1. The hydraulic diameter is then defined by equation (2.1.10).

$$D_{hyd} = \frac{2h}{h+1} \quad (2.1.10)$$

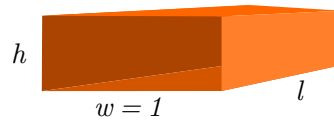


Figure 2.2: Rectangular duct with labels for the height h , width w and length l used in the calculation of the hydraulic diameter.

The magnitude of the Reynolds number categorises the flow into laminar, turbulent or a transition between the two. The range of each category varies somewhat within the literature. An example is given in equation (2.1.11) from Geankoplis [17].

$$\begin{aligned} Re < 2100 & \quad \text{Laminar} \\ 2100 \leq Re \leq 4000 & \quad \text{Transition range} \\ Re > 4000 & \quad \text{Turbulent} \end{aligned} \quad (2.1.11)$$

Bird et al. [21] defined the ranges as given in (2.1.12).

$$\begin{aligned} Re < 20 & \quad \text{Laminar flow with negligible rippling} \\ 20 < Re < 1500 & \quad \text{Laminar flow with pronounced rippling} \\ Re > 1500 & \quad \text{Turbulent} \end{aligned} \quad (2.1.12)$$

2.2 The Finite Volume Method

The Finite Volume method is a numerical method for solving partial differential equations by expressing them as algebraic equations [1]. When modelling fluid flow, the Finite Volume method is useful for discretisation of conservation laws.

2.2.1 Structure of the method

For modelling of the convective flow in this thesis, the method can be summarised in the following main steps:

1. Discretisation of the domain, specifying node points
2. Creation of three dimensional control volumes around each node
3. Discretisation of the appropriate governing equations describing the fluid flow
4. Integration of the equations over the control volumes
5. Approximation of derivative terms
6. Creation of the pressure linked equation (SIMPLE)
7. Iteration until convergence

The full discretisation of the transport equations from the form of the governing equations to the discretised form is described in chapter 3.

The integration over the control volumes is the most important step in the method [2]. In other numerical methods the flux terms in the governing equations are calculated at the node points along with the flow quantity in the flux term. By integration over the control volumes in the Finite Volume method, the flux terms appear on the cell faces instead.. This defines a *flux out* – *flux in* balance for each control volume. The integration over the control volumes therefore ensures conservation of the flow quantity ϕ across the control volume. By approximating the flux terms consistently everywhere, the conservation of ϕ is accomplished for the whole domain.

For other discretisation schemes, finite differences can be used to discretise the fluid property itself along with the flux terms as shown in figure 2.3. In the Finite Volume method, central differences are used to approximate the flux terms only as shown in figure 2.4 [1]. For the discretisation of the Momentum equation, this applies to the diffusive terms. The property itself appears in the convective terms in the Momentum equation and are instead discretised using the Upwind Differencing scheme as described in section 2.2.2.

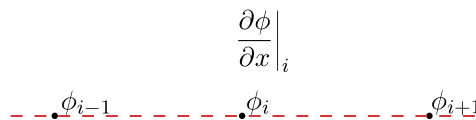


Figure 2.3: Discretisation method where the derivative $\left. \frac{\partial \phi}{\partial x} \right|_i$ is calculated in the same point as ϕ_i .

For the gradient of ϕ in the point i , the general central difference expression is shown in equation (2.2.1).

$$\left. \frac{\partial \phi}{\partial x} \right|_i = \frac{\phi_{i+1} - \phi_{i-1}}{2\delta x} \quad (2.2.1)$$

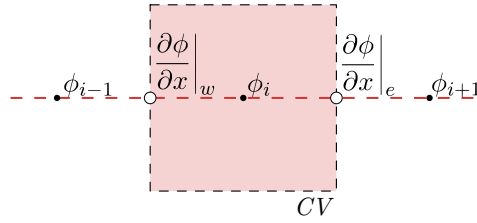


Figure 2.4: Discretisation in the Finite Volume method where the derivatives are calculated at the cell faces of the control volume CV around ϕ_i .

where $2\delta x$ notes the distance from ϕ_{i+1} to ϕ_{i-1} . Since the fluxes are given at the control volume faces, the gradients are defined in the middle between ϕ_{i-1} and ϕ_i and between ϕ_i and ϕ_{i+1} . The central differences needed for these flux terms surrounding node ϕ_i are given in equation (2.2.2).

$$\frac{\partial \phi}{\partial x} \Big|_w = \frac{\phi_i - \phi_{i-1}}{\delta x} \quad \frac{\partial \phi}{\partial x} \Big|_e = \frac{\phi_{i+1} - \phi_i}{\delta x} \quad (2.2.2)$$

Here δx notes the distance from ϕ_{i-1} to ϕ_i and from ϕ_i to ϕ_{i+1} , e signifies the eastern cell face and w signifies the western cell face of the control volume in figure 2.4. For a two or three dimensional case, the expressions for the northern, southern, top and bottom cell faces are also used.

2.2.2 The Upwind Differencing Scheme

After integration of the Momentum equation over the control volumes around the velocity nodes, the right hand side of the equation contains velocity gradients that can be approximated using central differences. After this, the right hand side terms contain the values at the velocity nodes themselves. On the left hand side the values of the velocities located on the cell faces appear instead. Equation (2.2.3) shows an example convection-diffusion equation after integration over the control volume [2]. F and D are defined in chapter 3.

$$F_e \phi_e - F_w \phi_w = D_e (\phi_E - \phi_P) - D_w (\phi_P - \phi_W) \quad (2.2.3)$$

The right hand side contains the terms ϕ_P , ϕ_E and ϕ_W located at the nodes, while the left hand side contains ϕ_e and ϕ_w defined at the cell faces of the control volume around node P . A discretisation scheme is needed for these cell face values.

The Upwind Differencing Scheme is a discretisation method that adapts to the direction of the flow. For flows that are highly convective, the convective terms in the Momentum Equation should be influenced the most by the value at the upwind node. When using a central differencing method, the neighbouring nodes are granted the same influence in the discretised equation since the direction of the flow is not taken into account.

Figure 2.5 from Versteeg and Malalasekera [2] shows a visualisation of the Upwind Differencing Scheme for eastgoing and westgoing flow (top and bottom respectively). The arrows indicate the flow direction. In positive (eastgoing in figure 2.5) convective flow, the western node w is located upwind from the centre node P , and should have a much larger influence in the Momentum Equation than the downstream node e . The cell face values ϕ_w and ϕ_e are then assigned as in equation (2.2.4).

$$\phi_w = \phi_W \quad \text{and} \quad \phi_e = \phi_P \quad (2.2.4)$$

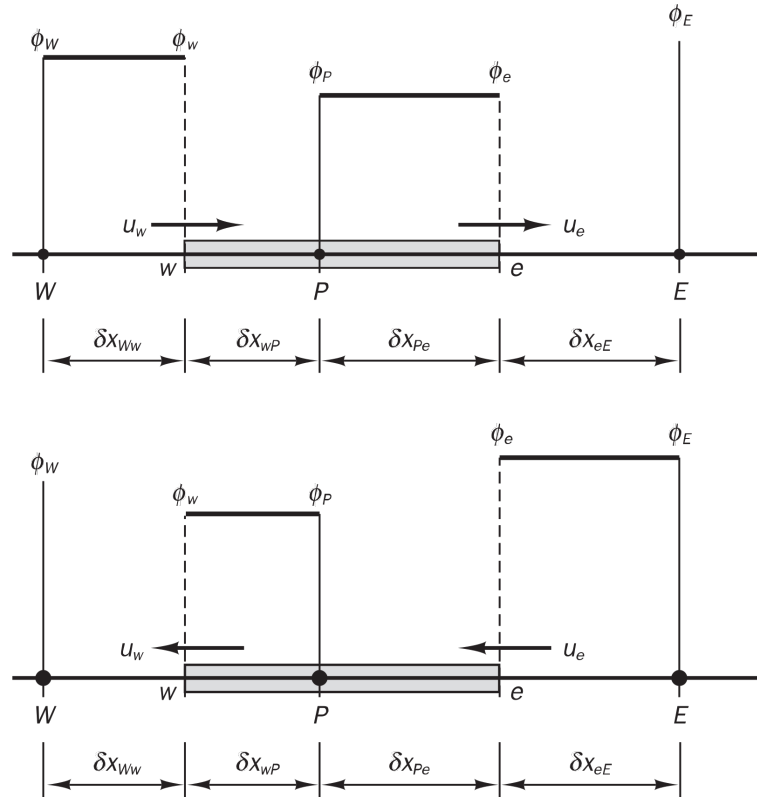


Figure 2.5: The Upwind Differencing Scheme visualised, the top figure shows the scheme for an eastgoing (positive) flow direction and the bottom figure shows the scheme for a westgoing (negative) flow direction. The figure is taken from Versteeg and Malalasekera [2].

For the negative flow (westgoing in figure 2.5) it is the eastern node that should have the greatest influence, as shown in equation (2.2.5).

$$\phi_w = \phi_P \quad \text{and} \quad \phi_e = \phi_E \quad (2.2.5)$$

It is also possible to use different discretisation schemes than the Upwind Differencing scheme, for example the Hybrid Discretisation Scheme or the QUICK Method [2].

2.2.3 Staggered Grid

Normally all the flow parameters and derivatives can be calculated at the same node points in the discretised domain. This means that a single node point would have a value for all the flow properties and derivatives. When using the Finite Volume Method, it is necessary to use a staggered grid instead. This means that the fluid properties are not all calculated in the same points in the domain. Instead, different grids are used for the different parameters. The scalars (pressure as well as density and viscosity if these are not constant) are calculated at one set of points, while the velocities are calculated at points located between these scalar node points. This yields three unique grids. The Continuity equation is placed at the scalar nodes in the domain, while the x - and y - components of the Momentum equation are placed on the u -velocity grid and the v -velocity grid, respectively.

The staggered grids are necessary because central differencing of the fluid flow equations cancel out the centre pressure node if the grids are not staggered. The result is that a non-uniform pressure field can appear uniform. Important information about the pressure field may not be well represented in the solution.

A visualisation of the staggered grid in two dimensions can be seen in figure 2.6. N is the number of scalar and v -velocity nodes in the domain in the x -direction and M is the number of scalar and u -velocity nodes in the y -direction. n is equal to N and is the number of u -velocity nodes in the x -direction and m is equal to $M - 1$ and is the number of v -velocity nodes in the y -direction.

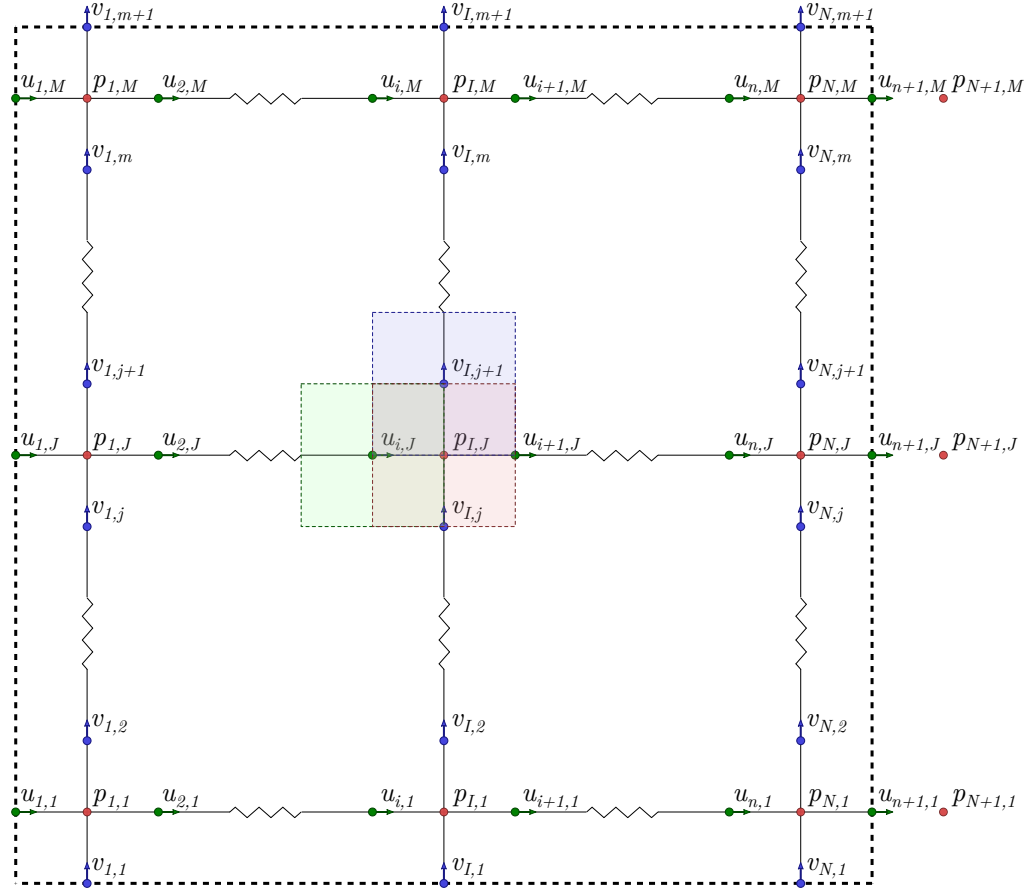


Figure 2.6: Staggered grid in two dimensions showing the locations of the nodes, indices and control volumes for u , v and p .

The control volumes drawn around the different node points in the centre of the figure shows the overlap. For the scalar node points, uppercase indexing letters I and J are used. For the velocities, the nodes are placed in between the scalar nodes and are therefore indexed with one uppercase and one lowercase letter.

2.2.4 SIMPLE-Algorithm

The Momentum equation is used for calculation of the velocity components, but another equation is needed to determine the pressure. A transformation of the continuity equation using the SIMPLE-algorithm provides such an equation [2]. In this section, the algorithm will be described in one dimension.

The SIMPLE-algorithm (*Semi-Implicit Method for Pressure-Linked Equations*) is as the name suggests a semi-implicit method, meaning it is based on a guessing and correcting scheme. The velocities and pressure are determined semi-implicitly at the same time by this guessing and correcting. The method was first proposed by Patankar and Spalding [22].

For an arbitrary property ϕ , the true value of ϕ can be expressed as a sum of a guessed value and a correction value. For a node with a known value or if the solution is converged, the correction value is zero. Equation (2.2.6) shown this relation when ϕ is the correct value, ϕ^* is the guessed value and ϕ' is the correction.

$$\phi = \phi^* + \phi' \quad (2.2.6)$$

Equations (2.2.7)-(2.2.9) shows the above expression for the true values of the pressure and velocities for a two dimensional model.

$$p = p^* + p' \quad (2.2.7)$$

$$u = u^* + u' \quad (2.2.8)$$

$$v = v^* + v' \quad (2.2.9)$$

The algorithm makes use of an initially guessed pressure to calculate the velocities, and then uses this velocities to calculate a pressure correction. This pressure correction is again used to calculate velocity corrections, and equations (2.2.7)-(2.2.9) are used to determine the true values of the velocities and the pressure. For an iterative scheme these "true" values will serve as the initial guess values in the next iteration. Figure 2.7 shows a visualisation of how the corrections are interacting. A visualisation of the whole SIMPLE-algorithm can be seen in figure 2.8.

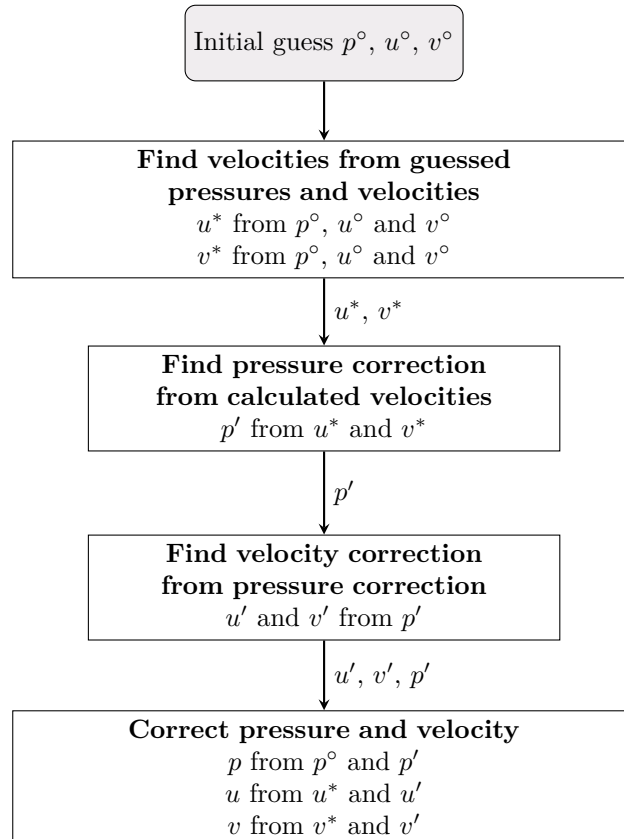


Figure 2.7: Correction cycle in the SIMPLE-algorithm

The velocities u^* and v^* in the first step in the visualisation in figure 2.7 are found from the discretised Momentum equation and the initial guesses of both the pressure and the velocities. Below follows the equations used for the correction of the pressure and velocities. The derivation of these equations are given in chapter 3, but the final equations and some brief steps are presented in the following sections.

2.2.4.1 The Velocity Correction Equation

The velocity correction equation can be obtained by replacing u with u^* and p with p^* in the Momentum equation. This new guessed velocity equation is then subtracted from the original Momentum equation to obtain equation (2.2.10). The same procedure is used to obtain a velocity correction for the v -velocity.

$$u_{i,J} = u_{i,J}^* - \frac{A_x}{a_{i,J}^{centre}} (p'_{I,J} - p'_{I-1,J}) \quad (2.2.10)$$

A_x is the control volume face area and a_i^{centre} is the coefficient multiplied with the centre node u_i in the Momentum equation. The velocity correction itself is equation (2.2.11).

$$u'_{i,J} = -\frac{A_x}{a_{i,J}^{centre}} (p'_{I,J} - p'_{I-1,J}) \quad (2.2.11)$$

and likewise for other velocity components.

2.2.4.2 The Pressure Correction Equation

The pressure correction equation comes from the Continuity equation. The velocity correction equation (2.2.10) is used and is inserted into the continuity equation. This yields the pressure correction equation, equation (2.2.12).

$$\nu_{I,J} p'_{I,J} + \nu_{I+1,J} p'_{I+1,J} + \nu_{I-1,J} p'_{I-1,J} + \nu_{I,J+1} p'_{I,J+1} + \nu_{I,J-1} p'_{I,J-1} = \beta_{I,J} \quad (2.2.12)$$

with

$$\nu_{I,J} = \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{centre}} + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{centre}} + \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{centre}} + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{centre}} \quad (2.2.13)$$

$$\nu_{I+1,J} = -\frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{centre}} \quad (2.2.14)$$

$$\nu_{I-1,J} = -\frac{\rho A_{x,i,J}^2}{a_{i,J}^{centre}} \quad (2.2.15)$$

$$\nu_{I,J+1} = -\frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{centre}} \quad (2.2.16)$$

$$\nu_{I,J-1} = -\frac{\rho A_{y,I,j}^2}{a_{I,j}^{centre}} \quad (2.2.17)$$

$$\beta_{I,J} = -A_x F_{x,e}^c + A_x F_{x,w}^c - A_y F_{y,n}^c + A_y F_{y,s}^c \quad (2.2.18)$$

The guessed velocities in the source term are taken as the values of the velocity at the previous iteration. The velocity terms in the source term therefore is equal to the continuity equation at the previous iteration. For a converged solution the pressure correction is zero, which fulfills the continuity equation.

2.2.4.3 Under-Relaxation Factors

To avoid divergence during the iterative scheme, the non-converged solution may be relaxed before it is sent to the next iteration.

Implementation of under-relaxation of the flow parameters makes sure the value that is sent to the next iteration is not overwhelmingly large even if the difference between the guessed value and the true value is vast. Under-relaxation is often crucial when the SIMPLE-algorithm is used since the method is a guess and correct method. If the correction would have been added directly and passed along, the value could have a large overshoot, and this may cause divergence. Instead a fraction of the correction is taken and added to the guess as shown in equations (2.2.19)-(2.2.21). Lowering the under-relaxation factors increases the computational time because only a fraction of the updated solution is passed on to the next iteration.

$$p^{new} = p^{\circ} + \alpha_p p' \quad (2.2.19)$$

$$u^{new} = \alpha_u (u^* + u') + (1 - \alpha_u) u^{\circ} \quad (2.2.20)$$

$$v^{new} = \alpha_v (v^* + v') + (1 - \alpha_v) v^{\circ} \quad (2.2.21)$$

The superscript new indicates the value that is passed on to the next iteration, $^{\circ}$ is the initial guess * is the secondary velocity guess calculated from the Momentum Equation, and $'$ signifies the correction.

It is suggested by Peric [23] and Peric et al. [24] that the optimal under-relaxation factors for the pressure and the velocities are given in equation (2.2.22).

$$\alpha_u + \alpha_p = 1 \quad (2.2.22)$$

The values of α_p and α_u are suggested to be approximately 0.2 and 0.8 respectively.

2.2.4.4 Visualisation of the Algorithm

Figure 2.8 shows a visualisation of the SIMPLE-algorithm in two dimensions with the calculation order and with arrows showing which parameters are passed on to the next step of the algorithm. The superscript $^{\circ}$ symbolises the initial guess or the value in the previous iteration. The coefficients a_u° and a_v° are functions of the values of the velocities at the previous iteration, and the source terms b_u° and b_v° are functions of the pressure at the previous iteration. * signifies the secondary velocity (guess) calculated from the Momentum Equation, and $'$ signifies the correction values. The superscript new indicates the value that is passed on to the next iteration. The implementation of the algorithm for the MATLAB model is given in chapter 4.

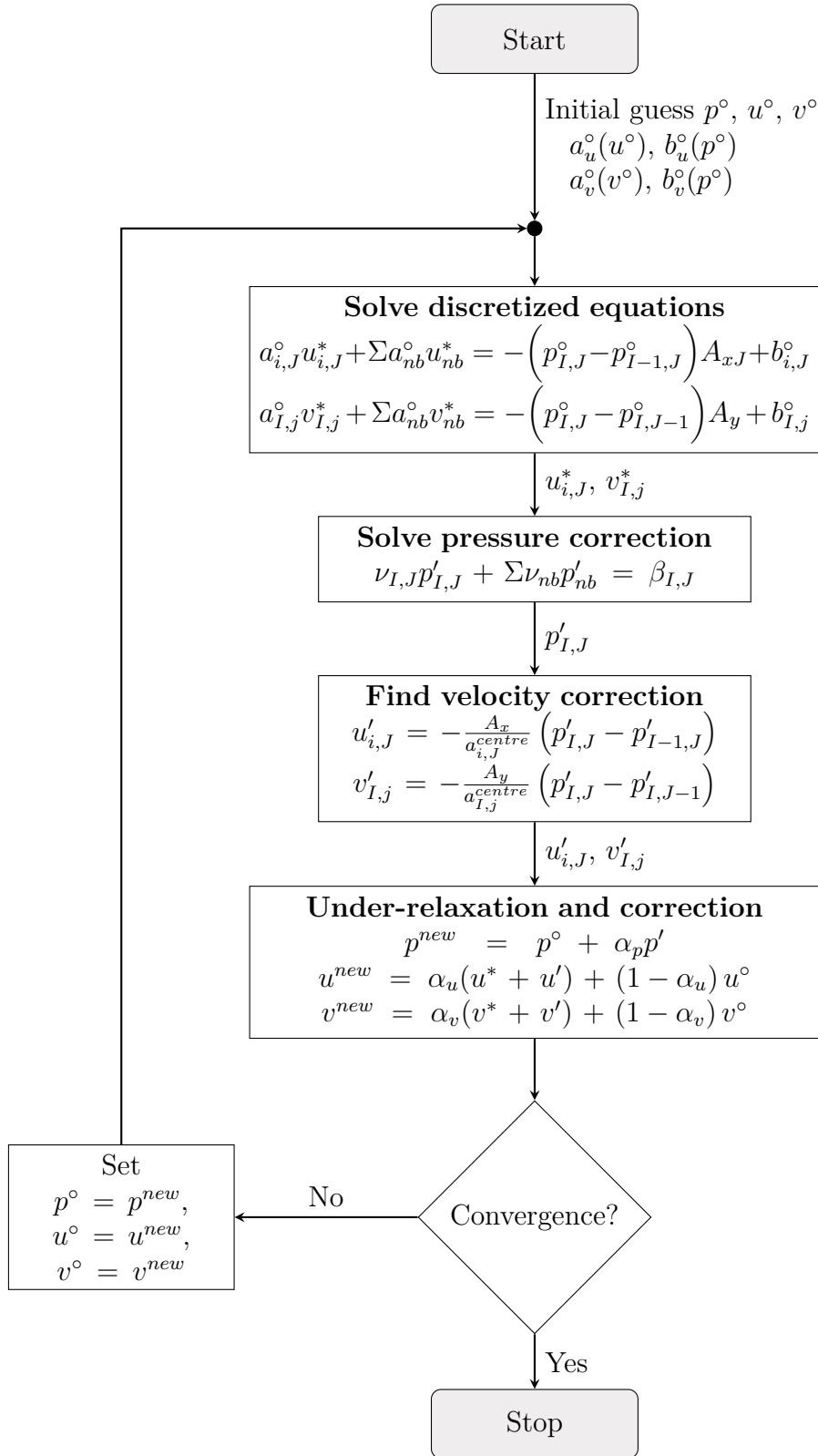


Figure 2.8: Visualisation of the SIMPLE-algorithm and the implemented procedure in MATLAB

2.3 Properties of Numerical Schemes

A numerical method that yields a result that is realistic and physical is characterised by a set of fundamental properties, where the three most important are the conservativeness, the boundedness and the transportiveness [2]. These properties are especially important when a small number of computational nodes are used. The accuracy of the discretisation schemes in the Finite Volume Method in relation to these properties is shortly accounted for in this section.

2.3.1 Conservativeness

Integrating the Momentum equation over the control volume CV yields a set of discretised equations. In the discretisation, terms for the flux across the control volume faces appear. Conservation of the flow across the domain is obtained when the flux out of a control volume is equal to the flux entering the next control volume [2]. This happens when the flux through a cell face is defined by the same expression for both the control volumes this cell face is a part of. The flux is then represented consistently, and the conservativeness is good.

2.3.2 Boundedness

The boundedness property states that if there is no source term, the boundary values of the solved property ϕ should be the limits for the possible solution values of ϕ [2]. This means that the value of the property within the domain should be between the inlet and the outlet value. In addition, in the discretised equation, the sign should be the same for all the coefficients a . This means that if an increase in the value of the property ϕ is observed at one node, the value of the property should also increase in the neighbouring nodes [2].

If a numerical scheme does not possess the boundedness property, the model may not converge, or the converged solution is "wavy" with over and undershoots [2].

2.3.3 Transportiveness

The Péclet number is a dimensionless number giving information about the rate of convection compared to the rate of diffusion. The Péclet number is defined as in equation (2.3.1)[2].

$$Pe = \frac{F}{D} = \frac{\rho u}{\Gamma/\delta x} \quad (2.3.1)$$

If the Péclet number is large, the flow is dominated by convection and the flow is less dependent on the downstream sections of the domain. This is often the case for engineering problems [25]. The upwind section is then cause for most of the influence on the node in question. The transportiveness of the numerical scheme is related to the value of the Péclet number and if the direction of influence in the domain is in accordance with the magnitude of Pe [2].

2.3.4 Properties and Accuracy of the Upwind Differencing Scheme

The Upwind Differencing scheme will be used to discretise the left hand side of the Momentum equation in this thesis. The discretisation scheme is conservative because the fluxes are expressed consistently over the whole domain. The coefficients a in the discretised momentum equation are always positive, and the boundedness criteria is therefore also met. Lastly, the transportiveness criteria is met because the direction of the flow is accounted for. Hence the Upwind Differencing Scheme should yields results that are realistic and physical.

The Upwind Differencing Scheme is using backwards differences, which come from Taylor series. The scheme is therefore first order accurate [2], and the errors associated with the neglected higher order terms may be significant. The results obtained are stable. Unfortunately, the Upwind Differencing Scheme is known for having issues with numerical diffusion errors, and can yield incorrect results if the flow is multi dimensional and the direction of the flow does not line up with one of the coordinate directions. The error that is caused by this is known as *false diffusion* because it appears like diffusion in the solution, and is often large for coarse grids [2]. Decreasing the size of the control volumes and creating a more refined solution grid may help, but this sacrifices memory and computational time.

The central differencing scheme is conservative and second order accurate, but not functional for convection-diffusion problems because it lacks the transportiveness property. The boundedness is also not good for cases where $Pe > 2$ [2]. Higher order methods may reduce the errors due to false diffusion, but they are generally less computationally stable [2].

2.4 Discretisation of the Domain

For numerical solution of the flow equations, the domain needs to be discretised to create points at which the fluid properties are calculated.

2.4.1 Control Volume

A control volume is drawn around each computational node in the domain. Cartesian coordinates are used, and the unit vectors for x - and y -direction is represented by figure 2.9. The positive flow direction of x - and y are left to right and bottom to top respectively, as shown in the figure.

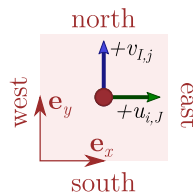


Figure 2.9: Schematic representation of the positive flow direction for the velocity components, as well as a representation of the orientation of the directions *west*, *east*, *north* and *south*.

Figure 2.10 shows a control volume drawn around the node point P . The width δx and height δy of the control volume are noted along with the cross-sectional areas A_x and A_y and the normal vectors \mathbf{n} . The same width δx and height δy are used for all the

control volumes in the domain. The control volume always has three dimensions, and figure 2.11 shows the same control volume with the third dimension also visible. The system depth δz is set to one in the two dimensional case. Note that the normal vectors in x - and z -directions have negative signs because of the angle the control volume is displayed from.

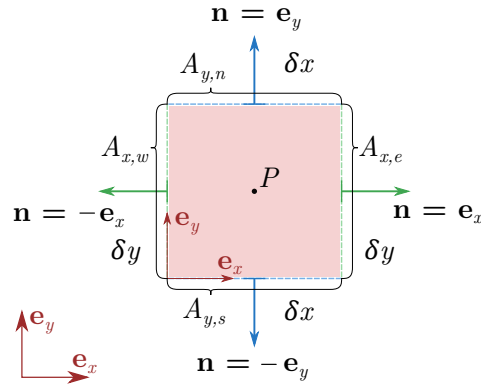


Figure 2.10: Control volume around computational node P with labels for the width δx and height δy of the control volume as well as the normal vectors \mathbf{n} and the cross-sectional areas A_x and A_y . The unit vectors \mathbf{e}_x and \mathbf{e}_y of the coordinate system are also shown.

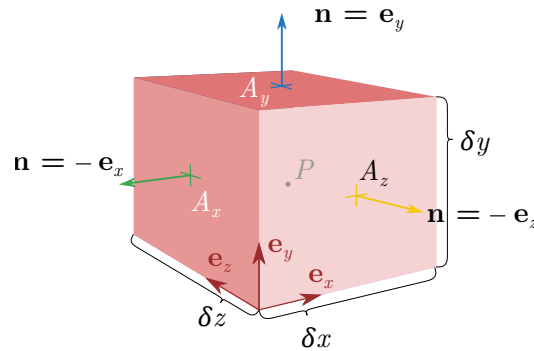


Figure 2.11: The control volume in figure 2.10 seen from a different angle and with labels in all three dimensions.

2.4.2 Global Indexing

Global indexing is used for the node points. This means that instead of using a vector position of the form (i, j) , all the node points are assigned a number from 1 to N where N is the number of nodes, following the expression in equation (2.4.1).

$$u(j, i) = u(i \cdot (j - 1) + i) \quad (2.4.1)$$

The counting can for example be started in the lower left corner of the domain, as shown in figure 2.12. As can be seen from the figure, the number of computational nodes in y -direction for the v -velocity is one less than for the scalars and the u -velocity. There is an equal number of computational nodes in x -direction for all the variables. The inlet velocity is located exactly at the inlet, while the outlet pressure is located one node outside of the computational domain. Note that in figure 2.6, the velocity

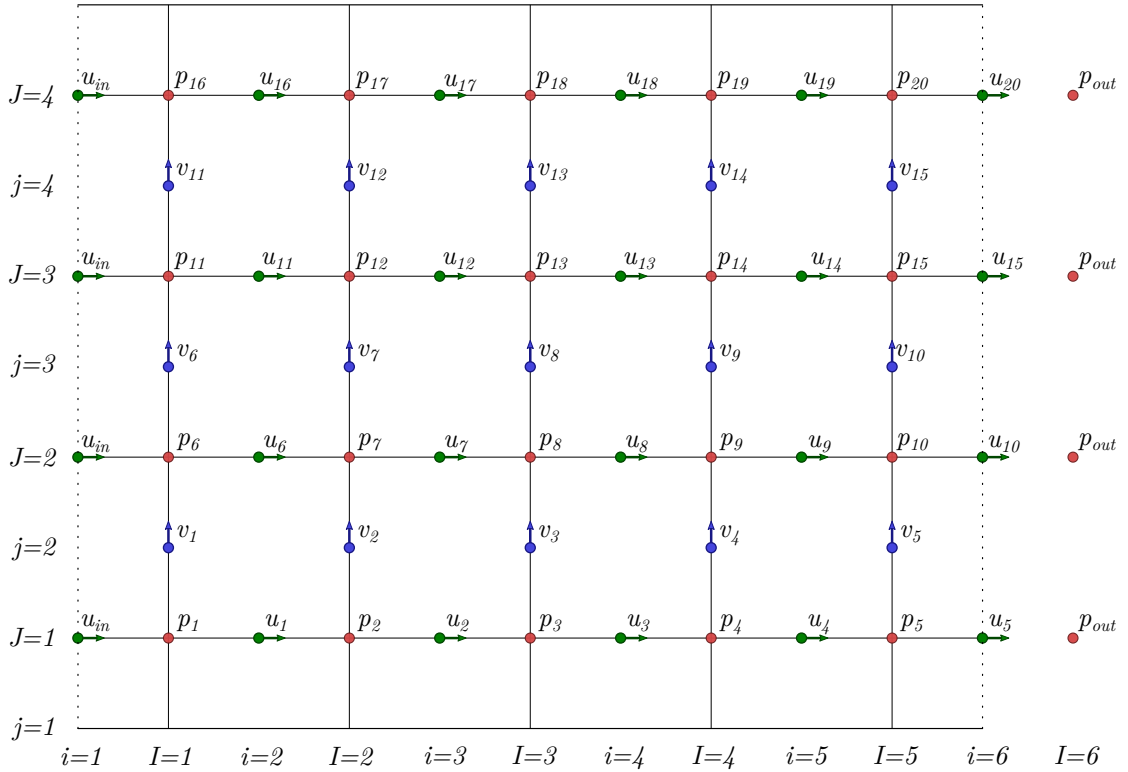


Figure 2.12: Example of a globally indexed system of node points.

node $u_{i,J}$ is located left of the scalar node $p_{I,J}$ and the velocity node $v_{I,j}$ is located below $p_{I,J}$. With the global indices in figure 2.12, the velocity nodes u_k and v_k are located right and above of the scalar node p_k instead.

By using this global indexing system the velocities and the pressure are stored in vectors of size $(1, N)$ instead of matrices of size (m, n) where m is the number of computational points in y -direction and n is the number of computational points in x -direction.

2.5 Non-Dimensional Equations

Non-dimensionalising the governing equations means that they are transformed in to a dimensionless form. This is done by dividing all parameters with a scale with the same unit as the parameter itself, removing all units.

Converting the flow equations to a dimensionless form can make the problem at hand easier to solve, and possible numerical difficulties in the solution are eliminated [2][25]. The difference between small or large values of parameters when the equation is made dimensionless give an indication to which terms are most important in the equation. For the regular equation, this is not the case, and larger values can simply mean that the property is measured in a larger scale. An example is pressure compared to velocity, where pressure has the unit Pa and is most often in order of magnitude of 10^5 . This may cause a problem if the velocity in m/s has a very low value, because the terms including the velocity are very small compared to the pressure, without being of less importance to the model. Such problems can be solved by converting the equations to their dimensionless form.

Dimensionless variables are noted with a circumflex $\hat{\chi}$ where χ is an arbitrary variable. Equation (2.5.1) shows the definition of the dimensionless variable $\hat{\chi}$.

$$\hat{\chi} = \frac{\chi}{\bar{\chi}} \quad (2.5.1)$$

where $\bar{\chi}$ is scale with the same unit as χ .

The dimensionless Continuity equation at steady state takes the same form as the regular Continuity equation, as seen in equation (2.5.2).

$$\hat{\nabla} \cdot (\hat{\rho}\hat{\mathbf{u}}) = 0 \quad (2.5.2)$$

The dimensionless Momentum equation will take the same form as the regular Momentum equation except the inverse of the Reynolds number appears as a coefficient in front of the diffusive terms as given in equation (2.5.3) [4][26].

$$\hat{\nabla} \cdot (\hat{\rho}\hat{\mathbf{u}}\hat{\mathbf{u}}) = -\hat{\nabla}\hat{p} - \frac{1}{Re}\hat{\nabla} \cdot \hat{\sigma} \quad (2.5.3)$$

The derivation of the dimensionless Continuity and Momentum Equations are given in section 3.4.

2.6 Solving Systems of Linear Algebraic Equations in MATLAB

As mentioned above, the Finite Volume method is used to convert the fluid flow equations into systems of linear algebraic equations. The system of linear algebraic equations for the velocity in one dimension is written as in equation (2.6.1). All the velocities u are represented in a vector due to the use of the global indexing system as described in section 2.4.2.

$$a_{i-1}u_{i-1} + a_iu_i + a_{i+1}u_{i+1} = b_i \quad (2.6.1)$$

where a are coefficients and b is the source term. The coefficients a can be sorted in the coefficient matrix U as shown in equation (2.6.2).

$$U = \begin{bmatrix} a_1 & a_2 & a_3 & & & \\ & \ddots & & & & \\ \dots & a_{i-1} & a_i & a_{i+1} & \dots & \\ & & & \ddots & & \\ & & & & a_{N-2} & a_{N-1} & a_N \end{bmatrix} \quad (2.6.2)$$

The source terms are stored in the vector b and u is the vector of velocities, and the system of linear algebraic equations can be written on the form $Uu = b$ as shown in equation (2.6.3) [27]. The first and last points 1 and N require boundary conditions.

$$\begin{bmatrix} a_1 & a_2 & a_3 & & & \\ & \ddots & & & & \\ \dots & a_{i-1} & a_i & a_{i+1} & \dots & \\ & & & \ddots & & \\ & & & & a_{N-2} & a_{N-1} & a_N \end{bmatrix} \begin{bmatrix} u_2 \\ \vdots \\ u_i \\ \vdots \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} b_2 \\ \vdots \\ b_i \\ \vdots \\ b_{N-1} \end{bmatrix} \quad (2.6.3)$$

A system of this form can be solved in MATLAB by using the *divided into* operator `\` as shown in equation (2.6.4) [28].

$$\mathbf{u} = \mathbf{A} \backslash \mathbf{b} \quad (2.6.4)$$

3

Discretisation

In this chapter, the the discretised Continuity, Momentum and SIMPLE-equations in two dimensions are obtained. The governing equations in two dimensions as given in section 2.1 are the starting point for the discretisation. The discretisation of the dimensionless Continuity and Momentum equations is also described. The governing equations in vector and component forms as well as some necessary theorems are given in appendix A. The discretisation of the two dimensional equations with all intermediate steps included can be found in appendix C.

The straight channel was first modelled in one dimension. The discretisation of the equations in one dimension is given in appendix B.

3.1 Continuity Equation

The Continuity Equation as given in equation (2.1.1) is integrated over the control volume CV . The transient term is omitted because of the steady state assumption. This yields equation (3.1.1).

$$\int_{CV} \nabla \cdot (\rho \mathbf{u}) dV = 0 \quad (3.1.1)$$

By the Gauss' theorem in equation (A.3.1) the volume integral can be converted to a surface integral, and equation (3.1.1) becomes equation (3.1.2).

$$\int_A \mathbf{n} \cdot (\rho \mathbf{u}) dA = 0 \quad (3.1.2)$$

In equation (3.1.2), $\mathbf{n} \cdot (\rho \mathbf{u})$ is the component of $\rho \mathbf{u}$ normal to the surface element dA .

The four surfaces are *west*, *east*, *south* and *north* for the two dimensional case as shown in figure 2.10. Splitting the surface integral into these four surfaces noted *w*, *e*, *s* and *n* yields equation (3.1.3).

$$\int_{A_{x,e}} \rho \mathbf{e}_x \cdot \mathbf{u} dA + \int_{A_{x,w}} \rho (-\mathbf{e}_x) \cdot \mathbf{u} dA + \int_{A_{y,n}} \rho \mathbf{e}_y \cdot \mathbf{u} dA + \int_{A_{y,s}} \rho (-\mathbf{e}_y) \cdot \mathbf{u} dA = 0 \quad (3.1.3)$$

Here *u* is the *x*-velocity component and *v* is the *y*-velocity component. Writing out the integrals yields equation (3.1.4).

$$\rho u_e A_{x,e} - \rho u_w A_{x,w} + \rho v_n A_{y,n} - \rho v_s A_{y,s} = 0 \quad (3.1.4)$$

where *u* is the *x*-velocity component and *v* is the *y*-velocity component. The Continuity Equation takes place at all the scalar nodes in the domain, which means that the cell face velocities u_e , u_w , v_s and v_n are located at the actual velocity nodes since a staggered grid is used. No interpolation is needed to determine the values of u_e , u_w , v_s and v_n . A visual representation of the staggered grid can be seen in figure 2.6.

The convective mass flux per unit are F^c is defined as in equation (3.1.5).

$$F_x^c = \rho u \quad F_y^c = \rho v \quad (3.1.5)$$

Since the control volume is rectangular with equally sized opposite cell faces, the area subscripts *w*, *e*, *s* and *n* may be omitted so that the equations only contains the terms A_x and A_y . The discretised Continuity equation is then equation (3.1.6).

$$F_{x,e}^c A_x - F_{x,w}^c A_x + F_{y,n}^c A_y - F_{y,s}^c A_y = 0 \quad (3.1.6)$$

3.2 Momentum Equation

The Momentum Equation in vector form is given in equation (2.1.3). The transient term is omitted because of the steady state assumption and the gravity term is omitted because the gravity is assumed to be acting in *z*-direction which is not taken into account in this thesis. This yields equation (3.2.1).

$$\nabla \cdot (\rho \mathbf{u} \mathbf{u}) = -\nabla p - \nabla \cdot \boldsymbol{\sigma} \quad (3.2.1)$$

The left and right hand side of the equation will be discretised separately before combining the equation in the end.

3.2.1 Left Hand Side

The left hand side of the momentum equation contains the convective terms of the equation, and the discretisation follow the same pattern as for the Continuity equation. **RHS** notes the right hand side of the equation. The integral over the control volume *CV* is taken to yield equation (3.2.2).

$$\int_{CV} \nabla \cdot (\rho \mathbf{u} \mathbf{u}) dV = \mathbf{RHS} \quad (3.2.2)$$

By Gauss' theorem in equation (A.3.1) the volume integral can again be converted to a surface integral. This yields equation (3.2.3).

$$\int_A \mathbf{n} \cdot (\rho \mathbf{u}\mathbf{u}) dA = \mathbf{RHS} \quad (3.2.3)$$

$\mathbf{n} \cdot (\rho \mathbf{u})$ is the component of $\rho \mathbf{u}$ normal to surface element dA . The four surfaces are the same as for the Continuity equation, *west*, *east*, *south* and *north* for the two dimensional case as shown in figure 2.10. The surface integral in equation (3.2.3) can be split into an integral for each of the normal surfaces noted w , e , s and n . The normal vectors around the control volume can be seen from figure 2.10. This yields equation (3.2.4).

$$\begin{aligned} \int_{A_{x,e}} \mathbf{e}_x \cdot \rho \mathbf{u}\mathbf{u} dA + \int_{A_{x,w}} -\mathbf{e}_x \cdot \rho \mathbf{u}\mathbf{u} dA \\ + \int_{A_{y,n}} \mathbf{e}_y \cdot \rho \mathbf{u}\mathbf{u} dA + \int_{A_{y,s}} -\mathbf{e}_y \cdot \rho \mathbf{u}\mathbf{u} dA = \mathbf{RHS} \end{aligned} \quad (3.2.4)$$

Taking the dot product of the unit vector \mathbf{e}_x or \mathbf{e}_y with one of the velocity vectors \mathbf{u} and integrating yields equation (3.2.5).

$$\rho (u\mathbf{u})_e A_{x,e} - \rho (u\mathbf{u})_w A_{x,w} + \rho (v\mathbf{u})_n A_{y,n} - \rho (v\mathbf{u})_s A_{y,s} = \mathbf{RHS} \quad (3.2.5)$$

where u is the x -velocity component and v is the y -velocity component. Equation (3.2.5) may then be multiplied with the unit vector \mathbf{e}_x or \mathbf{e}_y to obtain the x - and y - components of the equation. Since the control volume is rectangular with equally sized opposite cell faces, the area subscripts w , e , s and n may be omitted so that the equations only contains the terms A_x and A_y . The x - and y - components of equation (3.2.5) are given in equations (3.2.6) and (3.2.7) respectively.

$$\rho (uu)_e A_x - \rho (uu)_w A_x + \rho (vu)_n A_y - \rho (vu)_s A_y = \mathbf{RHS} \quad (3.2.6)$$

$$\rho (uv)_e A_x - \rho (uv)_w A_x + \rho (vv)_n A_y - \rho (vv)_s A_y = \mathbf{RHS} \quad (3.2.7)$$

Like for the Continuity equation, the convective mass flux per unit area F is introduced as shown in equation (3.2.8).

$$F_x = \rho u \quad F_y = \rho v \quad (3.2.8)$$

Unlike the coefficients F^c in the Continuity equation, the coefficients F are obtained from interpolation. This is because the velocities u_e , u_w , v_s and v_n in equations (3.2.6) and (3.2.7) are defined at the cell faces for the control volumes around the velocity nodes (see figure 2.6). No velocity value is calculated at these cell faces, but interpolation yields a value of the u - and v - velocity components. Figure 3.1 shows the velocity nodes $u_{i,J}$ and $v_{I,j}$ and the surrounding nodes with indices that are needed to define F around the nodes $u_{i,J}$ and $v_{I,j}$ for which the control volume CV is drawn around. The expressions for F for each component and each cell face are given in equations (3.2.9)-(3.2.16).

$$F_{x,e} = \rho \frac{u_{i,J} + u_{i+1,J}}{2} \quad (3.2.9) \quad F_{y,e} = \rho \frac{u_{i+1,J-1} + u_{i+1,J}}{2} \quad (3.2.13)$$

$$F_{x,w} = \rho \frac{u_{i-1,J} + u_{i,J}}{2} \quad (3.2.10) \quad F_{y,w} = \rho \frac{u_{i,J-1} + u_{i,J}}{2} \quad (3.2.14)$$

$$F_{x,n} = \rho \frac{v_{I-1,j+1} + v_{I,j+1}}{2} \quad (3.2.11) \quad F_{y,n} = \rho \frac{v_{I,j} + v_{I,j+1}}{2} \quad (3.2.15)$$

$$F_{x,s} = \rho \frac{v_{I-1,j} + v_{I,j}}{2} \quad (3.2.12) \quad F_{y,s} = \rho \frac{v_{I,j-1} + v_{I,j}}{2} \quad (3.2.16)$$

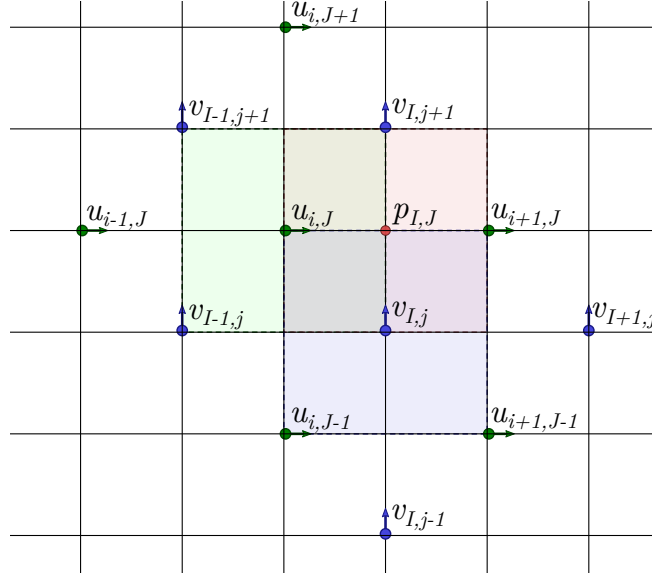


Figure 3.1: Node points with indices used in the expressions for the convective mass flux F .

Rewriting these with using the symbols P for the node point for which the control volume CV is drawn around and W , E , S and N for the neighbouring nodes yields equations (3.2.17)-(3.2.24).

$$F_{x,e} = \rho \frac{u_P + u_E}{2} \quad (3.2.17) \quad F_{y,e} = \rho \frac{u_{SE} + u_E}{2} \quad (3.2.21)$$

$$F_{x,w} = \rho \frac{u_W + u_P}{2} \quad (3.2.18) \quad F_{y,w} = \rho \frac{u_S + u_P}{2} \quad (3.2.22)$$

$$F_{x,n} = \rho \frac{v_{NW} + v_N}{2} \quad (3.2.19) \quad F_{y,n} = \rho \frac{v_P + v_N}{2} \quad (3.2.23)$$

$$F_{x,s} = \rho \frac{v_W + v_P}{2} \quad (3.2.20) \quad F_{y,s} = \rho \frac{v_S + v_P}{2} \quad (3.2.24)$$

The coefficients F are taken as knowns in the equation systems, and the velocities used to determine F are taken as the velocities at the previous iteration.

Equations (3.2.9)-(3.2.16) inserted into equations (3.2.6) and (3.2.7) yields equations (3.2.25) and (3.2.26) for the x - and y -components respectively.

$$F_{x,e}u_eA_x - F_{x,w}u_wA_x + F_{y,n}u_nA_y - F_{y,s}u_sA_y = \mathbf{RHS} \quad (3.2.25)$$

$$F_{x,e}v_eA_x - F_{x,w}v_wA_x + F_{y,n}v_nA_y - F_{y,s}v_sA_y = \mathbf{RHS} \quad (3.2.26)$$

The remaining velocity terms in equations (3.2.25) and (3.2.26) are still defined at the cell face of the control volumes. This is solved by use of the Upwind Differencing Scheme as presented in section 2.2.2. For this, the direction of the flow must be determined, which is done using the coefficients F . The \max operator is introduced, which makes it possible to represent the result for all the flow directions in one single equation.

Equation (3.2.27) is the discretised left hand side of the x -component momentum equation on coefficient form with the coefficients as given in equations 3.2.28-3.2.29.

$$a_P u_P + a_E u_E + a_W u_W + a_N u_N + a_S u_S = \mathbf{RHS} \quad (3.2.27)$$

with

$$a_P = -a_W - a_E - a_N - a_S + F_{x,e}A_x - F_{x,w}A_x + F_{x,n}A_y - F_{x,s}A_y \quad (3.2.28)$$

$$\begin{aligned} a_E &= -\max(0, -F_{x,e}A_x) & a_N &= -\max(0, -F_{x,n}A_y) \\ a_W &= -\max(F_{x,w}A_x, 0) & a_S &= -\max(F_{x,s}A_y, 0) \end{aligned} \quad (3.2.29)$$

Likewise, equation (3.2.30) is the discretised left hand side of the y -component momentum equation on coefficient form with the coefficients as given in equations 3.2.31-3.2.32.

$$a_P v_P + a_E v_E + a_W v_W + a_N v_N + a_S v_S = \mathbf{RHS} \quad (3.2.30)$$

with

$$a_P = -a_W - a_E - a_N - a_S + F_{y,e}A_x - F_{y,w}A_x + F_{y,n}A_y - F_{y,s}A_y \quad (3.2.31)$$

$$\begin{aligned} a_E &= -\max(0, -F_{y,e}A_x) & a_N &= -\max(0, -F_{y,n}A_y) \\ a_W &= -\max(F_{y,w}A_x, 0) & a_S &= -\max(F_{y,s}A_y, 0) \end{aligned} \quad (3.2.32)$$

3.2.2 Right Hand Side

The right hand side of the Momentum equation contains the diffusive terms of the equation. The shear stress term in equation (3.2.1) can be written out like in equation (3.2.33) for two dimensions. **LHS** denotes the left hand side of the momentum equation.

$$\mathbf{LHS} = -\nabla p - \frac{\partial \sigma_x}{\partial x} - \frac{\partial \sigma_y}{\partial y} \quad (3.2.33)$$

The x - and y - components of the Momentum equation in vector form can be obtained by taking the dot product with the unit vectors \mathbf{e}_x and \mathbf{e}_y respectively. The result are equations (3.2.34) and (3.2.35) respectively.

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} - \frac{\partial \sigma_{xx}}{\partial x} - \frac{\partial \sigma_{xy}}{\partial y} \quad (3.2.34)$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial y} - \frac{\partial \sigma_{yx}}{\partial x} - \frac{\partial \sigma_{yy}}{\partial y} \quad (3.2.35)$$

The expressions for the stress tensor components σ are inserted into equations (3.2.34) and (3.2.35). The expressions are given in appendix A. $\nabla \cdot \mathbf{u}$ is zero from the Continuity equation (2.1.2) for constant density, and equations (3.2.34) and (3.2.35) become equations (3.2.36) and (3.2.37).

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) \quad (3.2.36)$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) \quad (3.2.37)$$

Equations (3.2.36) and (3.2.37) can then be integrated over the control volume CV . For the diffusive terms, the volume integral is split, taking $dV = dA_x dx$ and $dV = dA_y dy$

as seen in equations (3.2.38) and (3.2.39).

$$\begin{aligned} \mathbf{LHS} = & - \int_{CV} \frac{\partial p}{\partial x} dV + \int_{\delta x} \int_{A_x} \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) dA_x dx \\ & + \int_{\delta y} \int_{A_y} \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) dA_y dy \quad (3.2.38) \end{aligned}$$

$$\begin{aligned} \mathbf{LHS} = & - \int_{CV} \frac{\partial p}{\partial y} dV + \int_{\delta x} \int_{A_x} \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) dA_x dx \\ & + \int_{\delta y} \int_{A_y} \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) dA_y dy \quad (3.2.39) \end{aligned}$$

The surface integrals are taken first, yielding equations (3.2.40) and (3.2.41).

$$\mathbf{LHS} = - \int_{CV} \frac{\partial p}{\partial x} dV + \int_{\delta x} \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) A_x dx + \int_{\delta y} \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) A_y dy \quad (3.2.40)$$

$$\mathbf{LHS} = - \int_{CV} \frac{\partial p}{\partial y} dV + \int_{\delta x} \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) A_x dx + \int_{\delta y} \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) A_y dy \quad (3.2.41)$$

The volume integral for the pressure terms are taken, and by the Fundamental Theorem of Calculus as given in equation (A.3.2), equations (3.2.40) and (3.2.41) become equations (3.2.42) and (3.2.43). Since the control volume is rectangular with equally sized opposite cell faces, the area subscripts w , e , s and n may be omitted so that the equations only contains the terms A_x and A_y .

$$\mathbf{LHS} = - \frac{\partial p}{\partial x} \Big|_P \delta x A_x + \mu \frac{\partial u}{\partial x} \Big|_e A_x - \mu \frac{\partial u}{\partial x} \Big|_w A_x + \mu \frac{\partial u}{\partial y} \Big|_n A_y - \mu \frac{\partial u}{\partial y} \Big|_s A_y \quad (3.2.42)$$

$$\mathbf{LHS} = - \frac{\partial p}{\partial y} \Big|_P \delta y A_y + \mu \frac{\partial v}{\partial x} \Big|_e A_x - \mu \frac{\partial v}{\partial x} \Big|_w A_x + \mu \frac{\partial v}{\partial y} \Big|_n A_y - \mu \frac{\partial v}{\partial y} \Big|_s A_y \quad (3.2.43)$$

The above gradients are approximated with central differences. For the pressure gradients equations (3.2.44) and (3.2.45) are used. The pressure points $p_{I,J}$, $p_{I-1,J}$ and $p_{I,J-1}$ then line up with existing pressure nodes. P corresponds to the centre node point for the velocity in this case, which are $u_{i,J}$ and $v_{I,j}$.

$$\frac{\partial p}{\partial x} \Big|_P = \frac{p_{I,J} - p_{I-1,J}}{\delta x} \quad (3.2.44)$$

$$\frac{\partial p}{\partial y} \Big|_P = \frac{p_{I,J} - p_{I,J-1}}{\delta y} \quad (3.2.45)$$

The velocity gradients are approximated with the central differences as shown in equations (3.2.46)-(3.2.53).

$$\frac{\partial u}{\partial x} \Big|_e = \frac{u_{i+1,J} - u_{i,J}}{\delta x} \quad (3.2.46)$$

$$\frac{\partial u}{\partial x} \Big|_w = \frac{u_{i,J} - u_{i-1,J}}{\delta x} \quad (3.2.47)$$

$$\frac{\partial u}{\partial y} \Big|_n = \frac{u_{i,J+1} - u_{i,J}}{\delta y} \quad (3.2.48)$$

$$\frac{\partial u}{\partial y} \Big|_s = \frac{u_{i,J} - u_{i,J-1}}{\delta y} \quad (3.2.49)$$

$$\frac{\partial v}{\partial x} \Big|_e = \frac{v_{I+1,j} - v_{I,j}}{\delta x} \quad (3.2.50)$$

$$\frac{\partial v}{\partial x} \Big|_w = \frac{v_{I,j} - v_{I-1,j}}{\delta x} \quad (3.2.51)$$

$$\frac{\partial v}{\partial y} \Big|_n = \frac{v_{I,j+1} - v_{I,j}}{\delta y} \quad (3.2.52)$$

$$\frac{\partial v}{\partial y} \Big|_s = \frac{v_{I,j} - v_{I,j-1}}{\delta y} \quad (3.2.53)$$

Since the velocity gradients are defined at the control volume faces w , e , s and n , the velocities in the right side of equations (3.2.46)-(3.2.53) line up with existing velocity nodes. The staggered grid indices are shown in figure 2.6.

The diffusion conductance D can be introduced, and is defined as in equation (3.2.54).

$$D_x = \frac{\mu}{\delta x} \quad D_y = \frac{\mu}{\delta y} \quad (3.2.54)$$

Inserting the gradients in equations (3.2.44)-(3.2.53) and the diffusion conductance D into equations (3.2.42) and (3.2.43) yields equations (3.2.55) and (3.2.56) for the x - and y -component respectively.

$$\begin{aligned} \text{LHS} = & -(p_{I,J} - p_{I-1,J})A_x + D_x A_x (u_{i+1,J} - u_{i,J}) - D_x A_x (u_{i,J} - u_{i-1,J}) \\ & + D_y A_y (u_{i,J+1} - u_{i,J}) - D_y A_y (u_{i,J} - u_{i,J-1}) \end{aligned} \quad (3.2.55)$$

$$\begin{aligned} \text{LHS} = & -(p_{I,J} - p_{I,J-1})A_y + D_x A_x (v_{I+1,j} - v_{I,j}) - D_x A_x (v_{I,j} - v_{I-1,j}) \\ & + D_y A_y (v_{I,j+1} - v_{I,j}) - D_y A_y (v_{I,j} - v_{I,j-1}) \end{aligned} \quad (3.2.56)$$

3.2.3 Combined Momentum Equation

The left and right side of the momentum equation can be put back together and rearranged as given in coefficient form below.

Equation (3.2.57) is the discretised x -component momentum equation with the coefficients as given in equation (3.2.58).

$$a_{i,J}u_{i,J} + a_{i+1,J}u_{i+1,J} + a_{i-1,J}u_{i-1,J} + a_{i,J+1}u_{i,J+1} + a_{i,J-1}u_{i,J-1} = b_{i,J} \quad (3.2.57)$$

with

$$\begin{aligned} a_{i,J} &= -a_{i+1,J} - a_{i-1,J} - a_{i,J+1} - a_{i,J-1} + F_{x,e}A_x - F_{x,w}A_y + F_{y,n}A_y - F_{y,s}A_y \\ a_{i+1,J} &= -\max(0, -F_{x,e}A_x) - D_x A_x \\ a_{i-1,J} &= -\max(F_{x,w}A_y, 0) - D_x A_y \\ a_{i,J+1} &= -\max(0, -F_{y,n}A_y) - D_y A_y \\ a_{i,J-1} &= -\max(F_{y,s}A_y, 0) - D_y A_y \\ b_{i,J} &= -(p_{I,J} - p_{I-1,J})A_x \end{aligned} \quad (3.2.58)$$

Likewise, equation (3.2.59) is the discretised y -component momentum equation with the coefficients as given in equation (3.2.60).

$$a_{I,j}v_{I,j} + a_{I+1,j}v_{I+1,j} + a_{I-1,j}v_{I-1,j} + a_{I,j+1}v_{I,j+1} + a_{I,j-1}v_{I,j-1} = b_{I,j} \quad (3.2.59)$$

with

$$\begin{aligned}
a_{I,j} &= -a_{I+1,j} - a_{I-1,j} - a_{I,j+1} - a_{I,j-1} + F_{x,e}A_x - F_{x,w}A_y + F_{y,n}A_y - F_{y,s}A_y \\
a_{I+1,j} &= -\max(0, -F_{x,e}A_x) - D_xA_x \\
a_{I-1,j} &= -\max(F_{x,w}A_y, 0) - D_xA_y \\
a_{I,j+1} &= -\max(0, -F_{y,n}A_y) - D_yA_y \\
a_{I,j-1} &= -\max(F_{y,s}A_y, 0) - D_yA_y \\
b_{I,j} &= -\left(p_{I,J} - p_{I,J-1}\right)A_y
\end{aligned} \tag{3.2.60}$$

3.3 SIMPLE-Equations

In this section the velocity correction and pressure correction equations for use with the SIMPLE-algorithm are derived.

3.3.1 Velocity Correction Equation

The discretised Momentum equation can be rewritten as an equation for the guessed variables as described in section 2.2.4 by exchanging all the variables with the guessed equivalents, for example u with u^* and p with p° . In this case, the "guessed" velocities u^* and v^* are the velocities obtained from the Momentum equation earlier in the algorithm for the same iteration, and the guessed pressure p° is the pressure from the previous iteration. The velocity correction equation can then be obtained by taking the discretised Momentum equation for u and subtracting the Momentum equation for the "guessed" velocity u^* as in equation (3.3.1).

$$\begin{aligned}
a_{i,J}(u_{i,J} - u_{i,J}^*) + a_{i+1,J}(u_{i+1,J} - u_{i+1,J}^*) + a_{i-1,J}(u_{i-1,J} - u_{i-1,J}^*) \\
+ a_{i,J+1}(u_{i,J+1} - u_{i,J+1}^*) + a_{i,J-1}(u_{i,J-1} - u_{i,J-1}^*) \\
= \left(-p_{I,J} + p_{I-1,J} + p_{I,J}^\circ - p_{I-1,J}^\circ \right) A_x + \cancel{b_{i,J}^\rho} - \cancel{b_{i,J}^\rho} \tag{3.3.1}
\end{aligned}$$

From the definition of the correction values in section 2.2.4 it follows that the terms of the form $u - u^*$ are equal to the velocity correction u' and the terms of the form $p - p^\circ$ are equal to the pressure correction p' . The velocity correction in the centre node $u'_{i,J}$ is kept while the velocity corrections in all the neighbouring nodes are omitted. This yields the velocity correction equation (3.3.2) for the velocity node $u_{i,J}$.

$$u'_{i,J} = -\frac{A_x}{a_{i,J}^{centre}} \left(p'_{I,J} - p'_{I-1,J} \right) \tag{3.3.2}$$

$a_{i,J}^{centre}$ is the velocity equation coefficient for the node $u_{i,J}$. Equation (3.3.3) shows the v -velocity correction for the node point $v_{I,j}$ which can be obtained in the same way.

$$v'_{I,j} = -\frac{A_y}{a_{I,j}^{centre}} \left(p'_{I,J} - p'_{I,J-1} \right) \tag{3.3.3}$$

The true velocity value is then obtained by equation (2.2.9) as written out in equations (3.3.4) and (3.3.5).

$$u_{i,J} = u_{i,J}^* - \frac{A_x}{a_{i,J}^{centre}} \left(p'_{I,J} - p'_{I-1,J} \right) \quad (3.3.4)$$

$$v_{I,j} = v_{I,j}^* - \frac{A_y}{a_{I,j}^{centre}} \left(p'_{I,J} - p'_{I,J-1} \right) \quad (3.3.5)$$

3.3.2 Pressure Correction Equation

The pressure correction equation is obtained from the Continuity equation (3.3.6) and the velocity correction equations (3.3.4) and (3.3.5).

$$\rho u_{i+1,J} A_x - \rho u_{i,J} A_x + \rho v_{I,j+1} A_y - \rho v_{I,j} A_y = 0 \quad (3.3.6)$$

The velocities u and v in equation (3.3.6) are replaced with equations (3.3.4) and (3.3.5) to yield equation (3.3.7). At the boundaries of the domain, one or more of the velocity terms in equation (3.3.6) are known. In this case, the known velocity term is not replaced by equations (3.3.4) or (3.3.5), but the known velocity value is kept. This is because the velocity correction is zero for a node with a known velocity [2].

$$\begin{aligned} & \rho A_x \left(u_{i+1,J}^* - \frac{A_x}{a_{i+1,J}^{centre}} \left(p'_{I+1,J} - p'_{I,J} \right) \right) \\ & - \rho A_x \left(u_{i,J}^* - \frac{A_x}{a_{i,J}^{centre}} \left(p'_{I,J} - p'_{I-1,J} \right) \right) + \rho A_y \left(v_{I,j+1}^* - \frac{A_y}{a_{I,j+1}^{centre}} \left(p'_{I,J+1} - p'_{I,J} \right) \right) \\ & - \rho A_y \left(v_{I,j}^* - \frac{A_y}{a_{I,j}^{centre}} \left(p'_{I,J} - p'_{I,J-1} \right) \right) = 0 \end{aligned} \quad (3.3.7)$$

Rearranging equation (3.3.7), collecting all the pressure correction terms on one side and all the guessed velocities on the other yields equation (3.3.8) with the coefficients in equation (3.3.9).

$$\nu_{I,J} p'_{I,J} + \nu_{I+1,J} p'_{I+1,J} + \nu_{I-1,J} p'_{I-1,J} + \nu_{I,J+1} p'_{I,J+1} + \nu_{I,J-1} p'_{I,J-1} = \beta_{I,J} \quad (3.3.8)$$

with

$$\begin{aligned} \nu_{I,J} &= \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{centre}} + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{centre}} + \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{centre}} + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{centre}} \\ \nu_{I+1,J} &= - \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{centre}} \\ \nu_{I-1,J} &= - \frac{\rho A_{x,i,J}^2}{a_{i,J}^{centre}} \\ \nu_{I,J+1} &= - \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{centre}} \\ \nu_{I,J-1} &= - \frac{\rho A_{y,I,j}^2}{a_{I,j}^{centre}} \\ \beta_{I,J} &= - A_x \rho u_{x,e}^* + A_x \rho u_{x,w}^* - A_y \rho u_{y,n}^* + A_y \rho u_{y,s}^* \end{aligned} \quad (3.3.9)$$

The source term takes the form of the Continuity equation and is equal to zero for the converged solution, since all the pressure correction terms are zero for the converged solution. The velocities in the source term are guessed velocities that are taken as the velocity values obtained from the Momentum equation

Figure 3.2 shows the numerical "molecule" for the pressure correction equation, showing where each term is located on the staggered grid. The velocity terms in the source term are located at the cell faces of the pressure control volume, and these cell faces line up with the velocity nodes.

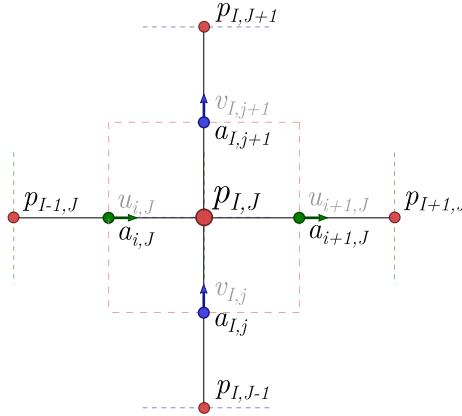


Figure 3.2: Shape of pressure correction equation "molecule" in two dimensions.

3.4 Dimensionless Equations

In this section the derivation of the two dimensional discretised equations given in sections 3.1 - 3.3 are repeated for making these equations dimensionless. The steps of the discretisation themselves are identical to what is given in sections 3.1 - 3.3, and only the main steps are repeated in this section.

The dimensionless Continuity equation, and therefore also the dimensionless pressure correction equation will take the same form as for the ordinary variables. The dimensionless Momentum equation will take close to the same form as the dimensional version, but with a factor $\frac{1}{Re}$ before the viscous terms as shown in equation (3.4.1) [29].

$$\hat{\nabla} \cdot (\hat{\rho} \hat{u} \hat{u}) = -\hat{\nabla} \hat{p} - \frac{1}{Re} \hat{\nabla} \cdot \hat{\sigma} \quad (3.4.1)$$

A diacritic *circumflex* $\hat{\cdot}$ is used to indicate that the variable ϕ is dimensionless.

3.4.1 Definition of dimensionless variables

Below follows an overview of the different dimensionless variables, lengths and operators. As given in equation (2.5.1), the numerator is the original parameter and the denominator is the scale for that parameter in the definitions of each dimensionless parameter.

The pressure is adjusted by subtracting the outlet pressure as defined in equation (3.4.2) before it is made dimensionless by dividing with an appropriate scale. \tilde{p} is the adjusted pressure and is zero at the outlet.

$$\tilde{p} = p - p_{out} \quad (3.4.2)$$

3.4.1.1 Variables

The dimensionless variables for the velocity vector $\hat{\mathbf{u}}$, adjusted pressure \tilde{p} , viscosity μ and density ρ is given in equations (3.4.3)-(3.4.6).

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{u_{in}} \quad (3.4.3)$$

$$\hat{p} = \frac{\tilde{p}}{\bar{p}} \quad (3.4.4)$$

$$\hat{\mu} = \frac{\mu}{\mu_{in}} = \frac{\mu}{\mu} \quad (3.4.5)$$

$$\hat{\rho} = \frac{\rho}{\rho_{in}} = \frac{\rho}{\rho} \quad (3.4.6)$$

u_{in} is the scaling factor for the velocities and is the inlet velocity. If the inlet velocity is not constant, the velocity scale is the average velocity at the inlet. All components of the velocity are normalised with the same scale. A diacritic *macron* $\bar{}$ is used to signify the scale for a variable. The pressure scale \bar{p} is given by equation (3.4.7) [16].

$$\bar{p} = \rho u_{in}^2 \quad (3.4.7)$$

ρ_{in} is the inlet density and μ_{in} is the inlet viscosity. The density and viscosity are constant over the domain and are expressed this way for simplicity in the derivation despite ρ_{in} being equal to ρ and μ_{in} being equal to μ .

3.4.1.2 Length, area, volume

All the length units are scaled with the same parameter, which is taken to be the hydraulic diameter D_{hyd} . δ_x , δ_y and δ_z are the width, height and depth of the control volume respectively. The definitions and directions of δ_x , δ_y and δ_z as well as A_x and A_y can be seen from figure 2.11.

Equations (3.4.8) - (3.4.19) show the definitions of the dimensionless versions of all length scales and variants of length scales.

$$\hat{x} = \frac{x}{D_{hyd}} \quad (3.4.8) \quad \hat{y} = \frac{y}{D_{hyd}} \quad (3.4.12) \quad \hat{z} = \frac{z}{D_{hyd}} \quad (3.4.16)$$

$$d\hat{x} = \frac{dx}{D_{hyd}} \quad (3.4.9) \quad d\hat{y} = \frac{dy}{D_{hyd}} \quad (3.4.13) \quad d\hat{z} = \frac{dz}{D_{hyd}} \quad (3.4.17)$$

$$\delta\hat{x} = \frac{\delta x}{D_{hyd}} \quad (3.4.10) \quad \delta\hat{y} = \frac{\delta y}{D_{hyd}} \quad (3.4.14) \quad \delta\hat{z} = \frac{\delta z}{D_{hyd}} \quad (3.4.18)$$

$$\frac{\partial}{\partial\hat{x}} = D_{hyd} \frac{\partial}{\partial x} \quad (3.4.11) \quad \frac{\partial}{\partial\hat{y}} = D_{hyd} \frac{\partial}{\partial y} \quad (3.4.15) \quad \frac{\partial}{\partial\hat{z}} = D_{hyd} \frac{\partial}{\partial z} \quad (3.4.19)$$

The cross sectional areas are given in equations (3.4.20)-(3.4.21) and the volume of the control volume is given in equation (3.4.22). Since the equations will be derived for two dimensions, the cross-sectional area in z -direction is not included.

$$\hat{A}_x = \delta\hat{y} \delta\hat{z} = \frac{1}{D_{hyd}^2} \delta y \delta z = \frac{1}{D_{hyd}^2} A_x \quad (3.4.20)$$

$$\hat{A}_y = \delta\hat{x} \delta\hat{z} = \frac{1}{D_{hyd}^2} \delta x \delta z = \frac{1}{D_{hyd}^2} A_y \quad (3.4.21)$$

$$\hat{V} = \delta\hat{x} \delta\hat{y} \delta\hat{z} = \frac{1}{D_{hyd}^3} \delta x \delta y \delta z = \frac{1}{D_{hyd}^3} V \quad (3.4.22)$$

Similarly the differentials of A and V are given in equations (3.4.23) and (3.4.24).

$$d\hat{A} = \frac{1}{D_{hyd}^2} dA \quad (3.4.23)$$

$$d\hat{V} = \frac{1}{D_{hyd}^3} dV \quad (3.4.24)$$

3.4.1.3 Operators, tensors

The ∇ operator is defined by equation (3.4.25) [30].

$$\nabla = \mathbf{i} \frac{\partial}{\partial x} + \mathbf{j} \frac{\partial}{\partial y} + \mathbf{k} \frac{\partial}{\partial z} \quad (3.4.25)$$

Since $\frac{\partial}{\partial \hat{x}} = D_{hyd} \frac{\partial}{\partial x}$ etc., the dimensionless ∇ operator is given by equation (3.4.26).

$$\hat{\nabla} = D_{hyd} \nabla \quad (3.4.26)$$

The stress tensors used in the 2D-equations are defined in equations (3.4.27)-(3.4.29), with $\nabla \cdot \mathbf{u} = 0$ from the Continuity equation (2.1.2).

$$\sigma_{xx} = -\mu \left[2 \frac{\partial u}{\partial x} - \frac{2}{3} (\nabla \cdot \mathbf{u}) \right] = -2\mu \frac{\partial u}{\partial x} \quad (3.4.27)$$

$$\sigma_{yy} = -\mu \left[2 \frac{\partial v}{\partial y} - \frac{2}{3} (\nabla \cdot \mathbf{u}) \right] = -2\mu \frac{\partial v}{\partial y} \quad (3.4.28)$$

$$\sigma_{xy} = -\mu \left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right] \quad (3.4.29)$$

The dimensionless stress tensor is defined in (3.4.30) where $\bar{\sigma}$ is the scale.

$$\hat{\sigma} = \frac{\sigma}{\bar{\sigma}} \quad (3.4.30)$$

The expressions for the stress tensor components in equations (3.4.27)-(3.4.29) are inserted into equation (3.4.30). The result is shown in equations (3.4.31)-(3.4.33).

$$\hat{\sigma}_{\hat{x}\hat{x}} = -\frac{1}{\bar{\sigma}} 2\mu \frac{\partial u}{\partial x} = -\frac{1}{\bar{\sigma}} \frac{\mu u_{in}}{D_{hyd}} 2\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{x}} \quad (3.4.31)$$

$$\hat{\sigma}_{\hat{y}\hat{y}} = -\frac{1}{\bar{\sigma}} 2\mu \frac{\partial v}{\partial y} = -\frac{1}{\bar{\sigma}} \frac{\mu u_{in}}{D_{hyd}} 2\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{y}} \quad (3.4.32)$$

$$\hat{\sigma}_{\hat{x}\hat{y}} = -\frac{1}{\bar{\sigma}} \mu \left[\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right] = -\frac{1}{\bar{\sigma}} \frac{\mu u_{in}}{D_{hyd}} \hat{\mu} \left[\frac{\partial \hat{u}}{\partial \hat{x}} + \frac{\partial \hat{v}}{\partial \hat{y}} \right] \quad (3.4.33)$$

To make the right hand side in the above equations dimensionless, the scale $\bar{\sigma}$ is defined as in equation (3.4.34).

$$\bar{\sigma} = \frac{\mu u_{in}}{D_{hyd}} \quad (3.4.34)$$

The dimensionless stress tensor components are then defined as in equations (3.4.35) - (3.4.37).

$$\hat{\sigma}_{\hat{x}\hat{x}} = -2\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{x}} \quad (3.4.35)$$

$$\hat{\sigma}_{\hat{y}\hat{y}} = -2\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{y}} \quad (3.4.36)$$

$$\hat{\sigma}_{\hat{x}\hat{y}} = -\hat{\mu} \left[\frac{\partial \hat{u}}{\partial \hat{x}} + \frac{\partial \hat{v}}{\partial \hat{y}} \right] \quad (3.4.37)$$

3.4.2 Variables as Functions of their Dimensionless Form

All variables, geometrical length scales, operators and tensors expressed with dimensionless parameters for interchanging in the transport equations are given in equations (3.4.38)-(3.4.55).

$$\mathbf{u} = u_{in} \hat{\mathbf{u}} \quad (3.4.38) \quad A_x = D_{hyd}^2 \hat{A}_x \quad (3.4.47)$$

$$\tilde{p} = \rho u_{in}^2 \hat{p} \quad (3.4.39) \quad A_y = D_{hyd}^2 \hat{A}_y \quad (3.4.48)$$

$$\mu = \mu \hat{\mu} \quad (3.4.40) \quad dA = D_{hyd}^2 d\hat{A} \quad (3.4.49)$$

$$\rho = \rho \hat{\rho} \quad (3.4.41) \quad V = D_{hyd}^3 \hat{V} \quad (3.4.50)$$

$$\delta x = D_{hyd} \delta \hat{x} \quad (3.4.42) \quad dV = D_{hyd}^3 d\hat{V} \quad (3.4.51)$$

$$\delta y = D_{hyd} \delta \hat{y} \quad (3.4.43) \quad \sigma = \bar{\sigma} \hat{\sigma} \quad (3.4.52)$$

$$\frac{\partial}{\partial x} = \frac{1}{D_{hyd}} \frac{\partial}{\partial \hat{x}} \quad (3.4.44) \quad \sigma_{xx} = -\frac{\mu u_{in}}{D_{hyd}} 2\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{x}} \quad (3.4.53)$$

$$\frac{\partial}{\partial y} = \frac{1}{D_{hyd}} \frac{\partial}{\partial \hat{y}} \quad (3.4.45) \quad \sigma_{yy} = -\frac{\mu u_{in}}{D_{hyd}} 2\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{y}} \quad (3.4.54)$$

$$\nabla = \frac{1}{D_{hyd}} \hat{\nabla} \quad (3.4.46) \quad \sigma_{xy} = -\frac{\mu u_{in}}{D_{hyd}} \hat{\mu} \left[\frac{\partial \hat{u}}{\partial \hat{x}} + \frac{\partial \hat{v}}{\partial \hat{y}} \right] \quad (3.4.55)$$

3.4.3 Dimensionless Continuity Equation

The Continuity equation with the transient term deleted is given in equation (2.1.2). With the dimensionless parameters from equations (3.4.38)-(3.4.55) inserted, the continuity equation becomes equation (3.4.56).

$$\frac{1}{D_{hyd}} \hat{\nabla} \cdot (\rho u_{in} \hat{\rho} \hat{\mathbf{u}}) = 0 \quad (3.4.56)$$

Integration over the dimensionless control volume $\hat{C}V$ yields equation (3.4.57), and Gauss' theorem given in equation (A.3.1) is again applied yielding equation (3.4.58). Equation (3.4.58) is then divided with the factor $\frac{\rho u_{in}}{D_{hyd}}$ which yields equation (3.4.59). Equation (3.4.59) takes the same form as equation (3.1.2), and the rest of the discretisation of the dimensionless Continuity equation follows the same steps as in section

3.1.

$$\int_{\hat{C}\hat{V}} \frac{1}{D_{hyd}} \hat{\nabla} \cdot (\rho u_{in} \hat{\rho} \hat{\mathbf{u}}) d\hat{V} = 0 \quad (3.4.57)$$

$$\frac{\rho u_{in}}{D_{hyd}} \int_{\hat{A}} \mathbf{n} \cdot (\hat{\rho} \hat{\mathbf{u}}) d\hat{A} = 0 \quad (3.4.58)$$

$$\int_{\hat{A}\mathbf{n}} (\hat{\rho} \hat{\mathbf{u}}) d\hat{A} = 0 \quad (3.4.59)$$

Equation (3.4.60) is the dimensionless continuity equation with \hat{F}^c as defined in equation (3.4.61)

$$\hat{F}_{x,e}^c \hat{A}_{x,e} - \hat{F}_{x,w}^c \hat{A}_{x,w} + \hat{F}_{y,n}^c \hat{A}_{y,n} - \hat{F}_{y,s}^c \hat{A}_{y,s} = 0 \quad (3.4.60)$$

with

$$\hat{F}_x^c = \hat{\rho} \hat{u} \quad \hat{F}_y^c = \hat{\rho} \hat{v} \quad (3.4.61)$$

3.4.4 Dimensionless Momentum Equation

The momentum equation with the transient term delited and the gravity term neglected is given in equation (3.2.1). With the dimensionless variables given in equations (3.4.38)-(3.4.55) inserted, the Momentum equation becomes equation (3.4.62).

$$\frac{\rho u_{in}^2}{D_{hyd}} \hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}} \hat{\mathbf{u}}) = -\frac{\bar{p}}{D_{hyd}} \hat{\nabla} \hat{p} - \frac{\bar{\sigma}}{D_{hyd}} \hat{\nabla} \cdot \hat{\sigma} \quad (3.4.62)$$

The scales for the pressure $\bar{p} = \rho u_{in}^2$ and the stress tensor $\bar{\sigma} = \frac{\mu u_{in}}{D_{hyd}}$ can be inserted to yield equation (3.4.63).

$$\frac{\rho u_{in}^2}{D_{hyd}} \hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}} \hat{\mathbf{u}}) = -\frac{\rho u_{in}^2}{D_{hyd}} \hat{\nabla} \hat{p} - \frac{\mu u_{in}}{D_{hyd}^2} \hat{\nabla} \cdot \hat{\sigma} \quad (3.4.63)$$

Equation (3.4.63) is then multiplied with the factor $\frac{D_{hyd}}{\rho u_{in}^2}$ to yield equation (3.4.64), which is equal to equation (3.4.1).

$$\hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}} \hat{\mathbf{u}}) = -\hat{\nabla} \hat{p} - \frac{\mu}{\rho u_{in} D_{hyd}} \hat{\nabla} \cdot \hat{\sigma} \quad (3.4.64)$$

3.4.4.1 Left Hand Side

The left side of equation (3.4.64) can be integrated directly over the dimensionless control volume $\hat{C}\hat{V}$ to yield equation (3.4.65). By Gauss' theorem in equation (A.3.1) equation (3.4.66) is obtained.

$$\int_{\hat{C}\hat{V}} \hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}} \hat{\mathbf{u}}) d\hat{V} = \mathbf{RHS} \quad (3.4.65)$$

$$\int_{\hat{A}} \mathbf{n} \cdot (\hat{\rho} \hat{\mathbf{u}} \hat{\mathbf{u}}) d\hat{A} = \mathbf{RHS} \quad (3.4.66)$$

Equation (3.4.66) takes the same form as equation (3.2.3), and the rest of the discretisation of the left hand side of the Momentum equation follows the same steps as in section 3.2.

The dimensionless convective mass flux \hat{F} are defined the same way as in equations (3.2.9)-(3.2.16). The left side of the x -component of the dimensionless Momentum equation is given in equation (3.4.67) with the coefficients in equations (3.4.68)-(3.4.69).

$$\hat{a}_P \hat{u}_P + \hat{a}_E \hat{u}_E + \hat{a}_W \hat{u}_W + \hat{a}_y \hat{u}_N + \hat{a}_S \hat{u}_S = \mathbf{RHS} \quad (3.4.67)$$

with

$$\hat{a}_P = -\hat{a}_W - \hat{a}_E - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_x + \hat{F}_{x,n} \hat{A}_y - \hat{F}_{x,s} \hat{A}_y \quad (3.4.68)$$

$$\begin{aligned} \hat{a}_E &= -\max(0, -\hat{F}_{x,e} \hat{A}_x) & \hat{a}_N &= -\max(0, -\hat{F}_{x,n} \hat{A}_y) \\ \hat{a}_W &= -\max(\hat{F}_{x,w} \hat{A}_x, 0) & \hat{a}_S &= -\max(\hat{F}_{x,s} \hat{A}_y, 0) \end{aligned} \quad (3.4.69)$$

Similarly, the left side of the y -component of the dimensionless Momentum equation is given in equation (3.4.70) with the coefficients in equations (3.4.71)-(3.4.72).

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_W \hat{v}_W + \hat{a}_N \hat{v}_N + \hat{a}_S \hat{v}_S = \mathbf{RHS} \quad (3.4.70)$$

with

$$\hat{a}_P = -\hat{a}_W - \hat{a}_E - \hat{a}_N - \hat{a}_S + \hat{F}_{y,e} \hat{A}_x - \hat{F}_{y,w} \hat{A}_x + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \quad (3.4.71)$$

$$\begin{aligned} \hat{a}_E &= -\max(0, -\hat{F}_{y,e} \hat{A}_x) & \hat{a}_N &= -\max(0, -\hat{F}_{y,n} \hat{A}_y) \\ \hat{a}_W &= -\max(\hat{F}_{y,w} \hat{A}_x, 0) & \hat{a}_S &= -\max(\hat{F}_{y,s} \hat{A}_y, 0) \end{aligned} \quad (3.4.72)$$

3.4.4.2 Right Hand Side

The difference in the form of the right side of the dimensionless Momentum equation and the right side of the ordinary Momentum equation is the presence of the factor $\frac{1}{Re}$ in front of the diffusive terms as seen in equation (3.4.64). The discretisation steps for equation (3.4.64) precisely follow the steps in section 3.2, except for the equation being integrated over the dimensionless control volume instead of the regular control volume.

The right hand side of equation (3.4.64) can be written as equation (3.4.73).

$$\mathbf{LHS} = -\hat{\nabla} \hat{p} - \frac{1}{Re} \hat{\nabla} \cdot \hat{\sigma} \quad (3.4.73)$$

The x - and y - components of equation (3.4.73) are obtained by taking the dot product with the unit vectors \mathbf{e}_x and \mathbf{e}_y respectively. The components of the stress tensors as given in appendix A can then be inserted to obtain equations (3.4.74) and (3.4.75) for x - and y respectively.

$$\mathbf{LHS} = -\frac{\partial \hat{p}}{\partial \hat{x}} + \frac{1}{Re} \left(\frac{\partial}{\partial \hat{x}} \left(\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{x}} \right) + \frac{\partial}{\partial \hat{y}} \left(\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{y}} \right) \right) \quad (3.4.74)$$

$$\mathbf{LHS} = -\frac{\partial \hat{p}}{\partial \hat{y}} + \frac{1}{Re} \left(\frac{\partial}{\partial \hat{x}} \left(\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{x}} \right) + \frac{\partial}{\partial \hat{y}} \left(\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{y}} \right) \right) \quad (3.4.75)$$

Equations (3.4.74) and (3.4.75) can then be integrated over the dimensionless control volume $\hat{C}\hat{V}$. For the diffusive terms, the volume integral is split, taking $d\hat{V} = d\hat{A}_x d\hat{x}$ and $d\hat{V} = d\hat{A}_y d\hat{y}$ as in equations (3.4.76) and (3.4.77).

$$\begin{aligned} \mathbf{LHS} &= -\frac{\partial \hat{p}}{\partial \hat{x}} \hat{V}_{CV} + \frac{1}{Re} \int_{\delta \hat{x}} \int_{\hat{A}_x} \frac{\partial}{\partial \hat{x}} \left(\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{x}} \right) d\hat{A}_x d\hat{x} \\ &\quad + \frac{1}{Re} \int_{\delta \hat{y}} \int_{\hat{A}_y} \frac{\partial}{\partial \hat{y}} \left(\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{y}} \right) d\hat{A}_y d\hat{y} \end{aligned} \quad (3.4.76)$$

$$\begin{aligned} \mathbf{LHS} = & -\frac{\partial \hat{p}}{\partial \hat{y}} \hat{V}_{CV} + \frac{1}{Re} \int_{\delta \hat{x}} \int_{\hat{A}_x} \frac{\partial}{\partial \hat{x}} \left(\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{x}} \right) d\hat{A}_x d\hat{x} \\ & + \frac{1}{Re} \int_{\delta \hat{y}} \int_{\hat{A}_y} \frac{\partial}{\partial \hat{y}} \left(\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{y}} \right) d\hat{A}_y d\hat{y} \end{aligned} \quad (3.4.77)$$

Equations (3.4.76) and (3.4.77) take the same form as equations (3.2.38) and (3.2.39), and the rest of the discretisation of the right hand side of the Momentum equation follows the same steps as in section 3.2.

The dimensionless diffusion conductance is defined as in equation (3.4.78).

$$\hat{D}_x = \frac{1}{Re} \frac{\hat{\mu}}{\delta \hat{x}} \quad \hat{D}_y = \frac{1}{Re} \frac{\hat{\mu}}{\delta \hat{y}} \quad (3.4.78)$$

The discretised right hand side of the dimensionless Momentum equation for x - and y are given in equations (3.4.79) and (3.4.80)

$$\begin{aligned} \mathbf{LHS} = & -(\hat{p}_{I,J} - \hat{p}_{I-1,J})\hat{A}_x + \hat{D}_x \hat{A}_x (\hat{u}_{i+1,J} - \hat{u}_{i,J}) - \hat{D}_x \hat{A}_x (\hat{u}_{i,J} - \hat{u}_{i-1,J}) \\ & + \hat{D}_y \hat{A}_y (\hat{u}_{i,J+1} - \hat{u}_{i,J}) - \hat{D}_y \hat{A}_y (\hat{u}_{i,J} - \hat{u}_{i,J-1}) \end{aligned} \quad (3.4.79)$$

$$\begin{aligned} \mathbf{LHS} = & -(\hat{p}_{I,J} - \hat{p}_{I,J-1})\hat{A}_y + \hat{D}_x \hat{A}_x (\hat{v}_{I+1,j} - \hat{v}_{I,j}) - \hat{D}_x \hat{A}_x (\hat{v}_{I,j} - \hat{v}_{I-1,j}) \\ & + \hat{D}_y \hat{A}_y (\hat{v}_{I,j+1} - \hat{v}_{I,j}) - \hat{D}_y \hat{A}_y (\hat{v}_{I,j} - \hat{v}_{I,j-1}) \end{aligned} \quad (3.4.80)$$

3.4.4.3 Combined Momentum Equation

Combining both sides of the x -component momentum equation yields equation (3.4.81) with the coefficients in equation (3.4.82). Note that the equation is of the same form as equation (3.2.57).

$$\hat{a}_{i,J} \hat{u}_{i,J} + \hat{a}_{i+1,J} \hat{u}_{i+1,J} + \hat{a}_{i-1,J} \hat{u}_{i-1,J} + \hat{a}_{i,J+1} \hat{u}_{i,J+1} + \hat{a}_{i,J-1} \hat{u}_{i,J-1} = \hat{b}_{i,J} \quad (3.4.81)$$

with

$$\begin{aligned} \hat{a}_{i,J} &= -\hat{a}_{i+1,J} - \hat{a}_{i-1,J} - \hat{a}_{i,J+1} - \hat{a}_{i,J-1} + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_y + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ \hat{a}_{i+1,J} &= -\max(0, -\hat{F}_{x,e} \hat{A}_x) - \hat{D}_x \hat{A}_x \\ \hat{a}_{i-1,J} &= -\max(\hat{F}_{x,w} \hat{A}_y, 0) - \hat{D}_x \hat{A}_x \\ \hat{a}_{i,J+1} &= -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \\ \hat{a}_{i,J-1} &= -\max(\hat{F}_{y,s} \hat{A}_y, 0) - \hat{D}_y \hat{A}_y \\ \hat{b}_{i,J} &= -\left(\hat{p}_{I,J} - \hat{p}_{I-1,J} \right) \hat{A}_x \end{aligned} \quad (3.4.82)$$

Similarly, combining both sides of the y -component momentum equation yields equation (3.4.83) with the coefficients in equation (3.4.84). Note that the equation is of the same form as equation (3.2.59).

$$\hat{a}_{I,j} \hat{v}_{I,j} + \hat{a}_{I+1,j} \hat{v}_{I+1,j} + \hat{a}_{I-1,j} \hat{v}_{I-1,j} + \hat{a}_{I,j+1} \hat{v}_{I,j+1} + \hat{a}_{I,j-1} \hat{v}_{I,j-1} = \hat{b}_{I,j} \quad (3.4.83)$$

with

$$\begin{aligned}
\hat{a}_{I,j} &= -\hat{a}_{I+1,j} - \hat{a}_{I-1,j} - \hat{a}_{I,j+1} - \hat{a}_{I,j-1} + \hat{F}_{x,e}\hat{A}_x - \hat{F}_{x,w}\hat{A}_y + \hat{F}_{y,n}\hat{A}_y - \hat{F}_{y,s}\hat{A}_y \\
\hat{a}_{I+1,j} &= -\max(0, -\hat{F}_{x,e}\hat{A}_x) - \hat{D}_x\hat{A}_x \\
\hat{a}_{I-1,j} &= -\max(\hat{F}_{x,w}\hat{A}_y, 0) - \hat{D}_x\hat{A}_y \\
\hat{a}_{I,j+1} &= -\max(0, -\hat{F}_{y,n}\hat{A}_y) - \hat{D}_y\hat{A}_y \\
\hat{a}_{I,j-1} &= -\max(\hat{F}_{y,s}\hat{A}_y, 0) - \hat{D}_y\hat{A}_y \\
\hat{b}_{I,j} &= -\left(\hat{p}'_{I,J} - \hat{p}'_{I,J-1}\right)\hat{A}_y
\end{aligned} \tag{3.4.84}$$

3.4.5 Dimensionless SIMPLE-Equations

The discretised dimensionless Continuity equation (3.4.56) takes the same form as the regular discretised Continuity equation in (3.1.6) and the discretised Momentum equation for the x - and y -component in equations (3.4.81) and (3.4.83) take the same form as the ordinary Momentum equation for the x - and y -component in equations (3.2.57) and (3.2.59). The dimensionless velocity and pressure correction equations will therefore take the same forms as the ordinary velocity equation (3.3.2) and pressure correction equation (3.3.8) which is explained in section 3.3.

The dimensionless velocity correction equation is obtained by taking the dimensionless Momentum equation and subtracting the dimensionless Momentum equation for the dimensionless guessed properties. The velocity corrections of the neighbouring nodes are omitted. The result is equation (3.4.85) for the u -velocity component $u_{i,J}$ and equation (3.4.86) for the v -velocity component $v_{I,j}$.

$$\hat{u}_{i,J} = \hat{u}_{i,J}^* - \frac{\hat{A}_x}{\hat{a}_{i,J}^{centre}} \left(\hat{p}'_{I,J} - \hat{p}'_{I-1,J} \right) \tag{3.4.85}$$

$$\hat{v}_{I,j} = \hat{v}_{I,j}^* - \frac{\hat{A}_y}{\hat{a}_{I,j}^{centre}} \left(\hat{p}'_{I,J} - \hat{p}'_{I,J-1} \right) \tag{3.4.86}$$

The dimensionless pressure correction equation is obtained from the dimensionless discretised Continuity equation (3.4.56) and the dimensionless velocity correction equations (3.4.85) and (3.4.86). The pressure correction is obtained for the adjusted pressure \hat{p} following equation (3.4.87).

$$\hat{p}' = \hat{p} - \hat{p}^* \tag{3.4.87}$$

The dimensionless velocity correction equations (3.4.85) and (3.4.86) are inserted into the dimensionless continuity equation (3.4.56). The equation is rearranged to collect all the pressure correction terms on one side of the equation. This yields the dimensionless pressure correction equation for the adjusted pressure in equation (3.4.88) with the coefficients in equation (3.4.89).

$$\hat{v}_{I,J}\hat{p}'_{I,J} + \hat{v}_{I+1,J}\hat{p}'_{I+1,J} + \hat{v}_{I-1,J}\hat{p}'_{I-1,J} + \hat{v}_{I,J+1}\hat{p}'_{I,J+1} + \hat{v}_{I,J-1}\hat{p}'_{I,J-1} = \hat{\beta}_{I,J} \tag{3.4.88}$$

with

$$\begin{aligned}
\hat{\nu}_{I,J} &= \hat{\rho} \frac{\hat{A}_x^2}{\hat{a}_{i+1,J}^{centre}} + \hat{\rho} \frac{\hat{A}_x^2}{\hat{a}_{i,J}^{centre}} + \hat{\rho} \frac{\hat{A}_y^2}{\hat{a}_{I,j+1}^{centre}} + \hat{\rho} \frac{\hat{A}_y^2}{\hat{a}_{I,j}^{centre}} \\
\hat{\nu}_{I+1,J} &= - \hat{\rho} \frac{\hat{A}_x^2}{\hat{a}_{i+1,J}^{centre}} \\
\hat{\nu}_{I-1,J} &= - \hat{\rho} \frac{\hat{A}_x^2}{\hat{a}_{i,J}^{centre}} \\
\hat{\nu}_{I,J+1} &= - \hat{\rho} \frac{\hat{A}_y^2}{\hat{a}_{I,j+1}^{centre}} \\
\hat{\nu}_{I,J-1} &= - \hat{\rho} \frac{\hat{A}_y^2}{\hat{a}_{I,j}^{centre}} \\
\hat{\beta}_{I,J} &= - \hat{A}_x \hat{\rho} \hat{u}_{x,e}^* + \hat{A}_x \hat{\rho} \hat{u}_{x,w}^* - \hat{A}_y \hat{\rho} \hat{u}_{y,n}^* + \hat{A}_y \hat{\rho} \hat{u}_{y,s}^*
\end{aligned} \tag{3.4.89}$$

4

Implementation

In this chapter, the properties of the flow are given, as well as the inlet and outlet properties, the boundary conditions and the implementation of these into the discretised equations and the coding in MATLAB.

4.1 Properties of the Flow and the Domain

In this chapter, the fluid flow to be modelled is described, and the properties of the flow are given.

4.1.1 Fluid Properties

The modelled fluid is water and the fluid properties will be taken to be constant with the values given in equation (4.1.1)[31]. Gravity is assumed to be effective in z -direction and is therefore not modelled in the two-dimensional domains.

$$\rho = 997 \left[\text{kg/m}^3 \right] \text{ at } 25^\circ\text{C} \quad \mu = 8.90 \cdot 10^{-4} \left[\text{Pa} \cdot \text{s} \right] \quad (4.1.1)$$

4.1.2 Domain Size

Schematic representations of the domains used are given in chapter 1. Figure 1.1 shows the straight channel domains and figures 1.2 and 1.3 show the backwards facing step (BFS) domain with two different expansion ratios. The expansion ratio of the BFS-domains is given in equation (4.1.2).

$$\text{Expansion ratio} = \frac{H}{h} \quad (4.1.2)$$

where h is the height of the channel at the inlet and H is the height of the channel after the expansion, the total height of the channel. Table 4.1 shows the sizes of the different domains. The unit for all length scales is meter. The domain BFS 1 is used to develop the model, and the domain BFS 2 is used to compare the results to excising

Domain	Total length	Total height	Step length	Step height	Expansion ratio
Short channel	3	1	-	-	-
Long channel	22	1	-	-	-
BFS 1	22	1.5	3	0.5	1.5
BFS 2	35	2	5	1	2

Table 4.1: Dimensions of the different domains used for the simulations.

literature as given in Biswas et al. [4]. The dimensions for the first domain used by Melaaen [3] were taken as example dimensions for use when developing the backwards facing step model, and the fluid flow parameters are not matched with what was used by Melaaen [3]. For the second domain as used by Biswas et al. [4], the Reynolds number was matched to what is given in the article. There are still some differences in the implementation of the simulations between this thesis and the article by Biswas et al. [4], which are discussed in chapter 6. The expansion ratio used is actually 1.9423, but was rounded off to 2 for simplicity.

4.2 Model Settings

In this section all necessary model settings and parameters are stated. The implementation of the boundary conditions is given in section 4.4.

4.2.1 Straight channel

Table 4.2 shows the parameters and model settings for the two dimensional straight channel that are the same for all variations of the Reynolds number. v_{in} is the inlet v -velocity, p_{out} is the outlet pressure, α are under-relaxation factors, N is the number of scalar computational nodes in x -direction, M is the number of scalar computational nodes in y -direction and Total is the total number of scalar computational nodes.

Parameter	Value	Unit
v_{in}	0	m/s
p_{out}	$1.01325 \cdot 10^5$	Pa
α_u	0.01	-
α_v	0.01	-
α_p	0.02	-
N	88	-
M	18	-
Total	1584	-

Table 4.2: Parameters and model settings for the two dimensional model

Table 4.3 shows the different Reynolds numbers used in the simulations and the corresponding inlet u -velocity u_{in} . The Reynolds number Re is calculated by equation (2.1.8) with the hydraulic diameter as defined in equation (2.1.9).

	$Re = 1120$	$Re = 560$
u_{in}	$1 \cdot 10^{-3}$ m/s	$5 \cdot 10^{-4}$ m/s

Table 4.3: Varying parameter for the two dimensional straight channel domain with different Reynolds numbers.

4.2.2 Backwards Facing Step

4.2.2.1 Domain One

Domain one is shown in the schematic in figure 1.2 and the dimensions are described in table 4.1 in the row labelled BFS 1. The model for this domain has a constant inlet velocity. In the thesis by Melaaen [3], a parabolic inlet profile was used, but since this domain is used to develop the backwards facing step model without matching the fluid parameters, a constant inlet velocity is used.

Table 4.4 shows the parameters and model settings for the first two dimensional backwards facing step domain that are the same for all simulations using this domain. v_{in} is the inlet v -velocity and p_{out} is the outlet pressure. N_{narrow} is the number of scalar computational nodes in x -direction in the narrow inlet section and N_{total} is the total number of scalar computational nodes in x -direction. M_{narrow} is the number of scalar computational nodes in y -direction in the narrow inlet section and M_{total} is the total number of scalar computational nodes in y -direction. Total is the total number of scalar computational nodes.

Parameter	Value	Unit
v_{in}	0	m/s
p_{out}	$1.01325 \cdot 10^5$	Pa
N_{narrow}	12	-
N_{total}	88	-
M_{narrow}	12	-
M_{total}	18	-
Total	1512	

Table 4.4: Parameters and model settings for the two dimensional model

Table 4.5 shows the different Reynolds numbers for the different simulations along with the corresponding parameters and model settings for the first two dimensional backwards facing step domain. The Reynolds number is calculated by equation (2.1.8) with the hydraulic diameter as defined in equation (2.1.9). α are under-relaxation factors.

	$Re = 1120$	$Re = 560$
u_{in}	$1 \cdot 10^{-3}$ m/s	$5 \cdot 10^{-4}$ m/s
α_u	0.01	0.005
α_v	0.01	0.005
α_p	0.02	0.010

Table 4.5: Varying parameters for the first backwards facing step domain with different Reynolds numbers.

4.2.2.2 Domain Two

Domain two is shown in the schematic in figure 1.3 and the dimensions are described in table 4.1 in the row labelled BFS 2. The model for this domain has a parabolic inlet velocity profile as given in equation (2.1.6)[16].

Table 4.6 shows the parameters and model settings for the second two dimensional backwards facing step domain that are the same for all simulations using this domain. v_{in} is the inlet v -velocity and p_{out} is the outlet pressure. N_{narrow} is the number of scalar computational nodes in x -direction in the narrow inlet section and N_{total} is the total number of scalar computational nodes in x -direction. M_{narrow} is the number of scalar computational nodes in y -direction in the narrow inlet section and M_{total} is the total number of scalar computational nodes in y -direction. Total is the total number of scalar computational nodes.

Parameter	Value	Unit
v_{in}	0	m/s
p_{out}	$1.01325 \cdot 10^5$	Pa
N_{narrow}	10	-
N_{total}	70	-
M_{narrow}	10	-
M_{total}	20	-
Total	1512	

Table 4.6: Parameters and model settings for the two dimensional model

Table 4.7 shows the different Reynolds numbers for the different simulations along with the corresponding parameters and model settings for the second backwards facing step domain. The Reynolds number is calculated by equation (2.1.8) with the hydraulic diameter D_{hyd} equal to $2h$ as defined by Biswas et al. [4]. α are under-relaxation factors.

Re	u_{avg}	u_{max}	α_u	α_v	α_p
0.0001	$4.46 \cdot 10^{-11}$	$8.92 \cdot 10^{-11}$	0.01	0.01	0.02
0.1	$4.46 \cdot 10^{-8}$	$8.92 \cdot 10^{-8}$	0.01	0.01	0.02
1	$4.46 \cdot 10^{-7}$	$8.92 \cdot 10^{-7}$	0.01	0.01	0.02
10	$4.46 \cdot 10^{-6}$	$8.92 \cdot 10^{-6}$	0.01	0.01	0.02
50	$2.23 \cdot 10^{-5}$	$4.46 \cdot 10^{-5}$	0.01	0.01	0.02
100	$4.46 \cdot 10^{-5}$	$8.92 \cdot 10^{-5}$	0.01	0.01	0.02
200	$8.93 \cdot 10^{-5}$	$1.79 \cdot 10^{-4}$	0.005	0.005	0.01
400	$1.79 \cdot 10^{-4}$	$3.57 \cdot 10^{-4}$	0.005	0.005	0.01

Table 4.7: Varying parameter for the second backwards facing step domain with different Reynolds numbers.

4.3 Initial Guesses

All the models start out with an initial guess for the velocity and pressure to be calculated from. The initial guesses for the different models are given in this section.

4.3.1 Straight Channel

The initial guesses for both velocity components and the adjusted pressure were taken as constants across the whole domain with the values as given in equations (4.3.1)-(4.3.3). The guesses are defined after the definition of the dimensionless variables, and the guess is therefore dimensionless.

$$\hat{u}_{guess} = \hat{u}_{in} = 1 \quad (4.3.1)$$

$$\hat{v}_{guess} = \hat{v}_{in} = 0 \quad (4.3.2)$$

$$\hat{p}_{guess} = \hat{p}_{out} = 0 \quad (4.3.3)$$

4.3.2 Backwards Facing Step

The same expressions are used for the initial guesses for both backwards facing step domains. The initial guesses for the velocity components were taken as two different constant values for the narrow section and wide section of the domain. The velocity guesses for the narrow section are given by equations (4.3.4) and (4.3.5) for the constant inlet velocity case.

$$\hat{u}_{guess}^{narrow} = \hat{u}_{in} = 1 \quad (4.3.4)$$

$$\hat{v}_{guess}^{narrow} = \hat{v}_{in} = 0 \quad (4.3.5)$$

For the parabolic inlet velocity case, the velocity guesses for the narrow section are given by equations (4.3.6) and (4.3.7).

$$\hat{u}_{guess}^{narrow} = \hat{u}_{max} \quad (4.3.6)$$

$$\hat{v}_{guess}^{narrow} = \hat{v}_{in} = 0 \quad (4.3.7)$$

The velocity guesses for the wide section should be lower than for the narrow section since the cross section of the channel increases after the expansion. The number of computational points for the velocities in y -direction is used for this as shown in equations (4.3.8) and (4.3.9). The decrease in guessed value from the narrow to the wide section is then varying with the expansion ratios for the BFS domains as given in table 4.1.

$$\hat{u}_{guess}^{wide} = \hat{u}_{guess}^{narrow} \frac{M_{narrow}}{M_{total}} \quad (4.3.8)$$

$$\hat{v}_{guess}^{wide} = \hat{v}_{guess}^{narrow} \frac{m_{narrow}}{m_{total}} \quad (4.3.9)$$

M_{narrow} is the number of u -velocity nodes in y -direction in the narrow section and M_{total} is the number of u -velocity nodes in y -direction in total and in the wide section. m_{narrow} is the number of v -velocity nodes in y -direction in the narrow section and m_{total} is the number of v -velocity nodes in y -direction in total and in the wide section.

The guess for the adjusted pressure is taken as constant across the whole domain as given in (4.3.10).

$$\hat{p}_{guess} = \hat{p}_{out} = 0 \quad (4.3.10)$$

4.4 Boundary Conditions

The no-slip and no-penetrate conditions are applied at the walls of the channel, which means that both the u - and the v -velocities are zero at all walls [16].

The momentum equations include two dimensional derivatives in both x - and y -direction, which means that the momentum equations for the u - and v -velocity each need two boundary conditions and two inlet/outlet conditions. The velocity at the southern and northern walls are set to be equal to zero for both the u - and v -velocity. The inlet u - and v -velocities are both known and are specified in section 4.2 for the different simulation cases. This only leaves the outlet boundary.

The pressure is two dimensional in each direction x and y , which means that two boundary conditions in each dimension are required. The boundary at the inlet as well as the southern and northern walls are already determined by the boundary conditions of the velocities, and the pressure does not need to be specified. The known outlet pressure is therefore a sufficient boundary condition for the pressure, which also provides the last needed boundary condition for the velocities.

Below follows the implementation of the boundary conditions mentioned above for the two dimensional straight channel. The additional boundaries and the boundary conditions needed for the backwards facing step model are described in section 4.5.1. The discretised momentum equation and pressure correction equations are stated for each of the different boundaries of the domain. The velocities and pressures in the discretised equations are noted with a letter subscript of the form u_P instead of the indexed version $u_{i,j}$ for simplicity. The equations are given in the dimensionless form. The velocities in the Momentum equation are given with the notations \hat{u} and \hat{v} in this section, but correspond to \hat{u}^* and \hat{v}^* in figure 2.8. The superscript $*$ to note these intermediate velocities are omitted in this section. The velocities \hat{u} and \hat{v} that occur in the source term in the pressure correction equation in this chapter are the velocities obtained from the Momentum equations.

Where the expressions for the convective mass flux F need to be altered, only the changed expression is given. The velocity correction can be directly obtained everywhere except at the outlet where a special implementation must be used.

4.4.1 Inlet

At the inlet, the velocities at the west node are known and are noted \hat{u}_{in} for the \hat{u} -velocity and \hat{v}_{in} for the \hat{v} -velocity. \hat{v}_{in} is equal to zero for all the simulation models is therefore omitted from the below discretised equations. In the case of the parabolic inlet velocity profile where \hat{u}_{in} is not a constant number, an index for the current row of the domain must be added to obtain the correct value.

4.4.1.1 Convective Mass Flux

At the inlet the convective mass fluxes $\hat{F}_{x,w}$ and $\hat{F}_{y,w}$ become equations (4.4.1) and (4.4.2). Both the \hat{u} -velocity nodes taking part in $\hat{F}_{y,w}$ are located at the inlet.

$$\hat{F}_{x,w} = \hat{\rho} \frac{\hat{u}_{in} + \hat{u}_P}{2} \quad (4.4.1)$$

$$\hat{F}_{y,w} = \hat{\rho} \hat{u}_{in} \quad (4.4.2)$$

4.4.1.2 Momentum Equation for the x -component

The Momentum Equation for the x -component at the inlet becomes equation (4.4.3) with the coefficients in equations (4.4.4)-(4.4.8). The western velocity node is the

known \hat{u}_{in} and is therefore moved to the source term.

$$\hat{a}_P \hat{u}_P + \hat{a}_E \hat{u}_E + \hat{a}_N \hat{u}_N + \hat{a}_S \hat{u}_S = \hat{b}_P \quad (4.4.3)$$

with

$$\begin{aligned} \hat{a}_P = & -\hat{a}_E - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_y + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ & + \max(\hat{F}_{x,w} \hat{A}_x, 0) + \hat{D}_x \hat{A}_x \end{aligned} \quad (4.4.4)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e} \hat{A}_x) - \hat{D}_x \hat{A}_x \quad (4.4.5)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.4.6)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s} \hat{A}_y, 0) - \hat{D}_y \hat{A}_y \quad (4.4.7)$$

$$\hat{b}_P = -\left(\hat{p}_P - \hat{p}_W\right) \hat{A}_x + \left(\max(\hat{F}_{x,w} \hat{A}_y, 0) - \hat{D}_x \hat{A}_y\right) \hat{u}_{in} \quad (4.4.8)$$

4.4.1.3 Momentum Equation for the y -component

The Momentum Equation for the y -component at the inlet becomes equation (4.4.9) with the coefficients in equations (4.4.10)-(4.4.14). The western velocity node is the known $\hat{v}_{in} = 0$ which is omitted from the source term.

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_N \hat{v}_N + \hat{a}_S \hat{v}_S = \hat{b}_P \quad (4.4.9)$$

with

$$\begin{aligned} \hat{a}_P = & -\hat{a}_E - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_y + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ & + \max(\hat{F}_{x,w} \hat{A}_x, 0) + \hat{D}_x \hat{A}_x \end{aligned} \quad (4.4.10)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e} \hat{A}_x) - \hat{D}_x \hat{A}_x \quad (4.4.11)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.4.12)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s} \hat{A}_y, 0) - \hat{D}_y \hat{A}_y \quad (4.4.13)$$

$$\hat{b}_P = -\left(\hat{p}_P - \hat{p}_S\right) \hat{A}_y \quad (4.4.14)$$

4.4.1.4 Pressure Correction Equation

The western velocity node is \hat{u}_{in} which is known, and no pressure correction is needed. \hat{u}_{in} has therefore been directly inserted into the Continuity equation under the derivation of the pressure correction equation. No link is then created to the western boundary. The result is equation (4.4.15) with the coefficients in equations (4.4.16)-(4.4.20).

$$\hat{v}_P \hat{p}'_P + \hat{v}_E \hat{p}'_E + \hat{v}_N \hat{p}'_N + \hat{v}_S \hat{p}'_S = \hat{\beta}_P \quad (4.4.15)$$

with

$$\hat{v}_P = -\hat{v}_E - \hat{v}_N - \hat{v}_S \quad (4.4.16)$$

$$\hat{v}_E = -\frac{\hat{\rho}\hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \quad (4.4.17)$$

$$\hat{v}_N = -\frac{\hat{\rho}\hat{A}_y^2}{\hat{a}_{v,N}^{centre}} \quad (4.4.18)$$

$$\hat{v}_S = -\frac{\hat{\rho}\hat{A}_y^2}{\hat{a}_{v,P}^{centre}} \quad (4.4.19)$$

$$\hat{\beta}_P = -\hat{A}_x\hat{\rho}\hat{u}_e + \hat{A}_x\hat{\rho}\hat{u}_{in} - \hat{A}_y\hat{\rho}\hat{v}_n + \hat{A}_y\hat{\rho}\hat{v}_s \quad (4.4.20)$$

4.4.2 Outlet

At the outlet, the pressure at the eastern node is known and is noted \hat{p}_{out} .

4.4.2.1 Convective Mass Flux

At the outlet, the convective mass flux $\hat{F}_{x,e}$ is set equal to $\hat{F}_{x,w}$ as in equation (4.4.21)[2]. $\hat{F}_{y,e}$ does not need to be altered.

$$\hat{F}_{x,e} = \hat{F}_{x,w} = \hat{\rho}\frac{\hat{u}_W + \hat{u}_P}{2} \quad (4.4.21)$$

$$(4.4.22)$$

4.4.2.2 Momentum Equation for the x -component

The Momentum Equation for the x -component at the outlet becomes equation (4.4.23) with the coefficients in equations (4.4.24)-(4.4.28). The eastern velocity node \hat{u}_W is outside of the domain, and the connection to this node is broken by setting \hat{a}_E equal to zero [2].

$$\hat{a}_P\hat{u}_P + \hat{a}_W\hat{u}_W + \hat{a}_N\hat{u}_N + \hat{a}_S\hat{u}_S = \hat{b}_P \quad (4.4.23)$$

with

$$\hat{a}_P = -\hat{a}_W - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e}\hat{A}_x - \hat{F}_{x,w}\hat{A}_y + \hat{F}_{y,n}\hat{A}_y - \hat{F}_{y,s}\hat{A}_y \quad (4.4.24)$$

$$\hat{a}_W = -\max(0, -\hat{F}_{x,w}\hat{A}_x) - \hat{D}_x\hat{A}_x \quad (4.4.25)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n}\hat{A}_y) - \hat{D}_y\hat{A}_y \quad (4.4.26)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s}\hat{A}_y, 0) - \hat{D}_y\hat{A}_y \quad (4.4.27)$$

$$\hat{b}_P = -\left(\hat{p}_P - \hat{p}_W\right)\hat{A}_x \quad (4.4.28)$$

4.4.2.3 Momentum Equation for the y -component

The Momentum Equation for the y -component at the outlet becomes equation (4.4.29) with the coefficients in equations (4.4.30)-(4.4.34). The eastern velocity node \hat{u}_W is outside of the domain, and the connection to this node is broken by setting \hat{a}_E equal to zero [2].

$$\hat{a}_P \hat{v}_P + \hat{a}_W \hat{v}_W + \hat{a}_N \hat{v}_N + \hat{a}_S \hat{v}_S = \hat{b}_P \quad (4.4.29)$$

with

$$\hat{a}_P = -\hat{a}_W - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_x + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \quad (4.4.30)$$

$$\hat{a}_W = -\max(\hat{F}_{x,w} \hat{A}_x, 0) - \hat{D}_x \hat{A}_y \quad (4.4.31)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.4.32)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s} \hat{A}_y, 0) - \hat{D}_y \hat{A}_y \quad (4.4.33)$$

$$\hat{b}_P = -\left(\hat{p}_P - \hat{p}_S\right) \hat{A}_y \quad (4.4.34)$$

4.4.2.4 Pressure Correction Equation

At the outlet, the eastern pressure node is known, and the pressure correction is zero for the known pressure. The pressure correction can therefore be set to zero at the eastern node which yields equation (4.4.15) with the coefficients in equations (4.4.36)-(4.4.40).

$$\hat{v}_P \hat{p}'_P + \hat{v}_W \hat{p}'_W + \hat{v}_N \hat{p}'_N + \hat{v}_S \hat{p}'_S = \hat{\beta}_P \quad (4.4.35)$$

with

$$\hat{v}_P = \frac{\hat{\rho} \hat{A}_x^2}{\hat{a}_{u,E}^{centre}} - \hat{v}_W - \hat{v}_N - \hat{v}_S \quad (4.4.36)$$

$$\hat{v}_W = -\frac{\hat{\rho} \hat{A}_x^2}{\hat{a}_{u,P}^{centre}} \quad (4.4.37)$$

$$\hat{v}_N = -\frac{\hat{\rho} \hat{A}_y^2}{\hat{a}_{v,N}^{centre}} \quad (4.4.38)$$

$$\hat{v}_S = \frac{\hat{\rho} \hat{A}_y^2}{\hat{a}_{v,P}^{centre}} \quad (4.4.39)$$

$$\hat{\beta}_P = -\hat{A}_x \hat{\rho} \hat{u}_e + \hat{A}_x \hat{\rho} \hat{u}_w - \hat{A}_y \hat{\rho} \hat{v}_n + \hat{A}_y \hat{\rho} \hat{v}_s \quad (4.4.40)$$

4.4.2.5 Velocity Correction Equation

Since the pressure correction at the eastern node at the outlet is zero, the eastern node vanishes from the \hat{u} -velocity correction equation, yielding equation (4.4.41).

$$\hat{u}_P = \hat{u}_P^* - \frac{\hat{A}_x}{\hat{a}_P^{centre}} \left(-\hat{p}'_W\right) \quad (4.4.41)$$

The \hat{v} -velocity correction equation does not need to be altered.

4.4.3 Walls

As described at the beginning of this section, all wall velocities are zero and the no-slip and no-penetrate conditions are used. The \hat{v} -velocity nodes coincide with the wall at both the northern and southern boundary of the domain. Due to the staggered grid, the \hat{u} -velocity nodes are placed so that the faces of the control volumes around the nodes line up with the walls, while the nodes themselves are located at a distance $\delta\hat{y}/2$ from the wall. $\delta\hat{y}$ is the height of the dimensionless control volumes.

4.4.3.1 Convective Mass Flux

Both velocities are zero at the walls. The convective mass fluxes become equations (4.4.42)-(4.4.43) for the northern wall and equations (4.4.44)-(4.4.45) for the southern wall.

$$\hat{F}_{x,n} = 0 \quad (4.4.42)$$

$$\hat{F}_{y,n} = \hat{\rho}\hat{v}_P \quad (4.4.43)$$

$$\hat{F}_{x,s} = 0 \quad (4.4.44)$$

$$\hat{F}_{y,s} = \frac{\hat{\rho}}{2}\hat{v}_P \quad (4.4.45)$$

4.4.3.2 Momentum Equation for the x -component

For implementation of the wall boundary condition, the discretised right hand side of the Momentum Equation for the x -component right after the integration over the control volume is taken as given in equation 4.4.46. The left hand side of the equation may be kept as before.

$$\begin{aligned} \mathbf{LHS} = & -\frac{\partial\hat{p}}{\partial\hat{x}}\Big|_P \delta\hat{x}\hat{A}_x + \frac{1}{Re}\hat{\mu}\frac{\partial\hat{u}}{\partial\hat{x}}\Big|_e \hat{A}_{x,e} - \frac{1}{Re}\hat{\mu}\frac{\partial\hat{u}}{\partial\hat{x}}\Big|_w \hat{A}_{x,w} \\ & + \frac{1}{Re}\hat{\mu}\frac{\partial\hat{u}}{\partial\hat{y}}\Big|_n \hat{A}_{y,n} - \frac{1}{Re}\hat{\mu}\frac{\partial\hat{u}}{\partial\hat{y}}\Big|_s \hat{A}_{y,s} \end{aligned} \quad (4.4.46)$$

First taking the north boundary into account, the gradient over the north face of the control volume is defined as equation (4.4.47) by use of a central difference.

$$\frac{\partial\hat{u}}{\partial\hat{y}}\Big|_n = \frac{\hat{u}_{wall} - \hat{u}_P}{\delta\hat{y}/2} \quad (4.4.47)$$

The distance from the centre node \hat{u}_P to the wall is $\delta\hat{y}/2$. This incorporates a shear force into the source term of the momentum equation which slows down the flow close to the wall. The wall shear stress is defined by equation (4.4.48), and the shear force can be defined as in equation (4.4.49)[16].

$$\hat{u}_{wall} = -\frac{1}{Re}\hat{\mu}\frac{\hat{u}_P}{\delta\hat{y}/2} \quad (4.4.48)$$

$$\hat{F}_s = -\frac{1}{Re}\hat{\mu}\hat{A}_y\frac{\hat{u}_P}{\delta\hat{y}/2} \quad (4.4.49)$$

The approximated gradient in equation (4.4.47) along with the approximations for the remaining gradients are inserted back into the right hand side of the Momentum equation for the x -component which yields equation (4.4.50).

$$\begin{aligned} \mathbf{LHS} = & \frac{1}{Re} \hat{\mu} \frac{\hat{u}_E - \hat{u}_P}{\delta \hat{x}} \hat{A}_{x,e} - \frac{1}{Re} \hat{\mu} \frac{\hat{u}_P - \hat{u}_W}{\delta \hat{x}} \hat{A}_{x,w} \\ & + 2 \frac{1}{Re} \hat{\mu} \frac{\hat{u}_{wall} - \hat{u}_P}{\delta \hat{y}} \hat{A}_{y,n} - \frac{1}{Re} \hat{\mu} \frac{\hat{u}_P - \hat{u}_S}{\delta \hat{y}} \hat{A}_{y,s} - (\hat{p}_P - \hat{p}_W) \hat{A}_x \end{aligned} \quad (4.4.50)$$

Further rearranging of equation (4.4.50) and combination with the left hand side yields the discretised Momentum Equation for the x -component (4.4.51) at the northern wall with the coefficients as given in equations (4.4.52)-(4.4.56).

$$\hat{a}_P \hat{u}_P + \hat{a}_E \hat{u}_E + \hat{a}_W \hat{u}_W + \hat{a}_S \hat{u}_N = \hat{b}_P \quad (4.4.51)$$

with

$$\begin{aligned} \hat{a}_P = & -\hat{a}_E - \hat{a}_W - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_y - \hat{F}_{x,w} \hat{A}_y + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ & + \max(0, -\hat{F}_{y,n} \hat{A}_y) + 2\hat{D}_y \hat{A}_y \end{aligned} \quad (4.4.52)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e} \hat{A}_y) - \hat{D}_x \hat{A}_y \quad (4.4.53)$$

$$\hat{a}_W = -\max(\hat{F}_{x,w} \hat{A}_y, 0) - \hat{D}_x \hat{A}_y \quad (4.4.54)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s} \hat{A}_y, 0) - \hat{D}_y \hat{A}_y \quad (4.4.55)$$

$$\hat{b}_P = -(\hat{p}_P - \hat{p}_W) \hat{A}_x \quad (4.4.56)$$

The implementation follows the same steps for the southern wall, where central differencing is used to approximate the gradient of the velocity over the southern cell face as given in equation (4.4.57).

$$\left. \frac{\partial \hat{u}}{\partial \hat{y}} \right|_s = \frac{\hat{u}_P - \hat{u}_{wall}}{\delta \hat{y}/2} \quad (4.4.57)$$

This yields the discretised Momentum Equation for the x -component (4.4.58) at the southern wall with the coefficients as given in equations (4.4.59)-(4.4.63).

$$\hat{a}_P \hat{u}_P + \hat{a}_E \hat{u}_E + \hat{a}_W \hat{u}_W + \hat{a}_N \hat{u}_N + \hat{a}_S \hat{u}_N = \hat{b}_P \quad (4.4.58)$$

with

$$\begin{aligned} \hat{a}_P = & -\hat{a}_E - \hat{a}_W - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_y - \hat{F}_{x,w} \hat{A}_y + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ & + \max(\hat{F}_{y,s} \hat{A}_y, 0) + 2\hat{D}_y \hat{A}_y \end{aligned} \quad (4.4.59)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e} \hat{A}_y) - \hat{D}_x \hat{A}_y \quad (4.4.60)$$

$$\hat{a}_W = -\max(\hat{F}_{x,w} \hat{A}_y, 0) - \hat{D}_x \hat{A}_y \quad (4.4.61)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.4.62)$$

$$\hat{b}_P = -(\hat{p}_P - \hat{p}_W) \hat{A}_x \quad (4.4.63)$$

4.4.3.3 Momentum Equation for the y -component

Since the \hat{v} -velocity nodes line up with the wall, the northern or southern \hat{v} -velocity nodes can be set to zero directly. This yields equation (4.4.64) at the north wall with the coefficients in equations (4.4.65)-(4.4.69).

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_W \hat{v}_W + \hat{a}_S \hat{v}_S = \hat{b}_P \quad (4.4.64)$$

with

$$\begin{aligned} \hat{a}_P = & -\hat{a}_E - \hat{a}_W - \hat{a}_S + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_x + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ & + \max(\hat{F}_{y,n} \hat{A}_y, 0) + \hat{D}_y \hat{A}_y \end{aligned} \quad (4.4.65)$$

$$\hat{a}_E = -\max(\hat{F}_{x,e} \hat{A}_x, 0) - \hat{D}_x \hat{A}_y \quad (4.4.66)$$

$$\hat{a}_W = -\max(\hat{F}_{x,w} \hat{A}_x, 0) - \hat{D}_x \hat{A}_y \quad (4.4.67)$$

$$\hat{a}_S = -\max(0, -\hat{F}_{y,s} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.4.68)$$

$$\hat{b}_P = -(\hat{p}_P - \hat{p}_S) \hat{A}_y \quad (4.4.69)$$

Equation (4.4.70) with the coefficients in equations (4.4.71)-(4.4.75) is the corresponding equation for the south wall boundary.

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_W \hat{v}_W + \hat{a}_N \hat{v}_N = \hat{b}_P \quad (4.4.70)$$

with

$$\begin{aligned} \hat{a}_P = & -\hat{a}_E - \hat{a}_W - \hat{a}_N + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_x + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ & + \max(\hat{F}_{y,s} \hat{A}_y, 0) + \hat{D}_y \hat{A}_y \end{aligned} \quad (4.4.71)$$

$$\hat{a}_E = -\max(\hat{F}_{x,e} \hat{A}_x, 0) - \hat{D}_x \hat{A}_y \quad (4.4.72)$$

$$\hat{a}_W = -\max(\hat{F}_{x,w} \hat{A}_x, 0) - \hat{D}_x \hat{A}_y \quad (4.4.73)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.4.74)$$

$$\hat{b}_P = -(\hat{p}_P - \hat{p}_S) \hat{A}_y \quad (4.4.75)$$

4.4.3.4 Pressure Correction Equation

Since the velocities are known at the walls, no pressure correction is needed for these points. The direct value of the velocities at the walls, which is zero can therefore be directly inserted into the Continuity equation under the derivation of the pressure correction equation. This creates no link to the northern or southern boundary which is the wall.

Equation 4.4.76 with the coefficients in equations (4.4.77)-(4.4.81) is the pressure correction equation for the northern wall boundary.

$$\hat{v}_P \hat{p}'_P + \hat{v}_E \hat{p}'_E + \hat{v}_W \hat{p}'_W + \hat{v}_S \hat{p}'_S = \hat{\beta}_P \quad (4.4.76)$$

with

$$\hat{v}_P = -\hat{v}_E - \hat{v}_W - \hat{v}_S \quad (4.4.77)$$

$$\hat{v}_E = -\frac{\hat{\rho} \hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \quad (4.4.78)$$

$$\hat{v}_W = -\frac{\hat{\rho} \hat{A}_x^2}{\hat{a}_{u,P}^{centre}} \quad (4.4.79)$$

$$\hat{v}_S = -\frac{\hat{\rho} \hat{A}_y^2}{\hat{a}_{v,P}^{centre}} \quad (4.4.80)$$

$$\hat{\beta}_P = -\hat{A}_x \hat{\rho} \hat{u}_e + \hat{A}_x \hat{\rho} \hat{u}_w + \hat{A}_y \hat{\rho} \hat{v}_s \quad (4.4.81)$$

Equation 4.4.82 with the coefficients in equations (4.4.83)-(4.4.87) is the pressure correction equation for the southern wall boundary.

$$\hat{v}_P \hat{p}'_P + \hat{v}_E \hat{p}'_E + \hat{v}_W \hat{p}'_W + \hat{v}_N \hat{p}'_N = \hat{\beta}_P \quad (4.4.82)$$

with

$$\hat{v}_P = -\hat{v}_E - \hat{v}_W - \hat{v}_N \quad (4.4.83)$$

$$\hat{v}_E = -\frac{\hat{\rho} \hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \quad (4.4.84)$$

$$\hat{v}_W = -\frac{\hat{\rho} \hat{A}_x^2}{\hat{a}_{u,P}^{centre}} \quad (4.4.85)$$

$$\hat{v}_N = -\frac{\hat{\rho} \hat{A}_y^2}{\hat{a}_{v,N}^{centre}} \quad (4.4.86)$$

$$\hat{\beta}_P = -\hat{A}_x \hat{\rho} \hat{u}_e + \hat{A}_x \hat{\rho} \hat{u}_w - \hat{A}_y \hat{\rho} \hat{v}_n \quad (4.4.87)$$

4.5 Backwards Facing Step

The model for the backwards facing step is constructed in the same way as the straight channel model, by use of global indexing. The global indexing starts in the lower left corner right after the step as in the simple illustration in figure 4.1 for an example resolution of 6 nodes in y -direction and 88 nodes in x -direction. Red numbers are scalar nodes, green nodes are u -velocity nodes and blue nodes are v -velocity nodes in accordance with the staggered grid.

440	352	352	264	264	176	176	88	88	12	12
439	351	351	263	263	175	175	87	87	11	11
439	351	351	263	263	175	175	87	87	11	11
438	350	350	262	262	174	174	86	86	10	10
437	349	349	261	261	173	173	85	85	9	9
437	349	349	261	261	173	173	85	85	9	9
436	348	348	260	260	172	172	84	84	8	8
436	348	348	260	260	172	172	84	84	8	8
435	347	347	259	259	171	171	83	83	7	7
435	347	347	259	259	171	171	83	83	7	7
434	346	346	258	258	170	170	82	82	6	6
434	346	346	258	258	170	170	82	82	6	6
433	345	345	257	257	169	169	81	81	5	5
433	345	345	257	257	169	169	81	81	5	5
432	344	344	256	256	168	168	80	80	4	4
432	344	344	256	256	168	168	80	80	4	4
431	343	343	255	255	167	167	79	79	3	3
431	343	343	255	255	167	167	79	79	3	3
430	342	342	254	254	166	166	78	78	2	2
430	342	342	254	254	166	166	78	78	2	2
429	341	341	253	253	165	165	77	77	1	1
429	341	341	253	253	165	165	77	77	1	1
428	340	340	252	252	164	164	77	77	1	1
428	340	340	252	252	164	164	77	77	1	1
427	339	339	251	251	163	163	77	77	1	1
427	339	339	251	251	163	163	77	77	1	1
426	338	338	250	250	162	162	77	77	1	1
426	338	338	250	250	162	162	77	77	1	1
425	337	337	249	249	161	161	77	77	1	1
425	337	337	249	249	161	161	77	77	1	1
424	336	336	248	248	160	160	77	77	1	1
424	336	336	248	248	160	160	77	77	1	1
423	335	335	247	247	159	159	77	77	1	1
423	335	335	247	247	159	159	77	77	1	1

Figure 4.1: Global indexing in the backwards facing step domains.

4.5.1 Boundary Conditions for the Backwards Facing Step

The boundary conditions for the two dimensional straight channel as described in section 4.4 are also applicable for the backwards facing step boundaries. This covers the inlet, outlet and walls for the backwards facing step. The southern wall is not one continuous boundary like for the straight channel, but the southern wall boundary condition is applied to both the two segments of southern wall in the domain. This leaves the western wall of the step in need for a boundary condition, as well as a special implementation around the corner of the step.

4.5.1.1 Western Wall at the Step

At the western wall after the backwards facing step, the \hat{u} -velocity nodes coincide with the wall instead of the \hat{v} -velocity nodes like for the northern and southern wall. Due to the staggered grid, the \hat{v} -velocity nodes are placed so that the faces of the control volumes around the nodes line up with the walls, while the nodes themselves are located at a distance $\delta\hat{x}/2$ from the wall where $\delta\hat{x}$ is the width of the control volumes.

4.5.1.1.1 Momentum Equation for the x -Component

The u -velocity nodes coincide with the wall and the known west velocity node can be inserted directly. The Momentum Equation for the x -Component at the west wall boundary becomes equation (4.5.1) with the coefficients in equations (4.5.2)-(4.5.6). The western velocity node is known and equal to zero and is omitted from the equation.

$$\hat{a}_P\hat{u}_P + \hat{a}_E\hat{u}_E + \hat{a}_N\hat{u}_N + \hat{a}_S\hat{u}_S = \hat{b}_P \quad (4.5.1)$$

with

$$\hat{a}_P = -\hat{a}_E - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e}\hat{A}_x - \hat{F}_{x,w}\hat{A}_y + \hat{F}_{y,n}\hat{A}_y - \hat{F}_{y,s}\hat{A}_y + \max(\hat{F}_{x,w}\hat{A}_x, 0) + \hat{D}_x\hat{A}_x \quad (4.5.2)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e}\hat{A}_x) - \hat{D}_x\hat{A}_x \quad (4.5.3)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n}\hat{A}_y) - \hat{D}_y\hat{A}_y \quad (4.5.4)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s}\hat{A}_y, 0) - \hat{D}_y\hat{A}_y \quad (4.5.5)$$

$$\hat{b}_P = -(\hat{p}_P - \hat{p}_W)\hat{A}_x \quad (4.5.6)$$

4.5.1.1.2 Momentum Equation for the y -Component

For the v -velocity, the implementation of the boundary condition at the western wall starts with the right side of the discretised momentum equation after the integration over the control volume as seen in equation (4.5.7). The left hand side of the equation is kept as before.

$$\text{LHS} = -\left.\frac{\partial \hat{p}}{\partial \hat{y}}\right|_P \delta \hat{y} \hat{A}_y + \frac{1}{Re} \hat{\mu} \left.\frac{\partial \hat{v}}{\partial \hat{x}}\right|_e \hat{A}_x - \frac{1}{Re} \hat{\mu} \left.\frac{\partial \hat{v}}{\partial \hat{x}}\right|_w \hat{A}_x + \frac{1}{Re} \hat{\mu} \left.\frac{\partial \hat{v}}{\partial \hat{y}}\right|_n \hat{A}_y - \frac{1}{Re} \hat{\mu} \left.\frac{\partial \hat{v}}{\partial \hat{y}}\right|_s \hat{A}_y \quad (4.5.7)$$

The gradient at the western cell face is defined as equation (4.5.8) by use of a central difference.

$$\left.\frac{\partial \hat{v}}{\partial \hat{x}}\right|_w = \frac{\hat{v}_P - \hat{v}_{wall}}{\delta \hat{x}/2} \quad (4.5.8)$$

The distance from the centre node \hat{v}_P to the wall is $\delta \hat{y}/2$. Like for the southern and northern walls, this incorporates a shear force into the source term of the momentum equation. The wall shear stress and the shear force are defined in equations (4.4.48) and (4.4.49). The approximated gradient in equation (4.5.8) in addition to the central differences for the remaining gradients in equation (4.5.7) are inserted back into the right hand side of the y -Momentum equation, and the equation is rearranged to yield equation (4.5.9) in combination with the left side of the equation. The coefficients are given in equations (4.5.10)-(4.5.14). The known $\hat{v}_{wall} = 0$ is omitted from the source term.

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_N \hat{v}_N + \hat{a}_S \hat{v}_S = \hat{b}_P \quad (4.5.9)$$

with

$$\hat{a}_P = -\hat{a}_E - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e}\hat{A}_x - \hat{F}_{x,w}\hat{A}_y + \hat{F}_{y,n}\hat{A}_y - \hat{F}_{y,s}\hat{A}_y + \max(\hat{F}_{x,w}\hat{A}_x, 0) + 2\hat{D}_x\hat{A}_x \quad (4.5.10)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e}\hat{A}_x) - \hat{D}_x\hat{A}_x \quad (4.5.11)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n}\hat{A}_y) - \hat{D}_y\hat{A}_y \quad (4.5.12)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s}\hat{A}_y, 0) - \hat{D}_y\hat{A}_y \quad (4.5.13)$$

$$\hat{b}_P = -(\hat{p}_P - \hat{p}_S)\hat{A}_y \quad (4.5.14)$$

4.5.1.1.3 Pressure Correction Equation

The western velocity node is \hat{u}_{wall} which is known and equal to zero, and no pressure correction is needed. The \hat{v}_{wall} velocity does not occur in the pressure correction at this point. \hat{u}_{wall} can be directly inserted into the Continuity equation under the derivation of the pressure correction equation and no link is then created to the western boundary. The result is equation 4.5.15 with the coefficients in equations (4.5.16)-(4.5.20). The known $\hat{u}_{wall} = 0$ is omitted from the equation.

$$\hat{v}_P\hat{p}'_P + \hat{v}_E\hat{p}'_E + \hat{v}_N\hat{p}'_N + \hat{v}_S\hat{p}'_S = \hat{\beta}_P \quad (4.5.15)$$

with

$$\hat{v}_P = -\hat{v}_E - \hat{v}_N - \hat{v}_S \quad (4.5.16)$$

$$\hat{v}_E = -\frac{\rho\hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \quad (4.5.17)$$

$$\hat{v}_N = -\frac{\rho\hat{A}_y^2}{\hat{a}_{v,N}^{centre}} \quad (4.5.18)$$

$$\hat{v}_S = -\frac{\rho\hat{A}_y^2}{\hat{a}_{v,P}^{centre}} \quad (4.5.19)$$

$$\hat{\beta}_P = -\hat{A}_x\hat{\rho}\hat{u}_e - \hat{A}_y\hat{\rho}\hat{v}_n + \hat{A}_y\hat{\rho}\hat{v}_s \quad (4.5.20)$$

4.5.1.2 Corner points

The v -velocity node directly right of the corner of the BFS-step and the u -velocity node directly above the corner need a special treatment different from the other sections of the domain. This is because the adjacent node cells that contribute to the equations for these points are one wall and one normal node. This means that the wall friction should be halved, since only half the cell face coincides with the wall. The pressure correction equation does not need an alteration at the corner.

Figure 4.2 shows the node points around the corner. Nodes u_{164} and v_{77} are the nodes in question. This numbering is for a coarseness of 88 computational points in total in the x -direction and 6 computational points in total in the y -direction and corresponds to the global indexing in figure 4.1. This is an example resolution that is not used in the simulations.

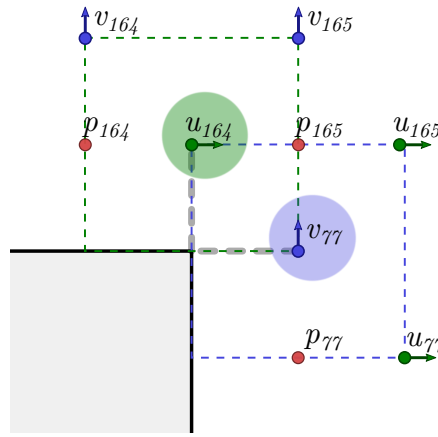


Figure 4.2: Indexed computational points around the backwards facing step.

The implementation for the u -velocity follows that of the southern wall, but with the shear stress halved like seen in equation (4.5.21)

$$\left. \frac{\partial \hat{u}}{\partial \hat{y}} \right|_s = \frac{1}{2} \frac{\hat{u}_P - \hat{u}_{wall}}{\delta \hat{y} / 2} \quad (4.5.21)$$

This yields equation (4.5.22) with the coefficients in equations (4.5.23)-(4.5.27).

$$\hat{a}_P \hat{u}_P + \hat{a}_E \hat{u}_E + \hat{a}_W \hat{u}_W + \hat{a}_N \hat{u}_N + \hat{a}_S \hat{u}_N = \hat{b}_P \quad (4.5.22)$$

with

$$\begin{aligned} \hat{a}_P = & -\hat{a}_E - \hat{a}_W - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_y - \hat{F}_{x,w} \hat{A}_y + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y - \\ & + \max(\hat{F}_{y,s} \hat{A}_y, 0) + \hat{D}_y \hat{A}_y \end{aligned} \quad (4.5.23)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e} \hat{A}_y) - \hat{D}_x \hat{A}_y \quad (4.5.24)$$

$$\hat{a}_W = -\max(\hat{F}_{x,w} \hat{A}_y, 0) - \hat{D}_x \hat{A}_y \quad (4.5.25)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.5.26)$$

$$\hat{b}_P = -\left(\hat{p}_P - \hat{p}_W \right) \hat{A}_x \quad (4.5.27)$$

Similarly, the implementation for the v -velocity at the corner follows that of the western wall, but with the shear stress halved like seen in equation (4.5.28) .

$$\left. \frac{\partial \hat{v}}{\partial \hat{x}} \right|_w = \frac{1}{2} \frac{\hat{v}_P - \hat{v}_{wall}}{\delta \hat{x}/2} \quad (4.5.28)$$

This yields equation (4.5.29) with the coefficients as given in equations (4.5.30)-(4.5.34).

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_N \hat{v}_N + \hat{a}_S \hat{v}_S = \hat{b}_P \quad (4.5.29)$$

with

$$\begin{aligned} \hat{a}_P = -\hat{a}_E - \hat{a}_N - \hat{a}_S + \hat{F}_{x,e} \hat{A}_x - \hat{F}_{x,w} \hat{A}_y + \hat{F}_{y,n} \hat{A}_y - \hat{F}_{y,s} \hat{A}_y \\ + \max(\hat{F}_{x,w} \hat{A}_x, 0) + \hat{D}_x \hat{A}_x \end{aligned} \quad (4.5.30)$$

$$\hat{a}_E = -\max(0, -\hat{F}_{x,e} \hat{A}_x) - \hat{D}_x \hat{A}_x \quad (4.5.31)$$

$$\hat{a}_N = -\max(0, -\hat{F}_{y,n} \hat{A}_y) - \hat{D}_y \hat{A}_y \quad (4.5.32)$$

$$\hat{a}_S = -\max(\hat{F}_{y,s} \hat{A}_y, 0) - \hat{D}_y \hat{A}_y \quad (4.5.33)$$

$$\hat{b}_P = -\left(\hat{p}_P - \hat{p}_S\right) \hat{A}_y \quad (4.5.34)$$

4.6 Dimensionless Equations For Comparison

For comparing the results to existing literature on flow over the backwards facing step, an article published by Biswas et al. [4] will be used. A different scale for the geometrical length scales in the domain is used. Instead of scaling the lengths, areas and volumes with the hydraulic diameter D_{hyd} , Biswas et al. [4] scaled these parameters with h , the initial height of the channel. $D_{hyd} = 2h$ is used for the hydraulic diameter. This means that the scaling factor used in Biswas et al. [4] is equal to $\frac{D_{hyd}}{2}$. A parabolic inlet profile will be used instead of a constant inlet velocity, and u_{avg} is used as scale instead of u_{in} for the velocities and in the pressure scale. Below follow updated dimensionless equations for implementation to obtain a model that fits the settings used by Biswas et al. [4].

4.6.1 Variables as functions of their dimensionless form

All variables, spatial parameters, operators and tensors expressed with dimensionless parameters for interchanging in the transport equations are given in equations (4.6.1)-(4.6.18).

$$\mathbf{u} = u_{avg} \hat{\mathbf{u}} \quad (4.6.1) \quad A_x = h^2 \hat{A}_x \quad (4.6.10)$$

$$\tilde{p} = \rho u_{avg}^2 \hat{p} \quad (4.6.2) \quad A_y = h^2 \hat{A}_y \quad (4.6.11)$$

$$\mu = \mu \hat{\mu} \quad (4.6.3) \quad dA = h^2 d\hat{A} \quad (4.6.12)$$

$$\rho = \rho \hat{\rho} \quad (4.6.4) \quad V = h^3 \hat{V} \quad (4.6.13)$$

$$\delta x = h \delta \hat{x} \quad (4.6.5) \quad dV = h^3 d\hat{V} \quad (4.6.14)$$

$$\delta y = h \delta \hat{y} \quad (4.6.6) \quad \sigma = \bar{\sigma} \hat{\sigma} \quad (4.6.15)$$

$$\frac{\partial}{\partial x} = \frac{1}{h} \frac{\partial}{\partial \hat{x}} \quad (4.6.7) \quad \sigma_{xx} = -\frac{\mu u_{avg}}{h} 2\hat{\mu} \frac{\partial \hat{u}}{\partial \hat{x}} \quad (4.6.16)$$

$$\frac{\partial}{\partial y} = \frac{1}{h} \frac{\partial}{\partial \hat{y}} \quad (4.6.8) \quad \sigma_{yy} = -\frac{\mu u_{avg}}{h} 2\hat{\mu} \frac{\partial \hat{v}}{\partial \hat{y}} \quad (4.6.17)$$

$$\nabla = \frac{1}{h} \hat{\nabla} \quad (4.6.9) \quad \sigma_{xy} = -\frac{\mu u_{avg}}{h} \hat{\mu} \left[\frac{\partial \hat{u}}{\partial \hat{x}} + \frac{\partial \hat{v}}{\partial \hat{y}} \right] \quad (4.6.18)$$

4.6.2 Governing equations

The Continuity equation looks identical with the new scaling factor, as the geometrical scale vanishes like in equation (3.4.59). The Momentum equation is made dimensionless by interchanging the dimensionless variables in equations (4.6.1)-(4.6.18) as seen in equations (4.6.19)-(4.6.25).

$$\nabla \cdot (\rho \mathbf{u}) = -\nabla \tilde{p} - \nabla \cdot \sigma \quad (4.6.19)$$

$$\frac{\rho u_{in}^2}{h} \hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}}) = -\frac{\bar{p}}{h} \hat{\nabla} \hat{p} - \frac{\bar{\sigma}}{h} \hat{\nabla} \cdot \hat{\sigma} \quad (4.6.20)$$

$$\frac{\rho u_{in}^2}{h} \hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}}) = -\frac{\rho u_{in}^2}{h} \hat{\nabla} \hat{p} - \frac{\mu u_{avg}}{h^2} \hat{\nabla} \cdot \hat{\sigma} \quad (4.6.21)$$

$$\hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}}) = -\hat{\nabla} \hat{p} - \frac{\mu u_{avg}}{h^2} \frac{h}{\rho u_{in}^2} \hat{\nabla} \cdot \hat{\sigma} \quad (4.6.22)$$

$$\hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}}) = -\hat{\nabla} \hat{p} - \frac{\mu}{\rho u_{avg} h} \hat{\nabla} \cdot \hat{\sigma} \quad (4.6.23)$$

$$\hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}}) = -\hat{\nabla} \hat{p} - \frac{\mu u_{avg}}{h^2} \frac{h}{\rho u_{in}^2} \hat{\nabla} \cdot \hat{\sigma} \quad (4.6.24)$$

$$\hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}}) = -\hat{\nabla} \hat{p} - \frac{2}{Re} \hat{\nabla} \cdot \hat{\sigma} \quad (4.6.25)$$

The rest of the discretisation follows the steps as given in section (3.4). The result is equation (4.6.26) with the coefficients in equations (4.6.27)-(4.6.32) for the x -Momentum equation.

$$\hat{a}_{i,J} \hat{u}_{i,J} + \hat{a}_{i+1,J} \hat{u}_{i+1,J} + \hat{a}_{i-1,J} \hat{u}_{i-1,J} + \hat{a}_{i,J+1} \hat{u}_{i,J+1} + \hat{a}_{i,J-1} \hat{u}_{i,J-1} = \hat{b}_{i,J} \quad (4.6.26)$$

with

$$\hat{a}_{i,J} = -\hat{a}_{i+1,J} - \hat{a}_{i-1,J} - \hat{a}_{i,J+1} - \hat{a}_{i,J-1} + \hat{F}_{x,e}\hat{A}_x - \hat{F}_{x,w}\hat{A}_y + \hat{F}_{y,n}\hat{A}_y - \hat{F}_{y,s}\hat{A}_y \quad (4.6.27)$$

$$\hat{a}_{i+1,J} = -\max(0, -\hat{F}_{x,e}\hat{A}_x) - \hat{D}_x\hat{A}_x \quad (4.6.28)$$

$$\hat{a}_{i-1,J} = -\max(\hat{F}_{x,w}\hat{A}_y, 0) - \hat{D}_x\hat{A}_y \quad (4.6.29)$$

$$\hat{a}_{i,J+1} = -\max(0, -\hat{F}_{y,n}\hat{A}_y) - \hat{D}_y\hat{A}_y \quad (4.6.30)$$

$$\hat{a}_{i,J-1} = -\max(\hat{F}_{y,s}\hat{A}_y, 0) - \hat{D}_y\hat{A}_y \quad (4.6.31)$$

$$\hat{b}_{i,J} = -\left(\hat{p}_{I,J} - \hat{p}_{I-1,J}\right)\hat{A}_x \quad (4.6.32)$$

Equation (4.6.33) with the coefficients in equations (4.6.34)-(4.6.39) is the y -Momentum equation.

$$\hat{a}_{I,j}\hat{v}_{I,j} + \hat{a}_{I+1,j}\hat{v}_{I+1,j} + \hat{a}_{I-1,j}\hat{v}_{I-1,j} + \hat{a}_{I,j+1}\hat{v}_{I,j+1} + \hat{a}_{I,j-1}\hat{v}_{I,j-1} = \hat{b}_{I,j} \quad (4.6.33)$$

with

$$\hat{a}_{I,j} = -\hat{a}_{I+1,j} - \hat{a}_{I-1,j} - \hat{a}_{I,j+1} - \hat{a}_{I,j-1} + \hat{F}_{x,e}\hat{A}_x - \hat{F}_{x,w}\hat{A}_y + \hat{F}_{y,n}\hat{A}_y - \hat{F}_{y,s}\hat{A}_y \quad (4.6.34)$$

$$\hat{a}_{I+1,j} = -\max(0, -\hat{F}_{x,e}\hat{A}_x) - \hat{D}_x\hat{A}_x \quad (4.6.35)$$

$$\hat{a}_{I-1,j} = -\max(\hat{F}_{x,w}\hat{A}_y, 0) - \hat{D}_x\hat{A}_y \quad (4.6.36)$$

$$\hat{a}_{I,j+1} = -\max(0, -\hat{F}_{y,n}\hat{A}_y) - \hat{D}_y\hat{A}_y \quad (4.6.37)$$

$$\hat{a}_{I,j-1} = -\max(\hat{F}_{y,s}\hat{A}_y, 0) - \hat{D}_y\hat{A}_y \quad (4.6.38)$$

$$\hat{b}_{I,j} = -\left(\hat{p}_{I,J} - \hat{p}_{I,J-1}\right)\hat{A}_y \quad (4.6.39)$$

The change in the factor in front of the diffusive terms is given in the coefficient D as given in equation (4.6.40).

$$\hat{D}_x = \frac{2}{Re} \frac{\hat{\mu}}{\delta \hat{x}} \quad \hat{D}_y = \frac{2}{Re} \frac{\hat{\mu}}{\delta \hat{y}} \quad (4.6.40)$$

4.7 Convergence Criteria

Three types of convergence criteria are used, which must all be satisfied when the model is converged.

The first type criterion C_1 is the residual of the momentum equation on the form of equation (4.7.1).

$$\mathbf{U} \cdot \mathbf{u}^* - b_u = R_u \quad (4.7.1)$$

\mathbf{U} is the coefficient matrix and b_u is the source term for the u -velocity, while u^* is the calculated velocity after matrix inversion in the current iteration. C_1 is defined as given in equation (4.7.2).

$$C_1 = \sqrt{R_u \cdot R_u^T} \quad (4.7.2)$$

C_2 is the corresponding convergence criterion for the v -velocity as defined in equation (4.7.3).

$$C_2 = \sqrt{R_v \cdot R_v^T} \quad (4.7.3)$$

The second type criterion C_3 is a summation of the source term of the pressure correction β . β is equal to the Continuity equation, and the criterion C_3 determines if the Continuity equation is fulfilled and the pressure corrections are close to zero. C_3 is found by taking the absolute value of the sum of all the entries in the vector β like defined in equation (4.7.4)

$$C_3 = \left| \sum \beta \right| \quad (4.7.4)$$

The third type convergence criteria C_4 checks the difference between the velocity u^* after the matrix inversion and the initial guess u^{circ} coming into the current iteration. C_4 is defined as in equation (4.7.5).

$$C_4 = \max(|u^\circ - u^*|) \quad (4.7.5)$$

C_5 is the corresponding convergence criterion for the v -velocity and is defined in equation (4.7.6).

$$C_5 = \max(|v^\circ - v^*|) \quad (4.7.6)$$

The convergence criteria C_1 , C_2 , C_4 and C_5 can be normalised with respect to the inlet velocity u_{in} or the average inlet velocity u_{avg} . Since the model is dimensionless and u_{in} or u_{avg} is used as a scale for the velocity, they are equal to 1 in the model and are therefore not shown in the expressions above.

The convergence criteria for all the two dimensional models were taken as in equations (4.7.7)-(4.7.11).

$$C_1 < 10^{-8} \quad (4.7.7)$$

$$C_2 < 10^{-8} \quad (4.7.8)$$

$$C_3 < 10^{-10} \quad (4.7.9)$$

$$C_4 < 10^{-8} \quad (4.7.10)$$

$$C_5 < 10^{-8} \quad (4.7.11)$$

A comparison was made testing with the limits for C_1 , C_2 , C_4 and C_5 set to 10^{-6} , 10^{-7} , 10^{-8} and 10^{-9} . It was found that there was not a significant change in the results between 10^{-8} and 10^{-9} , so 10^{-8} is assumed sufficient.

The convergence criteria C_1 , C_2 and C_3 are dependent on the number of computational nodes used in the domain and will by definition be larger when a higher number of nodes are used. The limits may need adjusting if a different set of computational nodes than what is specified in section 4.2 is used. For the convergence criteria C_4 and C_5 the **max** operator is used, and the criteria are therefore not dependent on the number of computational nodes used in the domain.

4.8 Plotting

The converged results are plotted using surface plots and velocity vector plots, also known as quiver plots. The model results are dimensionless variables that must be transferred back to their normal values before plotting.

4.8.1 Obtaining the Dimensional Variables

Equation (4.8.1) shows the relation for obtaining the ordinary velocity from the dimensionless velocity.

$$\mathbf{u} = u_{in} \hat{\mathbf{u}} \quad (4.8.1)$$

Equation (4.8.2) shows the definition of the dimensionless adjusted pressure \hat{p} which is calculated in the model.

$$\hat{p} = \frac{\tilde{p}}{\rho u_{in}^2} = \frac{p - p_{out}}{\rho u_{in}^2} \quad (4.8.2)$$

The ordinary pressure can be obtained by equation (4.8.3) for the plotting.

$$p = \rho u_{in}^2 \hat{p} + p_{out} \quad (4.8.3)$$

The pressure correction is obtained by equation (4.8.4).

$$p' = \rho u_{in}^2 \hat{p}' \quad (4.8.4)$$

4.8.2 Velocity Vector Plots

For the velocity vector plots, a combined velocity variable must be made, combining the u - and v - velocity components. Due to the use of a staggered grid, the velocity components are first obtained at the locations of the scalar node points by interpolation as in equations (4.8.5) and (4.8.6).

$$u_{I,J} = \frac{1}{2} (u_{i-1,J} + u_{i,J}) \quad (4.8.5)$$

$$v_{I,J} = \frac{1}{2} (v_{I,j-1} + v_{I,j}) \quad (4.8.6)$$

Figure 4.3 shows the scalar node point $p_{I,J}$ and the surrounding node points used to calculate the velocities at the scalar nodes. The **MATLAB** plotting function **quiver** can

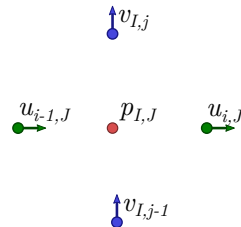


Figure 4.3: The points included in the calculation of velocity for quiver/contour plots.

then be used to obtain a velocity vector plot using the u - and v components $u_{I,J}$ and $v_{I,J}$ located at the scalar nodes. The first scalar node after the inlet is located at $\delta x/2a$ half control volume with from the inlet

The `MATLAB` plotting function `contour` is used to create a contour plot for combination with the vector plot. For this, the magnitude of the combined velocities is needed, which is found by equation (4.8.7) for the velocities at scalar nodes [30].

$$|\mathbf{u}_{I,J}| = \sqrt{u_{I,J}^2 + v_{I,J}^2} \quad (4.8.7)$$

4.9 Composition and Working Principle of the Code

In this section, a map presenting the composition of the two dimensional backwards facing step models is given. The map shows how the model is divided into scripts, functions and other elements as can be seen from the legend on the bottom right on page 63. The map also describes how the model for the two dimensional straight channel is build up, the difference is that the contents of the scripts labelled `u_velocity`, `v_velocity` and `pressure correction` are given directly in the main and not saved in individual scripts like for the backwards facing step models. In the two dimensional straight channel model, the helper functions are not needed. The order of calculation in the code follows the visualisation in figure 2.8.

The `main` contains the definitions of all the fluid properties and the `while` loop that runs for each iteration until convergence is reached. The coefficients F are obtained from the velocities at the previous iteration before the velocities `u_star` and `v_star` are obtained using F . `u_star` and `v_star` are then used in `beta` to obtain the pressure correction `p_corr`.

4.9.1 Code Options

Some options to plot additional parameters or to modify the models in the codes are available in the beginning of the two dimensional straight channel model and the backwards facing step model with a constant inlet velocity. Some of the options were useful in order to locate mistakes in the troubleshooting phase of the work, and others create extra plots that may be interesting. These options are explained in this section.

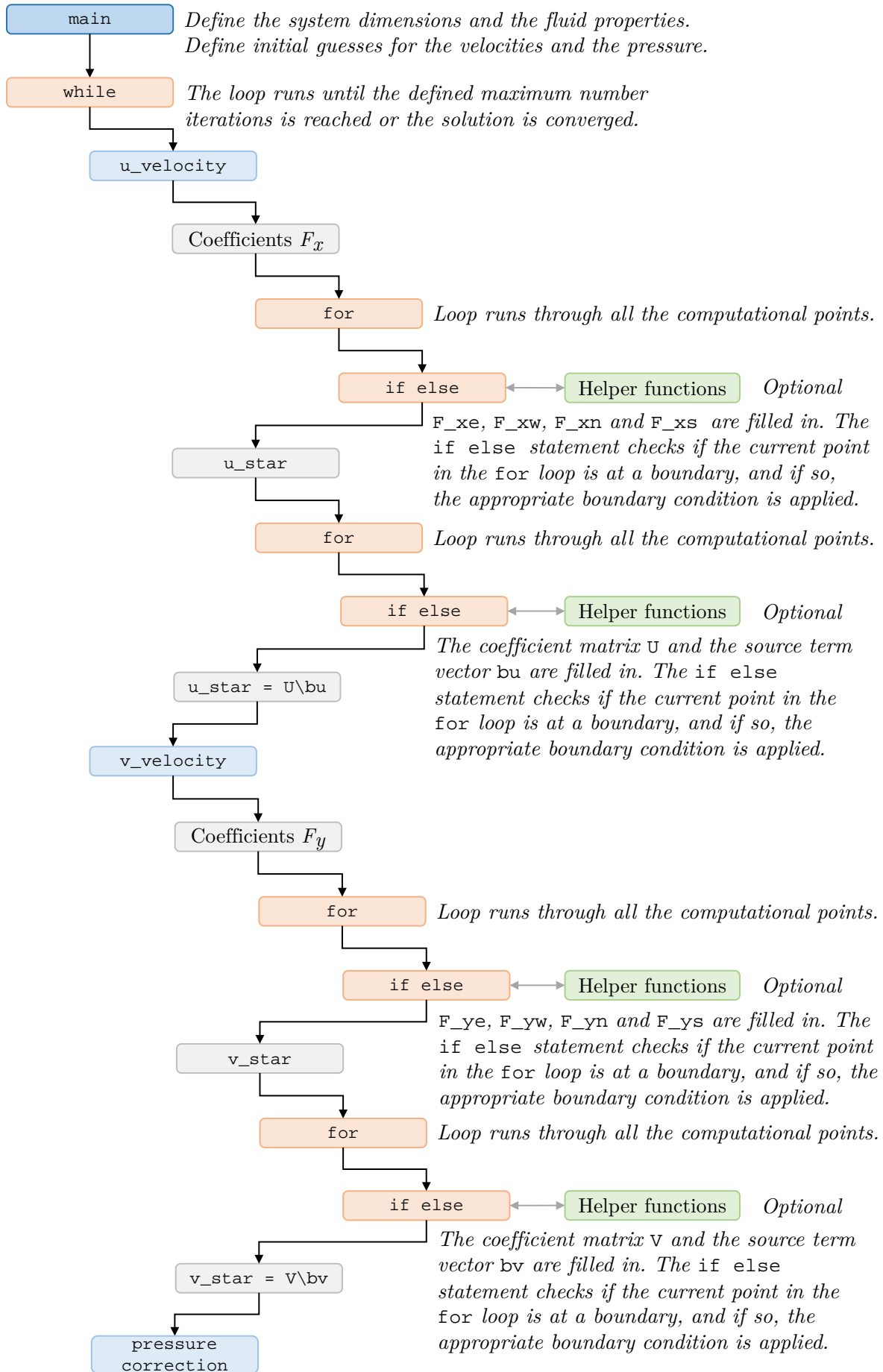
4.9.1.1 Plot Initial Guesses

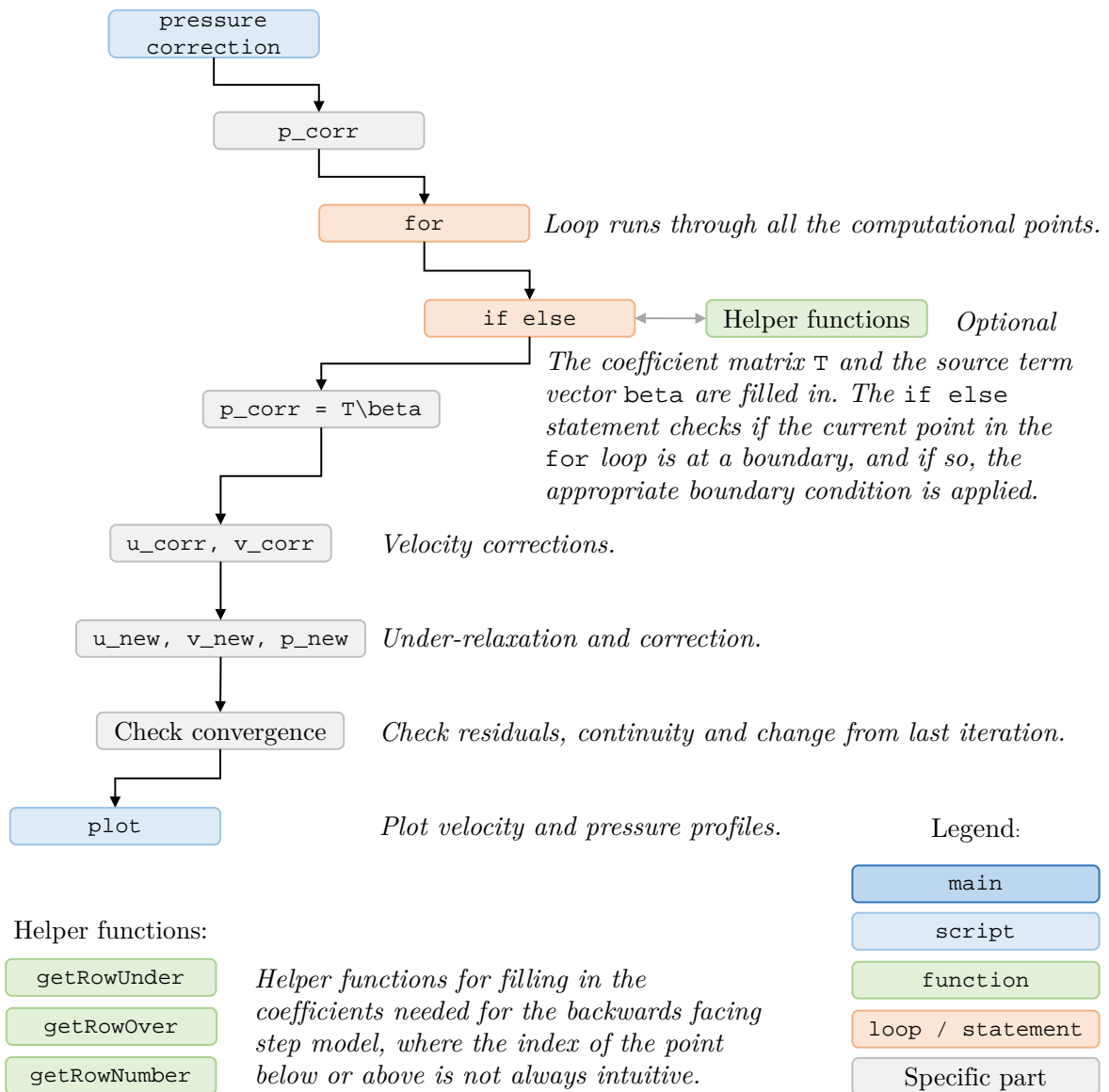
The option `plotInitialProfiles` plots the initial guesses of the velocities and the pressure.

4.9.1.2 Plot Profiles After Each Iteration

With the option `plotiterationwise` enabled, the velocity, pressure and pressure correction profiles are plotted after every iteration before pausing. This option was useful when troubleshooting, as it made it possible to see in an easy manner if the solution is developing in the correct direction after each update.

The option `printSetPlotIt` plots the velocity, pressure and pressure correction profiles are plotted each iteration specified and saved to a `.gif` file. The option `gifIntermediates` additionally creates a `.gif` file with the initial guess, intermediate, correction and new values of the two velocity components.





4.9.1.3 Disable Solution of v -velocity

With the option `solvvvel`, the solution of the v -velocity component can be switched off. In that case, the v -velocity component is set to zero across the whole domain. This is not a realistic result for the models with a constant inlet velocity, but was still a method to try to isolate the errors during debugging, as approximately one third of the code is decoupled from the main.

4.9.1.4 Additional Plots

The options `plotCircVels` and `plotCorrVels` enables plotting of the intermediate velocities u^* and v^* and the velocity corrections u' and v' respectively. In combination with the `plotiterationwise` option, this allows for all the calculations and updates in the models to be investigated.

4.9.1.5 Remove the Backwards Facing Step

The option `onlyChannel` in the backwards facing step model blocks off the backwards facing step so that the domain becomes a straight channel. This was useful when

debugging the backwards facing step model, as it could be discovered if a mistake was related to the step.

5

Results

The results for the fluid flow models for two dimensions are given in this chapter. Three different `MATLAB` models were used to obtain the results, one for the two dimensional straight channel, one for the backwards facing step domain as used by Melaaen [3] and one for the backwards facing step domain as used by Biswas et al. [4]. The results for the one dimensional model are given in appendix B.

5.1 Two Dimensional Straight Channel

In this section, the results from the two dimensional straight channel model are given. The `MATLAB` code `channel_2D.m` was used to obtain the results, and the code is given in appendix E.

Table 5.1 shows the number of iterations and convergence times for the two dimensional model for different channel lengths L . The short channel with length $L = 3$ corresponds to the inlet section before the backwards facing step domain in figure 1.2 as used by Melaaen [3], and shows the behaviour of the flow when it is not fully developed. The long channel with $L = 22$ corresponds to the length of the whole backwards facing step domain. N and M are the number of scalar node points in x - and y -direction, and *Total* signifies the total amount of scalar node pints. 18 times 88 points were chosen as the resolution because this corresponds to the maximum possible resolution obtained for the BFS models.

Re	L	N	M	Total	Iterations	Time
560	3 m	88	18	1584	2098	19 min
560	22 m	88	18	1584	2075	20 min
1120	3 m	88	18	1584	2105	21 min
1120	22 m	88	18	1584	2096	19 min

Table 5.1: Different convergence times for different numbers of computational nodes for the two dimensional model.

The plots shown below are for the simulation with Reynolds number $Re = 560$.

5.1.1 Short channel

In this section, the surface plots of the fluid flow parameters in a short channel with length $L = 3$ are given. The height of the channel is $h = 1$. 18 times 88 computational points were used for all the plots below and they are shown from both the inlet and the outlet. The Reynolds number Re is equal to 560.

Figure 5.1 shows the u -velocity component profile for the short channel seen from the inlet and figure 5.2 shows the same profile seen from the outlet. As can be seen, the profile is not fully developed as the outlet profile is not yet a proper parabola.

Figure 5.3 shows the v -velocity component profile for the short channel seen from the inlet and figure 5.4 shows the same profile seen from the outlet. There is an increase in the v -velocity near the southern wall and a decrease near the northern wall after the inlet. The positive flow direction for the v -velocity is upwards, which means that this increase and decrease reflects a flow inwards towards the centre of the channel. This corresponds well to the behaviour that is to be expected due to the friction from the walls with a constant inlet velocity profile. The friction is largest towards the inlet, since the inlet u -velocity is constant for all y . As can be seen, the profile is not fully developed as the velocity at the outlet has not reached zero.

Figure 5.5 shows the pressure profile for the short channel seen from the inlet and figure 5.6 shows the same profile seen from the outlet. Note that the scale has a low variation, which means that the pressure is close to constant across the domain. The slight increase in pressure at the walls at the inlet corresponds to the sharp velocity gradients in these points, as can be seen at the same location in the velocity plots in figures 5.1 and 5.3.

Figure 5.7 shows the pressure correction for the short channel seen from the inlet and figure 5.8 shows the same profile seen from the outlet. Note that the scale is of order of magnitude 10^{-10} Pa. When converged, the pressure correction should be close to zero across the domain for the continuity equation to be fulfilled. The outlet pressure is known and the pressure correction is therefore plotted as zero at the last point in the plot at the outlet. The pressure correction does not smoothly approach zero at the outlet as there is a small increase in the centre of the channel and decrease towards the walls of the channel. This may mean that the outlet boundary condition is not completely satisfied.

The flow in this case is not fully developed, which may cause some problems. At the outlet, the velocity gradients $\frac{\partial u}{\partial x}$ and $\frac{\partial v}{\partial x}$ are not specified to be zero, which would be another possible outlet boundary condition instead of specifying the outlet pressure. For the last computational point, the convective mass flux at the east cell face $F_{x,e}$ is still specified to be equal to $F_{x,w}$, the convective mass flux at the west cell face. This is not completely accurate when the flow is not developed.

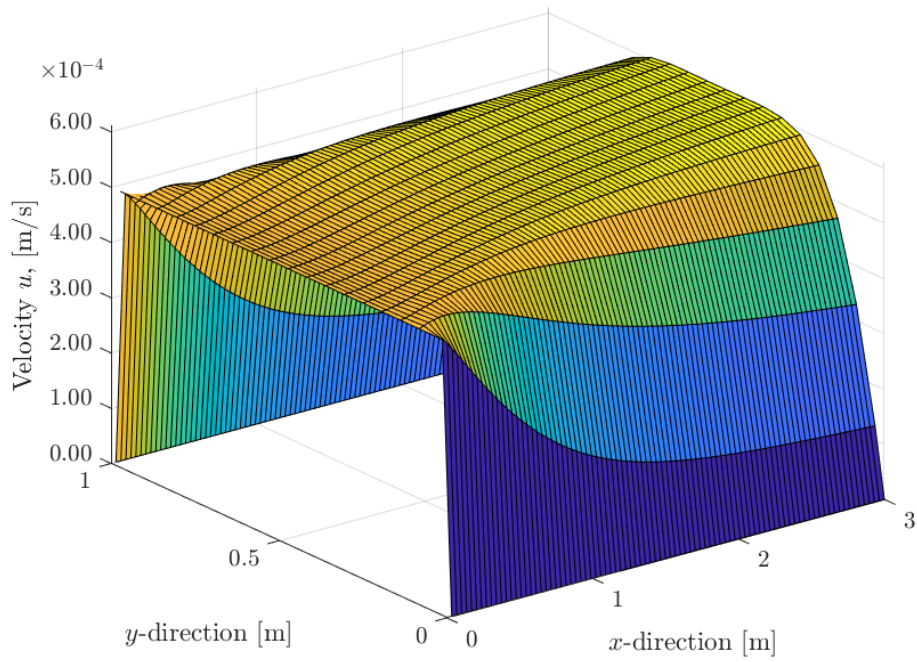


Figure 5.1: u -velocity seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

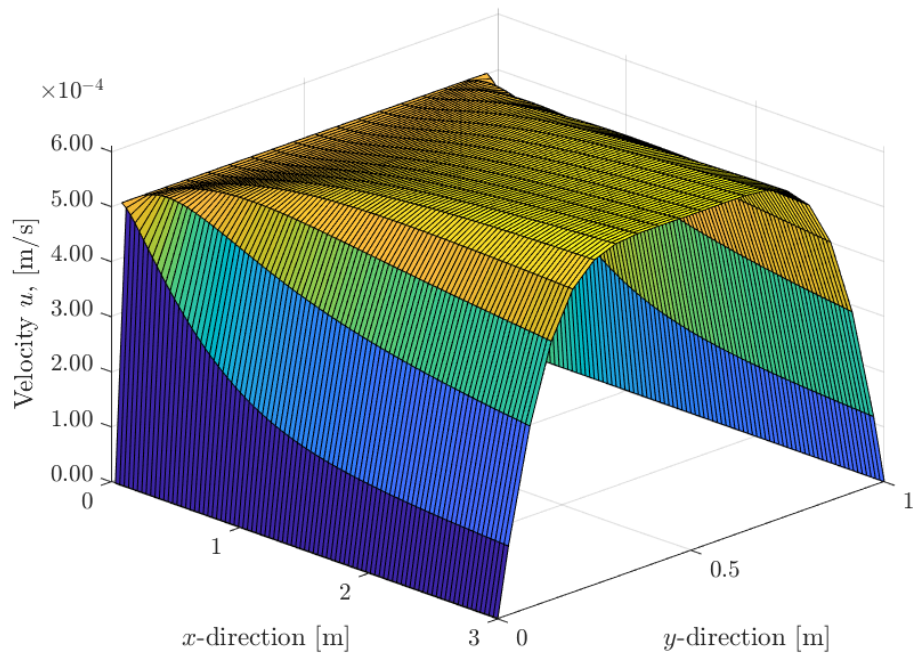


Figure 5.2: u -velocity seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

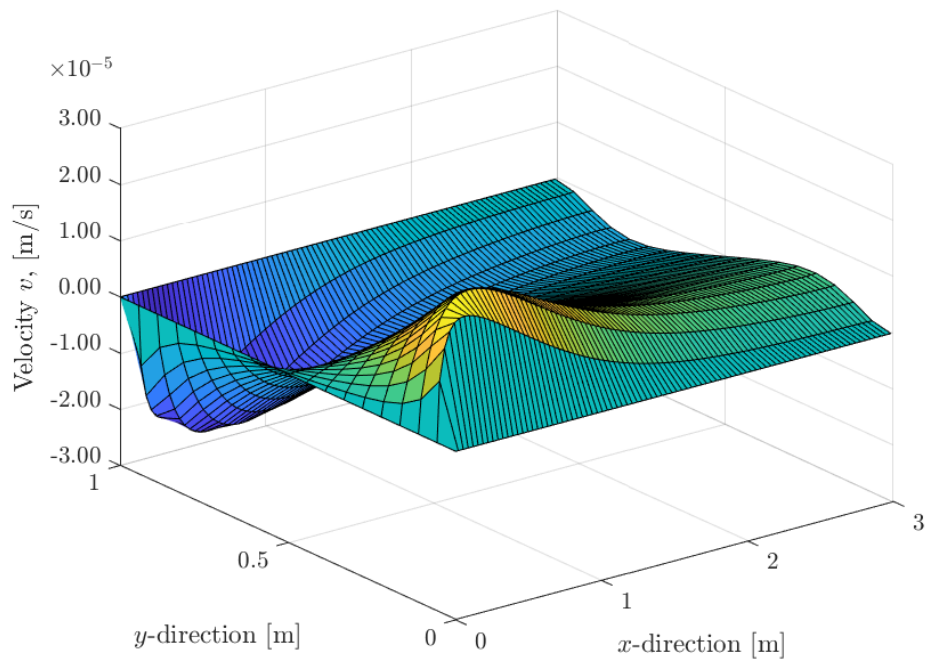


Figure 5.3: v -velocity seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

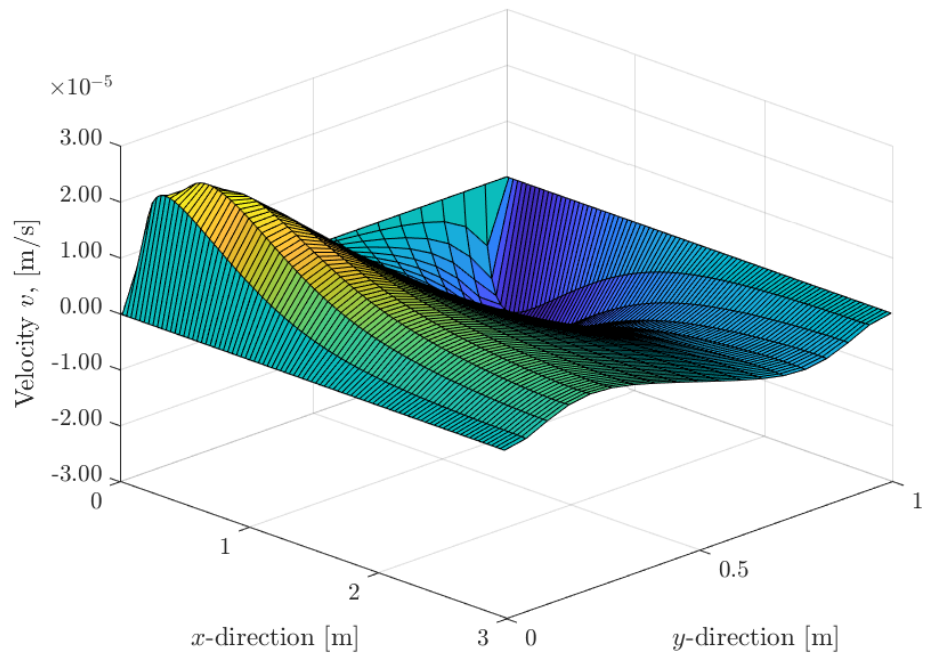


Figure 5.4: v -velocity seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

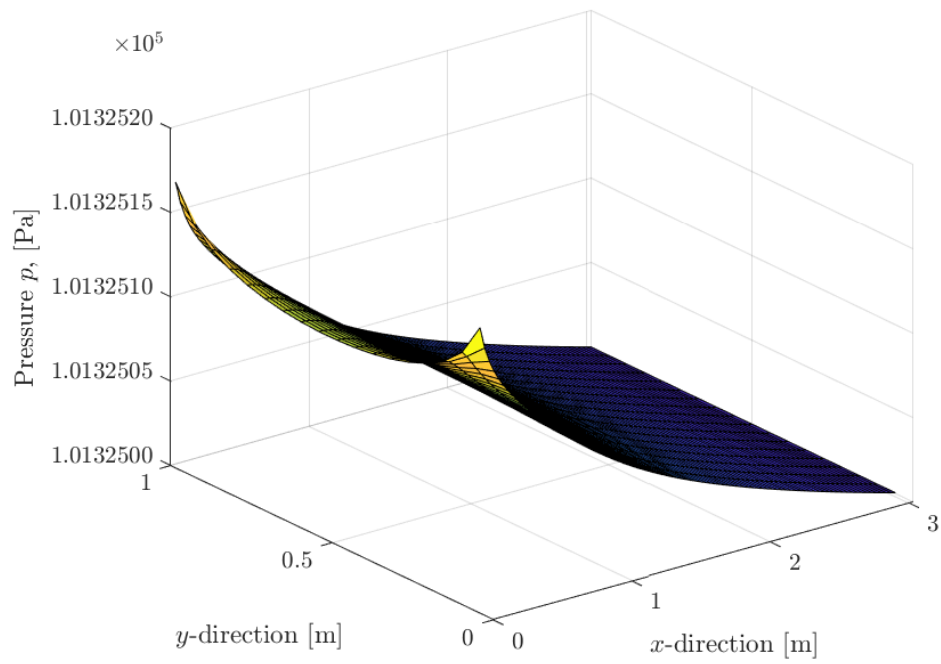


Figure 5.5: Pressure p seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

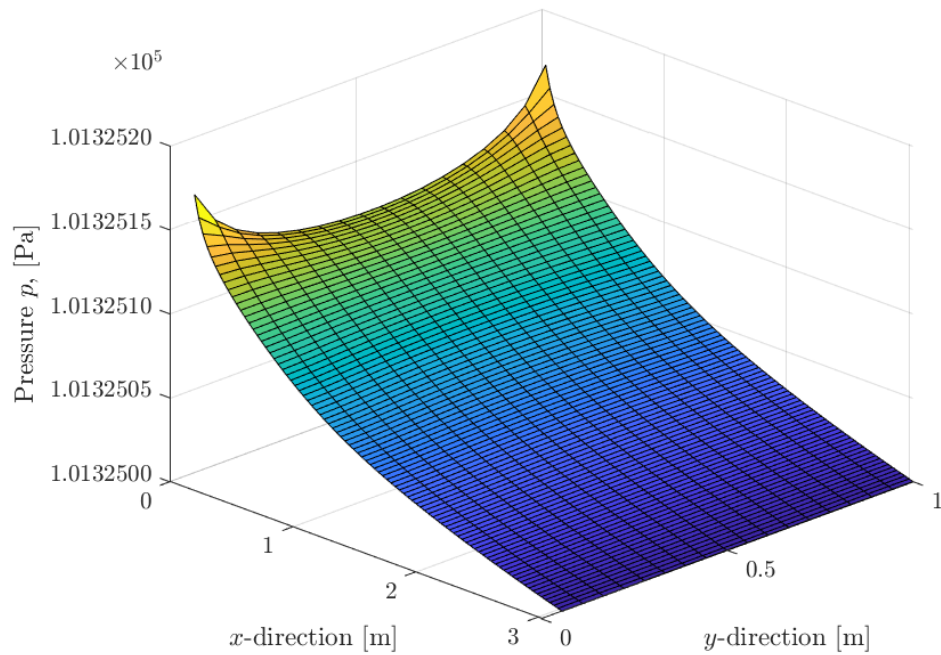


Figure 5.6: Pressure p seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

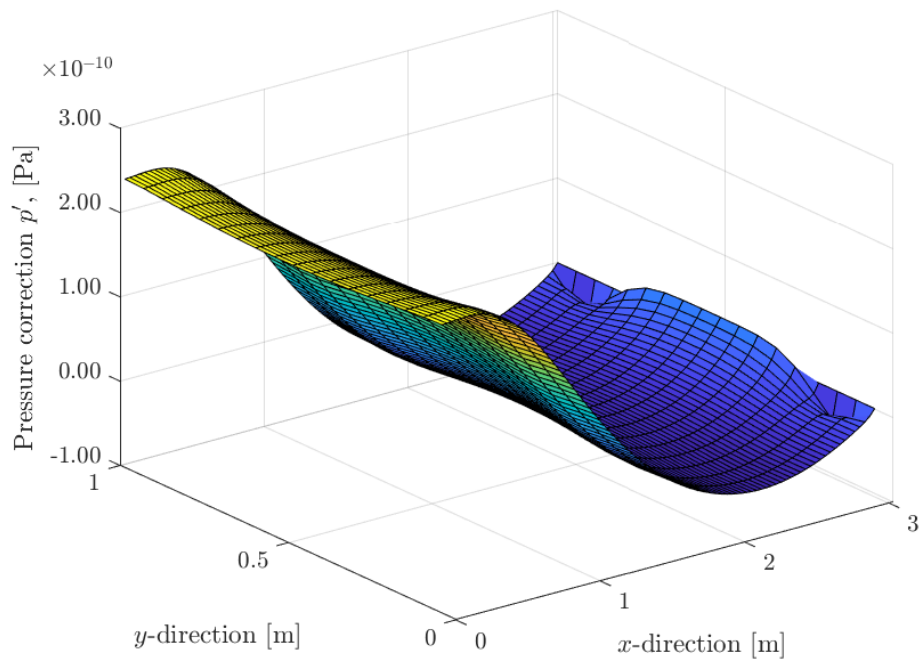


Figure 5.7: Pressure correction p' seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

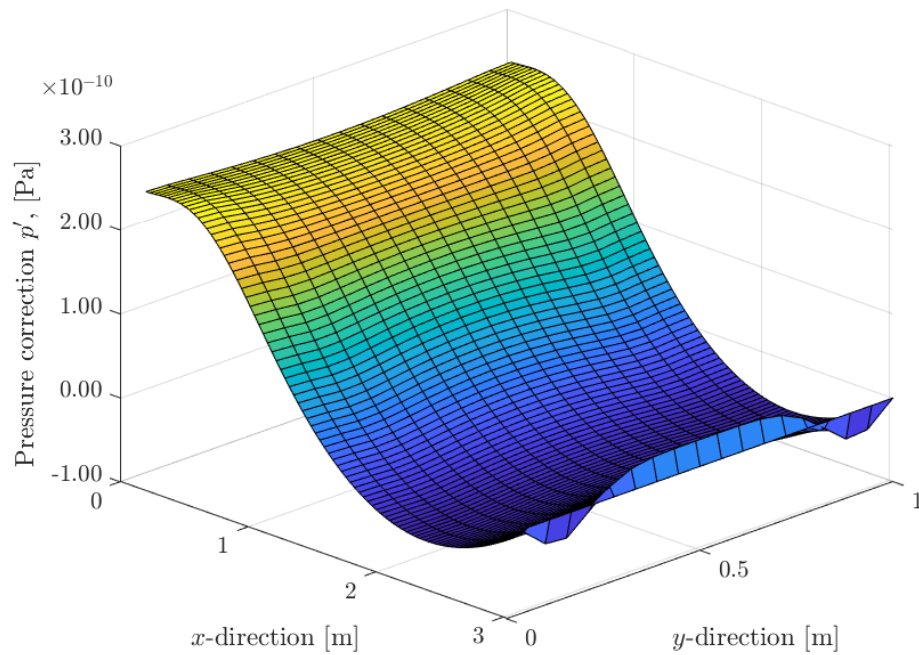


Figure 5.8: Pressure correction p' seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

5.1.2 Long channel

In this section, the surface plots of the fluid flow parameters in a long channel with length $L = 22$ are given. The height of the channel is $h = 1$ like for the inlet section in the BFS domain. 18 times 88 computational points were used for all the plots below and they are shown from both the inlet and the outlet. The Reynolds number is $Re = 560$ for the plots below.

Figure 5.9 shows the u -velocity component profile for the long channel seen from the inlet and figure 5.10 shows the same profile seen from the outlet. The flow is still not fully developed, despite that the profile at the outlet looks to have reached the parabolic profile. A check up of the values in MATLAB reveals that the velocity gradient at the outlet is not zero, and the flow is therefore not fully developed.

Figure 5.11 shows the v -velocity component profile for the long channel seen from the inlet and figure 5.12 shows the same profile seen from the outlet. There is again a flow towards the centre of the channel right after the inlet like for the short channel. This is seen from the increase in the v -velocity near the southern wall and the decrease near the northern wall after the inlet and is due to the friction from the walls. The same amount of computational points were used for the short and the long channel. This means that the inlet section, where the largest changes in the v -velocity occur, is less accurately represented for the extended channel. The v -velocity reaches a value close to zero at approximately 10 m.

Figure 5.13 shows the pressure profile for the long channel seen from the inlet and figure 5.14 shows the same profile seen from the outlet. The scale of the plot is again of low variation, and the pressure is close to constant across the domain like for the short channel.

Figure 5.15 shows the pressure correction for the long channel seen from the inlet and figure 5.16 shows the same profile seen from the outlet. Note that the scale is of order of magnitude 10^{-10} Pa. When converged, the pressure correction should be close to zero across the domain for the continuity equation to be fulfilled. The outlet pressure is known and the pressure correction is therefore zero at the outlet.

Like for the short channel, the pressure correction profile has a small wave-like jump at the points directly before the outlet which is due to the fact that the flow is not fully developed. The magnitude of this is very small and therefore insignificant to the converged solution. Increasing the length of the channel until the flow is fully developed removes this issue. For the height of 1 m, this does not occur until approximately $x = 50$.

For the simulation with $Re = 1120$, the long channel $L = 22$ is visibly not long enough for the flow to be fully developed. The u -velocity profile does not reach a parabolic profile at the outlet, and the v -velocity profile is not completely equal to zero at the outlet.

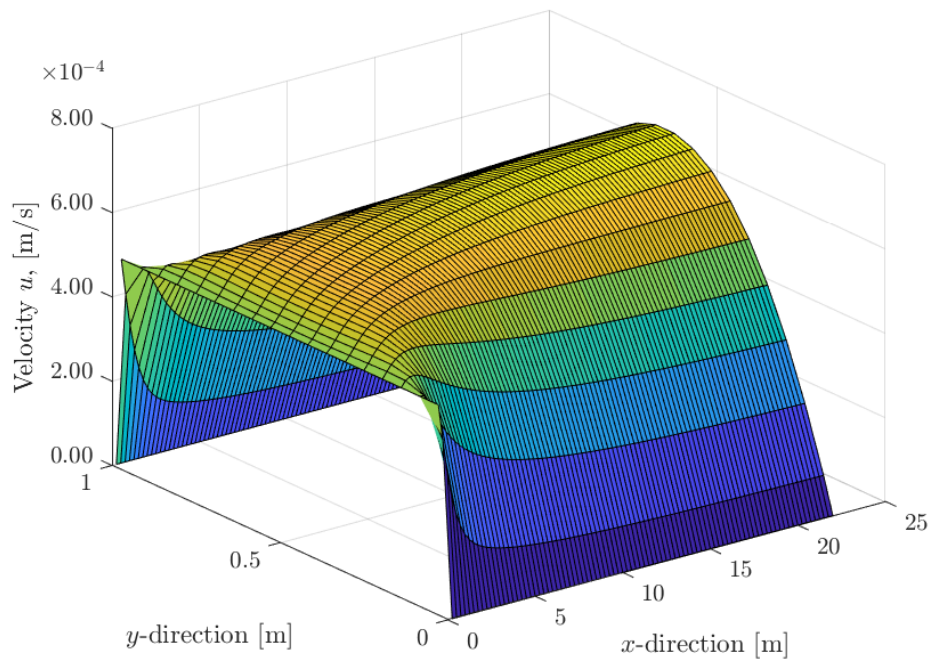


Figure 5.9: u -velocity seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

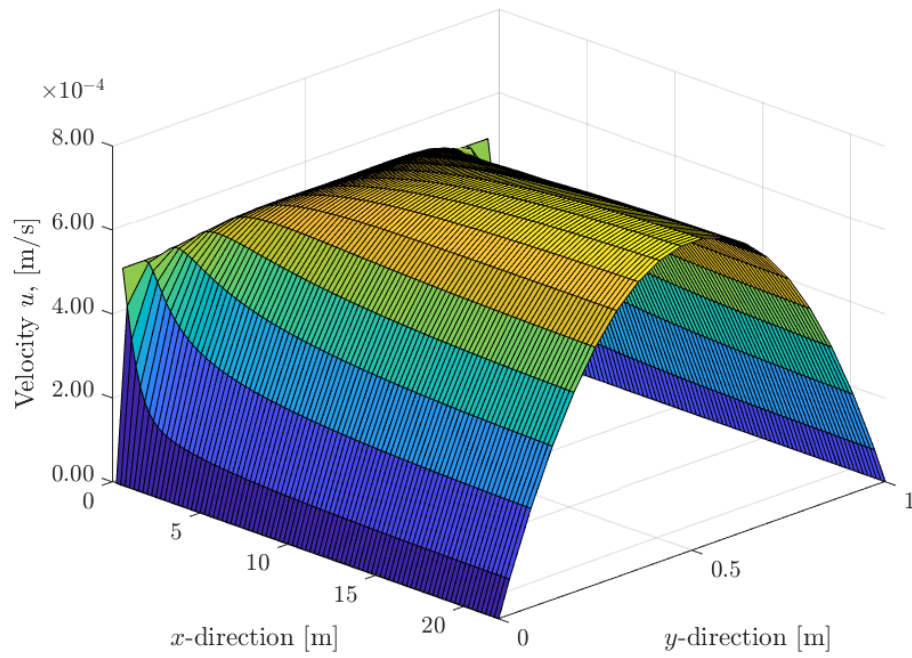


Figure 5.10: u -velocity seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

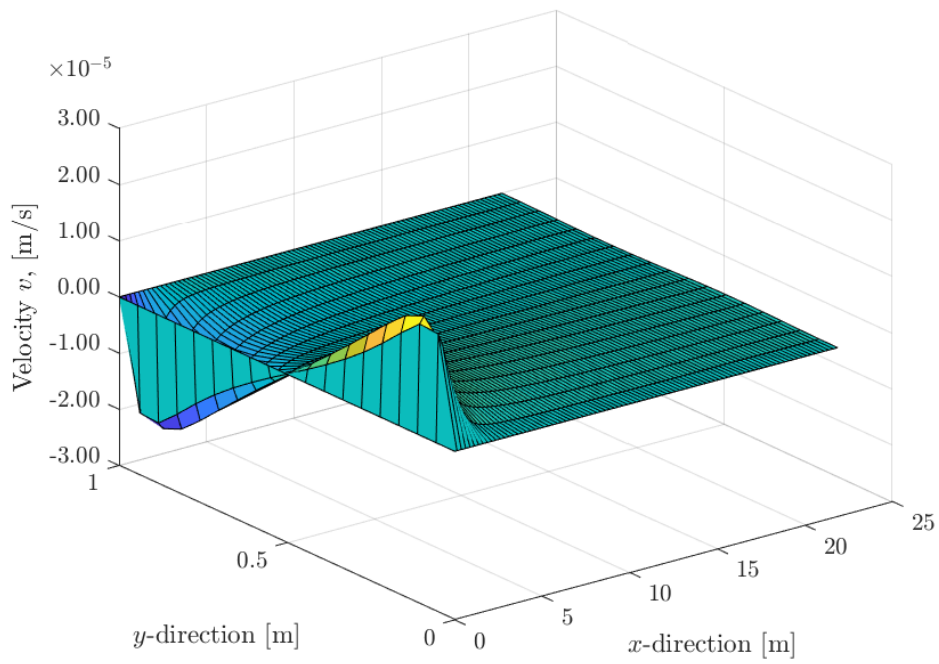


Figure 5.11: v -velocity seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

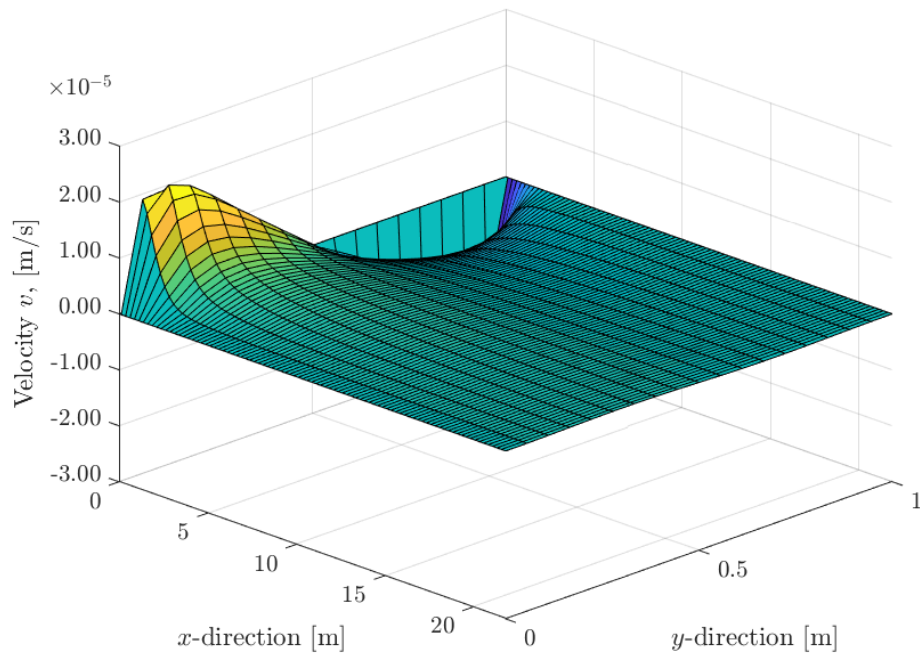


Figure 5.12: v -velocity seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

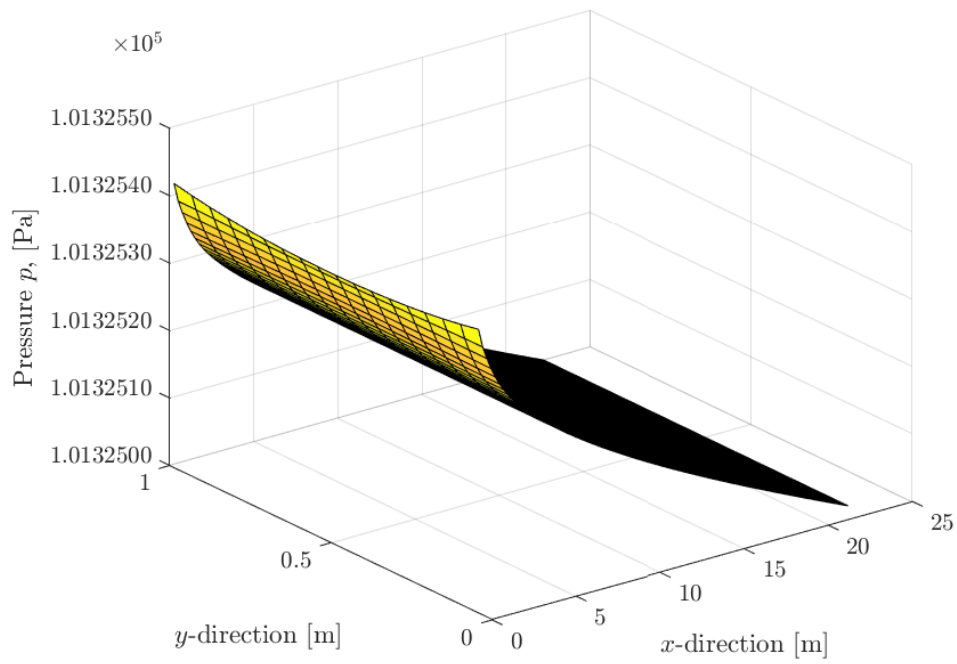


Figure 5.13: Pressure p seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

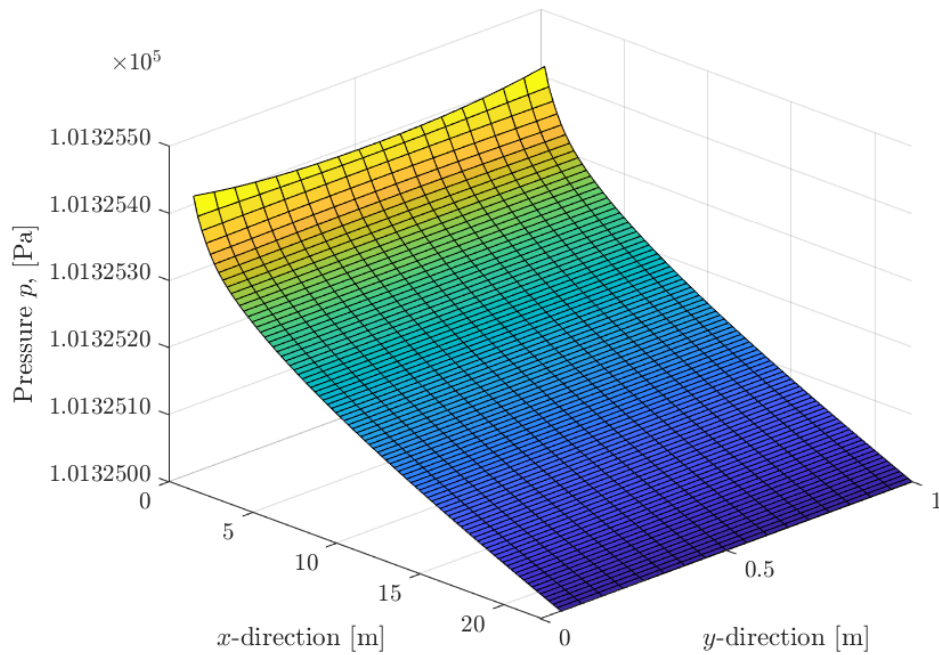


Figure 5.14: Pressure p seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

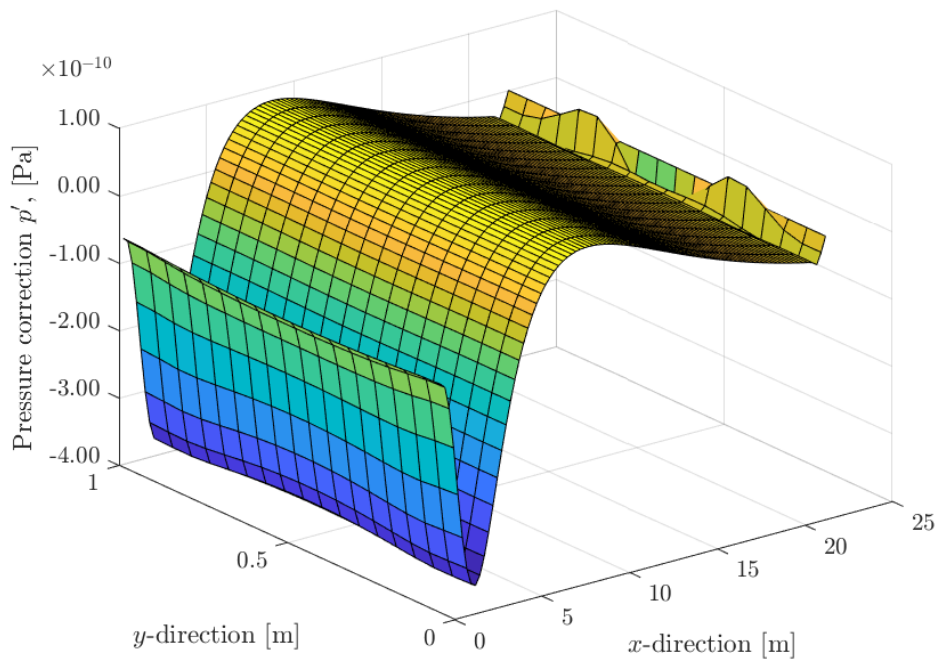


Figure 5.15: Pressure correction p' seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

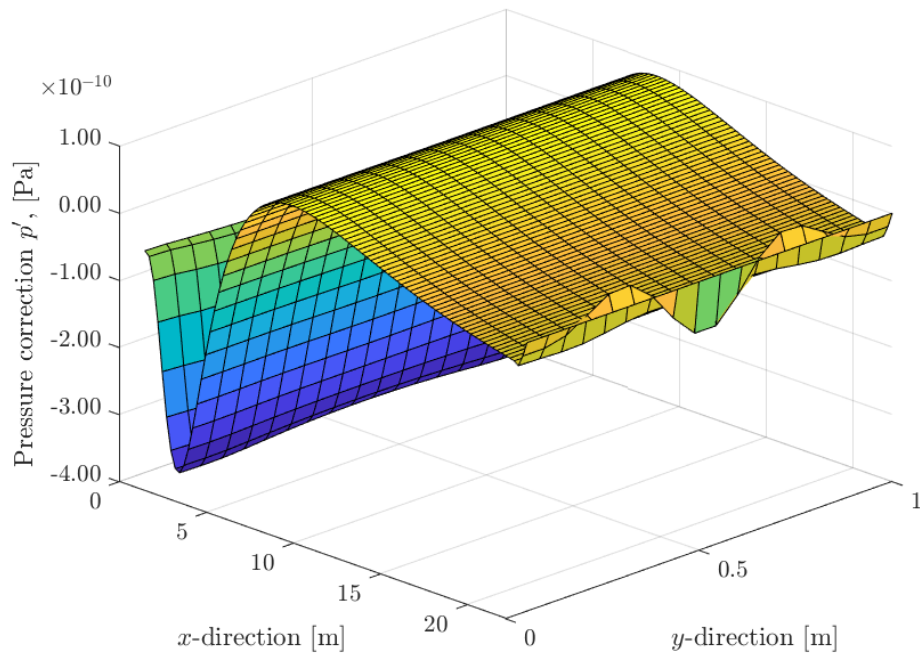


Figure 5.16: Pressure correction p' seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

5.2 Backwards Facing Step Model

In this section, the results for the flow over the backwards facing step are given. The two domains shown in figures 1.2 and 1.3 were used, the first was used to develop the model and the second was used to compare the result with Biswas et al. [4] for different Reynolds numbers. The results for the domain in figure 1.2 are shown in section 5.2.1 and the results for the domain in figure 1.3 are shown in section 5.2.2.

5.2.1 Constant Inlet Velocity

In this section, the results for the flow over the backwards facing step domain as used by Melaaen [3] are given. The domain has a total length of $L = 22$ m which corresponds to the length of the long channel as shown in section 5.1.2. All the dimensions of the domain are given by figure 1.2 and in table 4.1. The MATLAB code `channel_BFS.m` was used to obtain the results, and is given in appendix E. 18 times 88 computational points with a total of 1512 scalar nodes were used for all the plots below and they are shown from both the inlet and the outlet. This resolution is around the highest possible resolution for the model with the current settings without the model stopping due to singularity in one or more of the coefficient matrices.

Table 5.2 shows the two different inlet u -velocities used as given in section 4.4 and the corresponding number of iterations and computational time before convergence was reached. The under-relaxation factors were reduced to half for $Re = 560$ in comparison to $Re = 1120$ as described in section 4.2.

u_{in}	Re	Iterations	Time
$1 \cdot 10^{-3}$	1120	10261	1 h 35 min
$5 \cdot 10^{-4}$	560	12286	1 h 44 min

Table 5.2: Number of iterations and convergence time for the backwards facing step model with a constant inlet velocity.

Below the plotted results for $Re = 560$ are shown. The hydraulic diameter D_{hyd} is defined as in equation (2.1.9), and is equal to h .

5.2.1.1 Surface Plots

Figure 5.17 shows the u -velocity component profile for the flow over the backwards facing step seen from the inlet and figure 5.18 shows the same profile seen from the outlet. As can be seen, the profile is fully developed at around $x = 8$ as the outlet profile is parabolic and the profile does not change further. The recirculation zone after the step is visible, but is easier to see from the velocity vector plots given in section 5.2.1.2 where the u - and v -velocity components are combined.

Figure 5.19 shows the v -velocity component profile for the flow over the backwards facing step seen from the inlet and figure 5.20 shows the same profile seen from the outlet. As can be seen, the profile at the inlet follows the pattern from the flow in the straight channel as presented in section 5.1, where there is a preliminary flow towards the centre of the channel. The flow is fully developed as the outlet profile is zero.

Figure 5.21 shows the pressure profile for the flow over the backwards facing step seen from the inlet and figure 5.22 shows the same profile seen from the outlet. Like for the

two dimensional straight channel plots the scale is of low variation, and the pressure is close to constant across the domain.

Figure 5.23 shows the pressure correction for the flow over the backwards facing step seen from the inlet, and figure 5.24 shows the same profile seen from the outlet. Unlike the result from the two dimensional straight channel, the pressure correction is equal to zero towards the outlet because the flow is fully developed. The same outlet boundary condition and implementation was used in all cases.

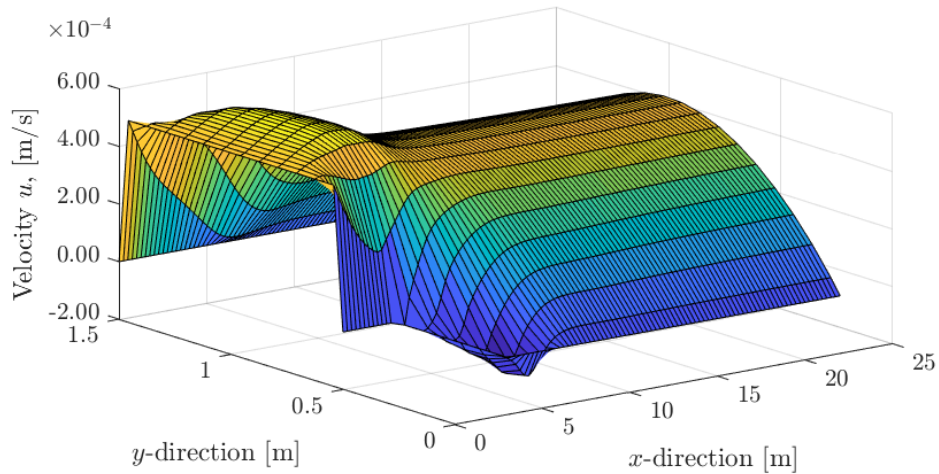


Figure 5.17: u -velocity seen from the inlet for the backwards facing step model.

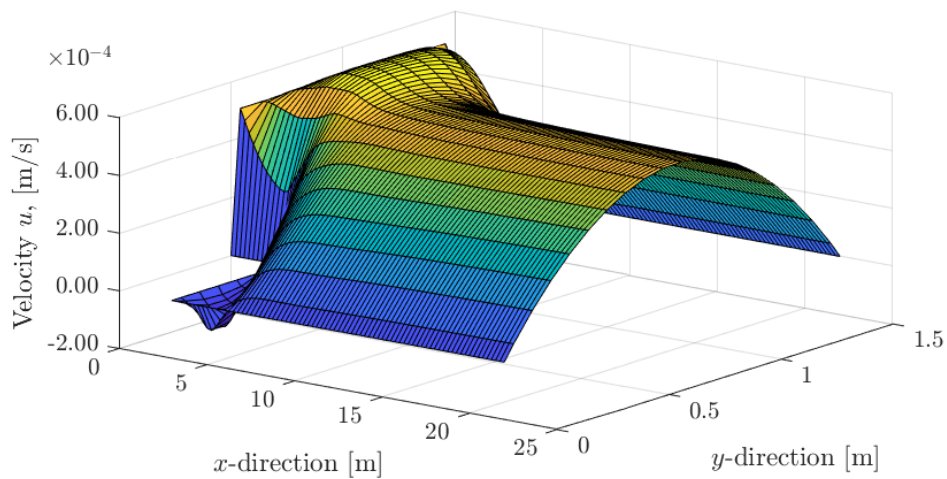


Figure 5.18: u -velocity seen from the outlet for the backwards facing step model.

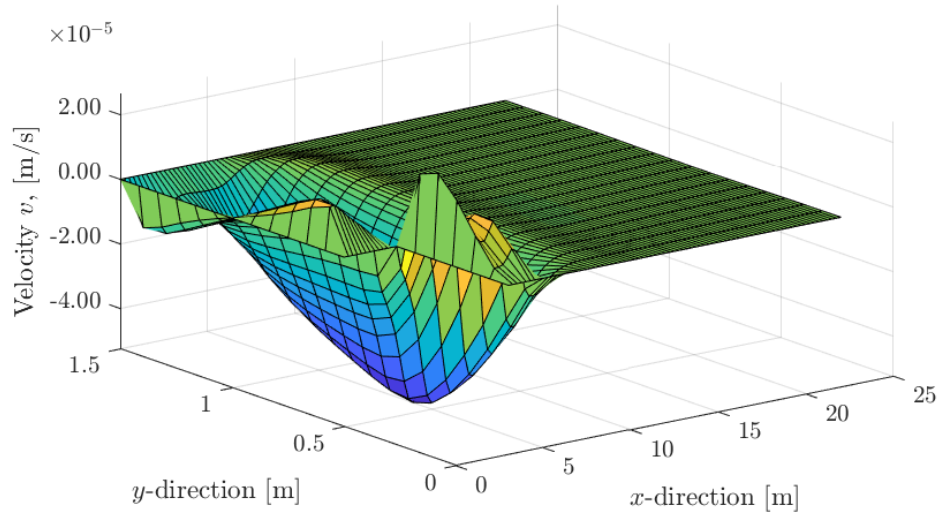


Figure 5.19: v -velocity seen from the inlet for the backwards facing step model.

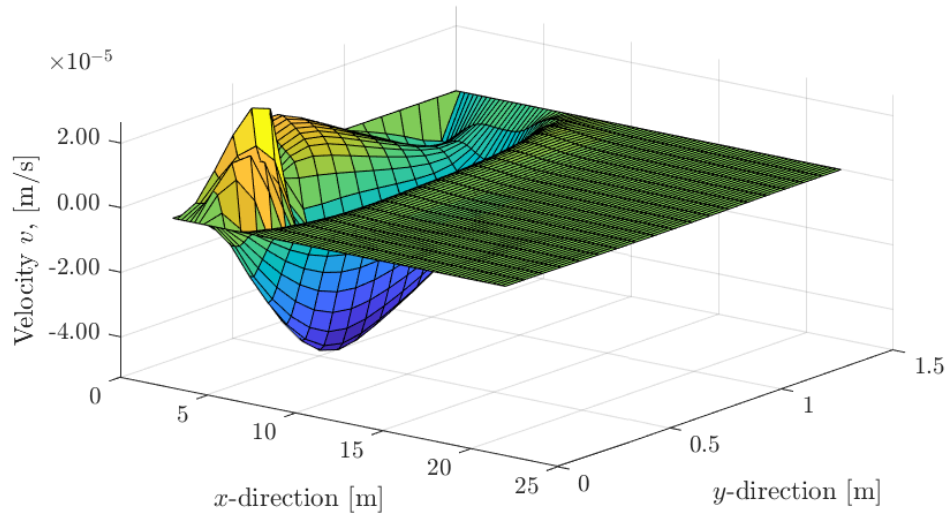


Figure 5.20: v -velocity seen from the outlet for the backwards facing step model.

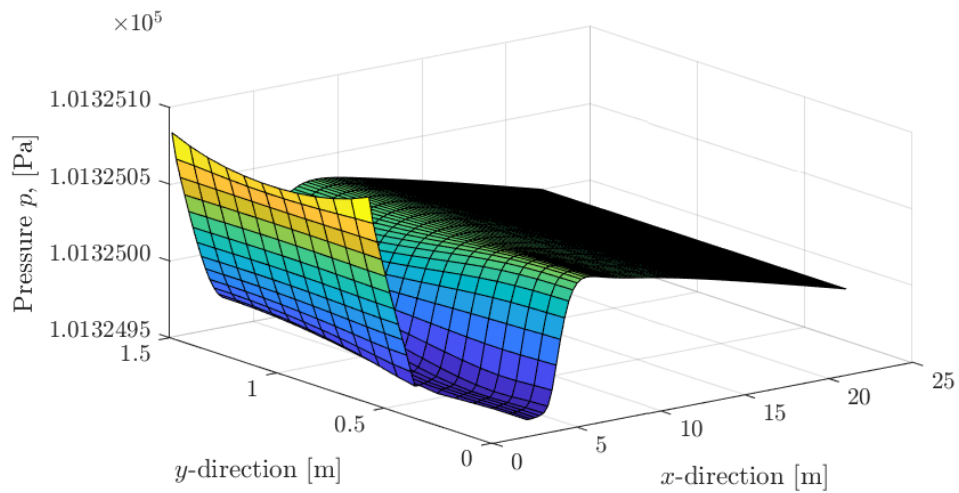


Figure 5.21: Pressure p seen from the inlet for the backwards facing step model.

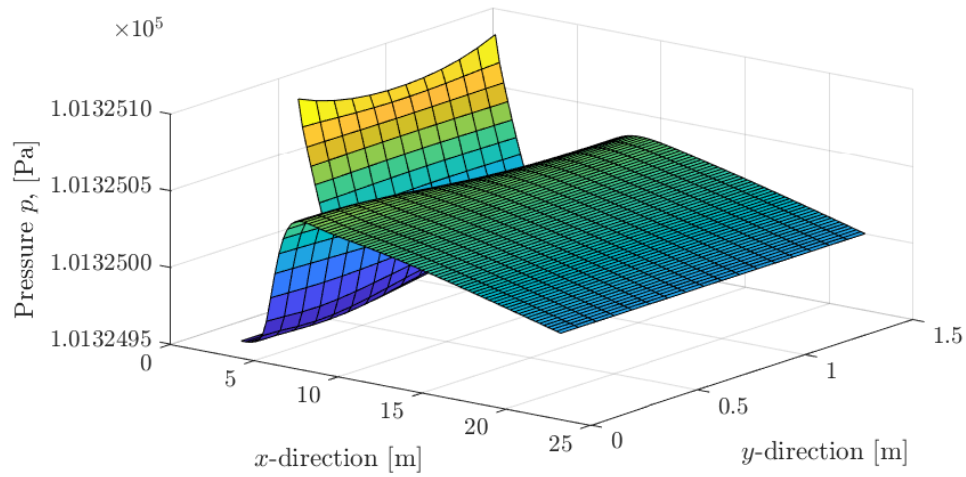


Figure 5.22: Pressure p seen from the outlet for the backwards facing step model.

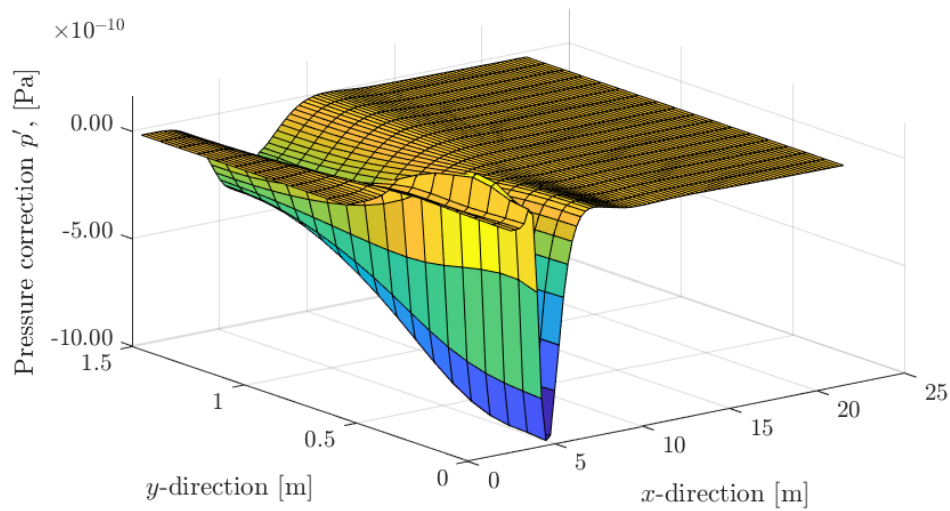


Figure 5.23: Pressure correction p' seen from the inlet for the backwards facing step model.

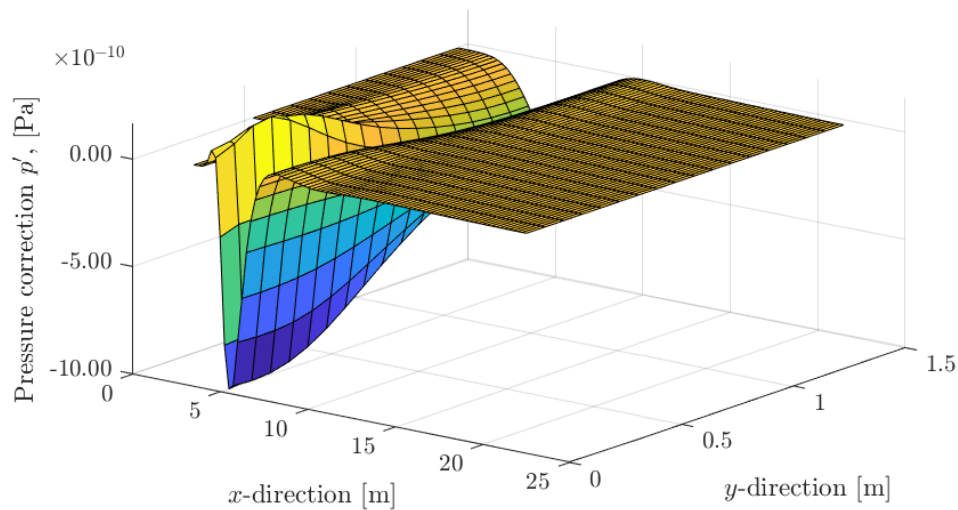


Figure 5.24: Pressure correction p' seen from the outlet for the backwards facing step model.

5.2.1.2 Velocity Vector Plots

In the velocity vector plots shown in this section, the velocities are represented as arrows. The background color signifies the value of the velocity at each point. In all the velocity vector plots presented in this thesis, dark blue represents the lowest value and yellow is the highest possible value as seen in figure 5.25. The actual value of the velocities varies for all the plots. The arrows show the direction of the velocity in each point, but the magnitude is also reflected in the length of each arrow. The arrows are scaled relatively, which means that the highest velocity in the domain is assigned a specific arrow length and all the other arrow lengths are scaled accordingly. The points at which each velocity is calculated are located at the beginning of the stem of each arrow.



Figure 5.25: Color scale used in the velocity vector plots.

Figure 5.26 shows the velocity vector plot for the combined u and v -velocity for the flow over the backwards facing step.

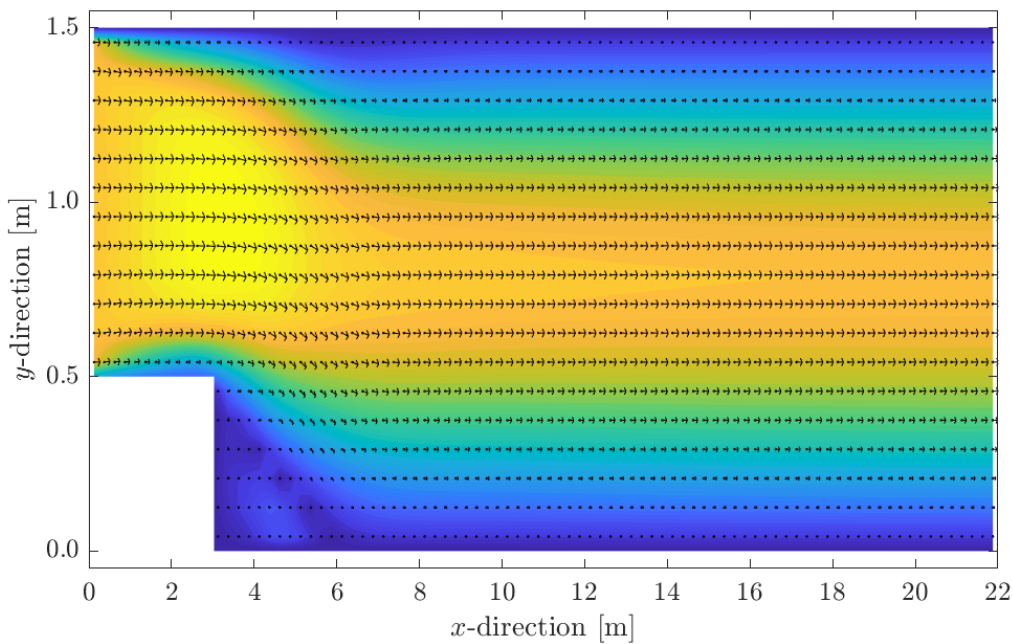


Figure 5.26: Velocity vector plot for the backwards facing step model.

Figure 5.27 shows a zoomed in version of the same velocity plot as in figure 5.26. The plot is zoomed in to show the flow from the steps to three times the width of the step. The length of the arrows is scaled to 3 times the length of the arrows in figure 5.26. The recirculation zone is visible. Since the resolution is quite low, it is hard to determine where the flow separation due to the recirculation zone ends, but it is clear that it is somewhere at around 6 m. This is equivalent to around 12 times the step height.

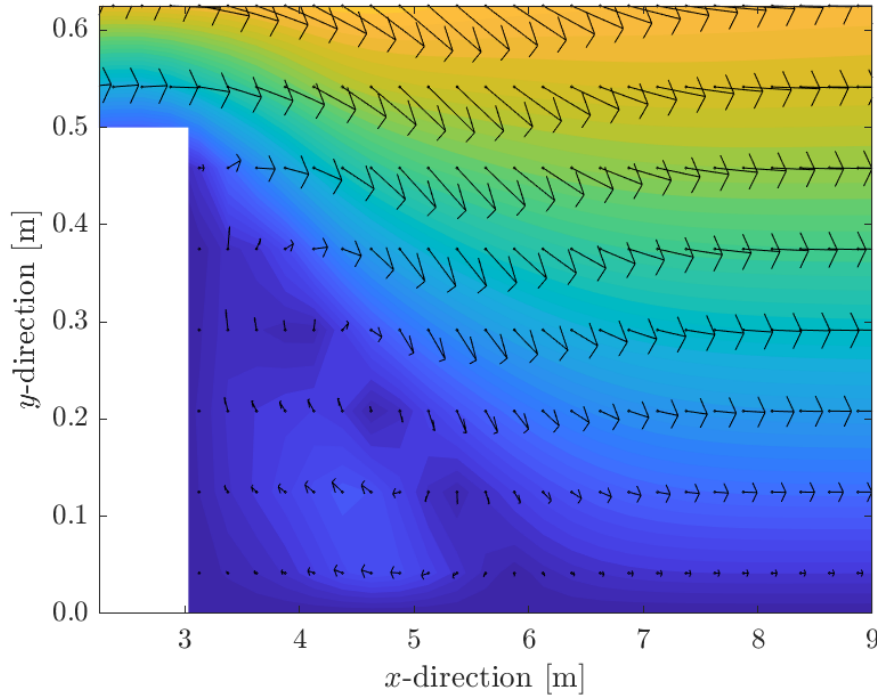


Figure 5.27: Velocity vector plot for the backwards facing step model zoomed in on the recirculation zone after the step.

5.2.2 Parabolic Inlet Velocity Profile

In this section, the results for the flow over the backwards facing step domain as used by Biswas et al. [4] are given for a variety of low Reynolds numbers. The domain has different dimensions from the domain used to obtain the results in section 5.2.1, all dimensions are given by figure 1.3 and in table 4.1. The total length of this domain is $L = 35$.

A parabolic profile was used at the inlet for the u -velocity instead of the constant inlet velocity used in section 5.2.1. The MATLAB code `channel_BFS_parabolic.m` was used to obtain the results, and is given in appendix E. 20 times 70 computational points with a total of 1300 scalar nodes were used for all the simulations. The results were obtained for a variety of Reynolds numbers and will be compared in chapter 6 to the results found by Biswas et al. [4].

Table 5.3 shows the different Reynolds numbers used for the flow over the backwards facing step with a parabolic inlet velocity profile as specified in table 4.7. The number of iterations and the convergence times for the model are also shown. Biswas et al. [4] provides results for Reynolds numbers between 0.0001 and 100, and the higher Reynolds numbers were added to see how the model behaves. For the two higher Reynolds numbers, the under-relaxation factors were halved compared to the lower Reynolds numbers to achieve convergence. The hydraulic diameter D_{hyd} is defined as $2h$ like by Biswas et al. [4]. Still h is used as a scaling parameter for all the spacial dimensions, which means that the Reynolds numbers in this section are equivalent the Reynolds numbers in section 5.2.1.

Re [-]	Iterations [-]	Time
0.0001	1879	11 min
0.1	1879	13 min
1	1879	13 min
10	2033	13 min
50	2599	18 min
100	3284	22 min
200*	10280	63 min
400*	18726	117 min

Table 5.3: Number of iterations and convergence time for the backwards facing step model with parabolic inlet profile for a range of Reynolds numbers. * Under-relaxation factors were halved.

5.2.2.1 Velocity Vector Plots

In this section, the velocity vector plots for the set of Reynolds numbers as shown in table 5.3 are given. In all the velocity vector plots dark blue represents the lowest value of the velocity in the domain for the current settings and yellow is the highest possible value for the velocity (see figure 5.25). The whole domain is shown in all the plots, which makes it difficult to see the recirculation zones after the step in detail. Zoomed in plots of the recirculation zones for the different Reynolds numbers are compared in section 5.2.2.2.

Figure 5.28 shows the velocity vector plot for the combined u and v -velocity for the flow over the backwards facing step with the Reynolds number $Re = 0.0001$. There is no visible recirculation zone.

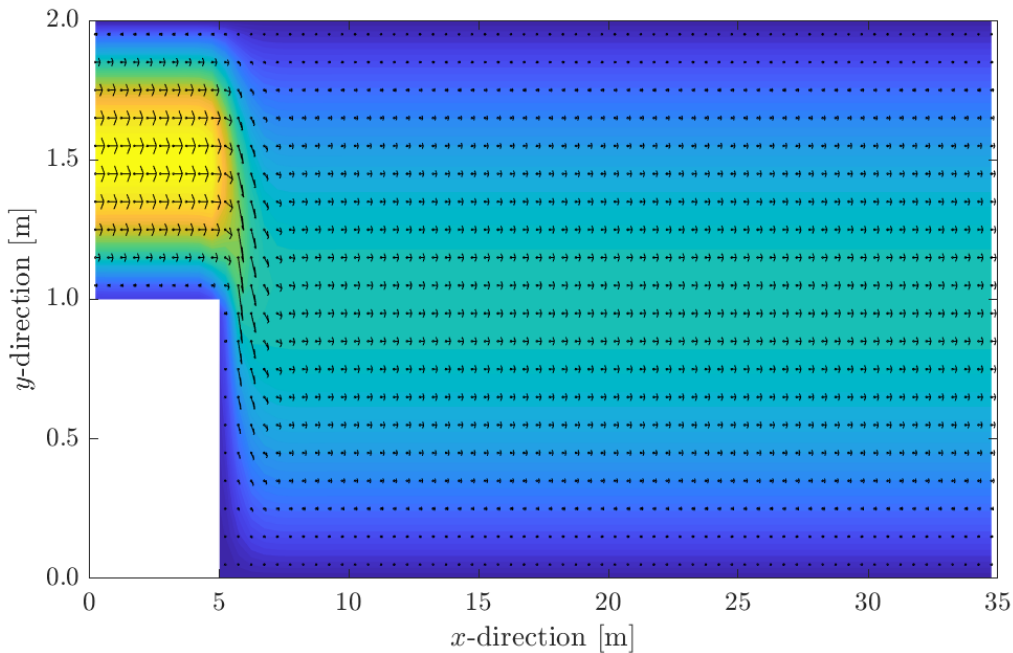


Figure 5.28: Velocity vector plot for the backwards facing step model with $Re = 0.0001$.

Figure 5.29 shows the velocity vector plot with Reynolds number $Re = 0.1$. There is still no visible recirculation zone.

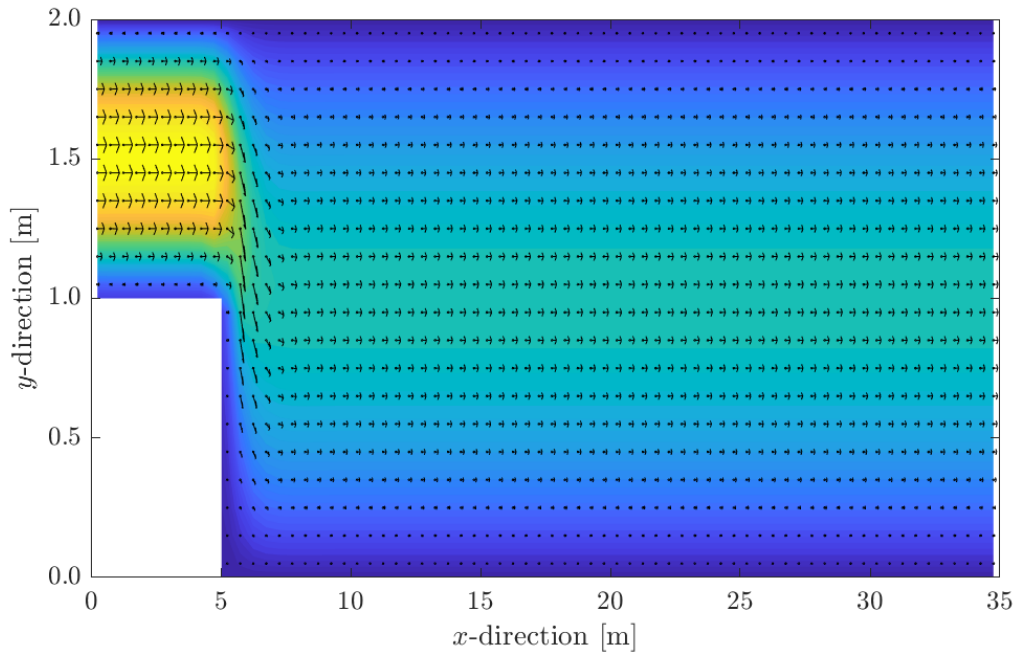


Figure 5.29: Velocity vector plot for the backwards facing step model with $Re = 0.1$.

Figure 5.30 shows the velocity vector plot with Reynolds number $Re = 1$. There is no visible recirculation zone. Figure 5.31 shows the velocity vector plot with Reynolds number $Re = 10$. The recirculation zone is not prominent for the Reynolds numbers between 0.0001 and 10, and the velocity plots look very similar. Figure 5.32 shows the velocity vector plot with Reynolds number $Re = 50$. The recirculation zone is starting to develop after the step. Figure 5.33 shows the velocity vector with Reynolds number $Re = 100$. The recirculation zone is visible. Figure 5.34 shows the velocity vector with $Re = 200$. The recirculation zone is now easy to spot. Figure 5.35 shows the velocity vector with $Re = 400$. The recirculation zone is visible, and a secondary recirculation zone is appearing at the northern wall after the first zone next to the step. This zone was observed by Armaly et al. [7] for Reynolds numbers larger than 400.

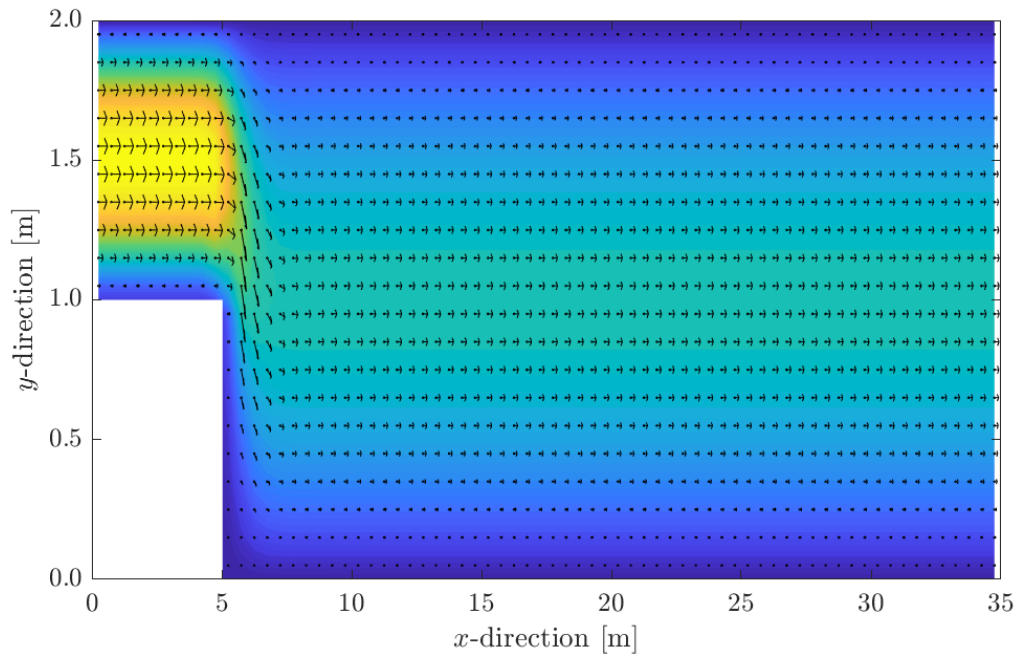


Figure 5.30: Velocity vector plot for the backwards facing step model with $Re = 1$.

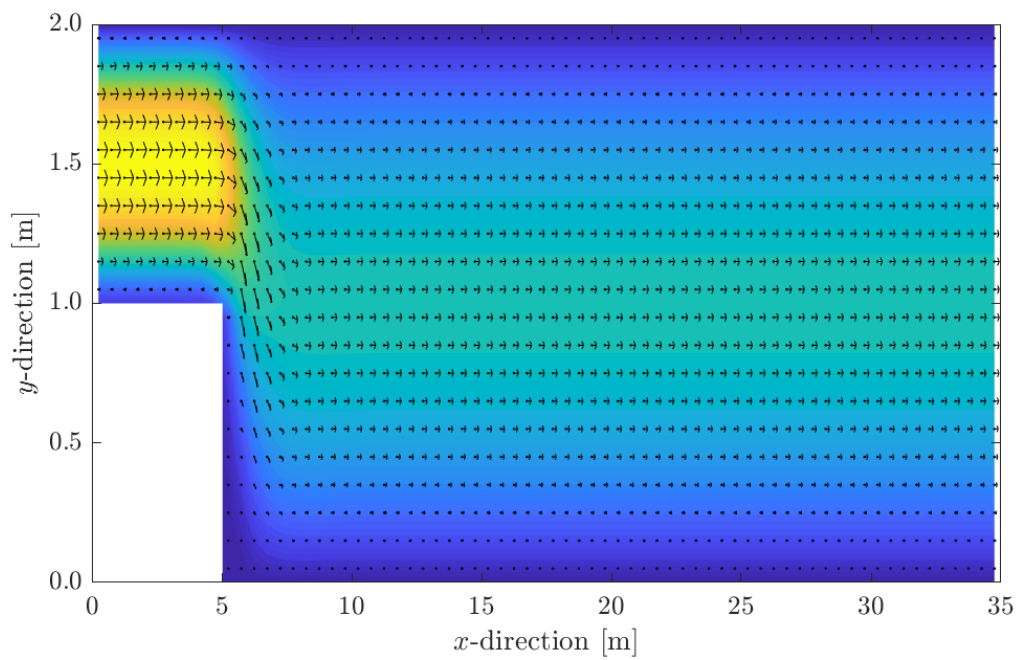


Figure 5.31: Velocity vector plot for the backwards facing step model with $Re = 10$.

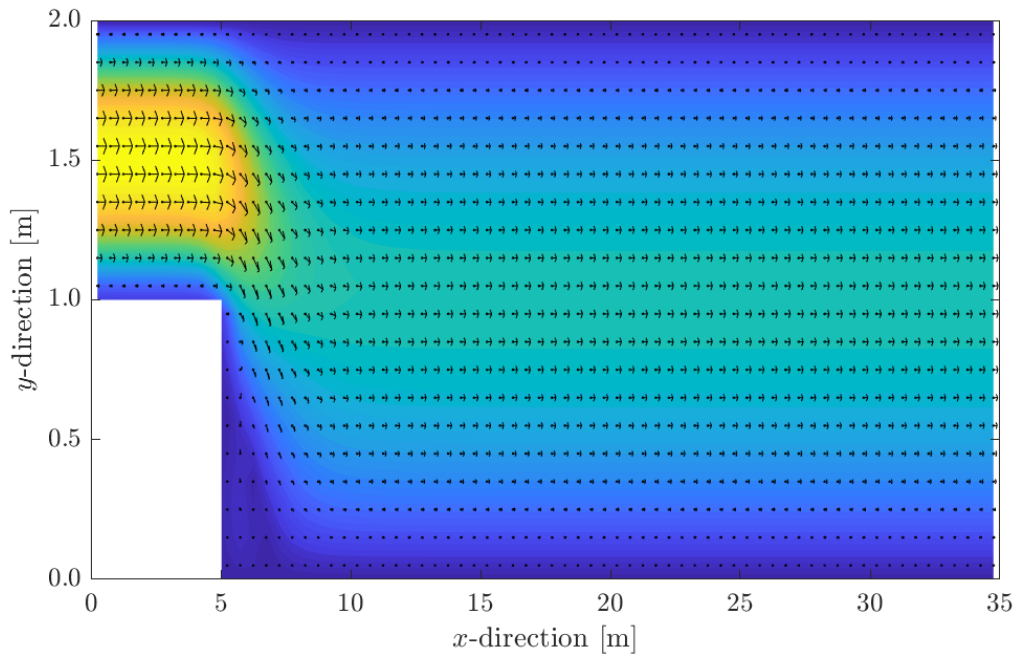


Figure 5.32: Velocity vector plot for the backwards facing step model with $Re = 50$.

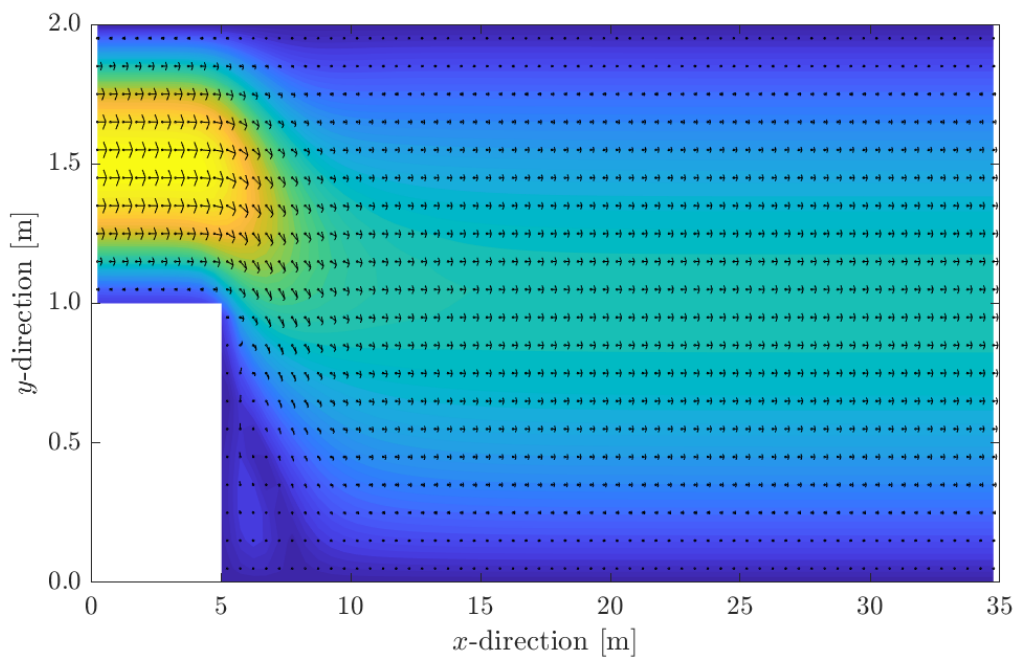


Figure 5.33: Velocity vector plot for the backwards facing step model with $Re = 100$.

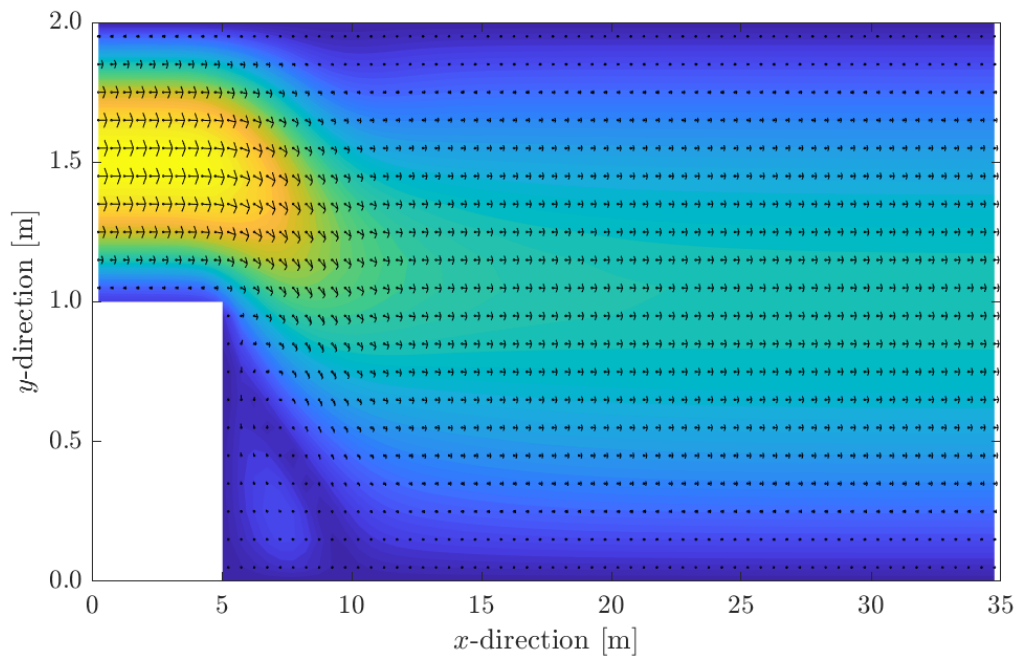


Figure 5.34: Velocity vector plot for the backwards facing step model with $Re = 200$.

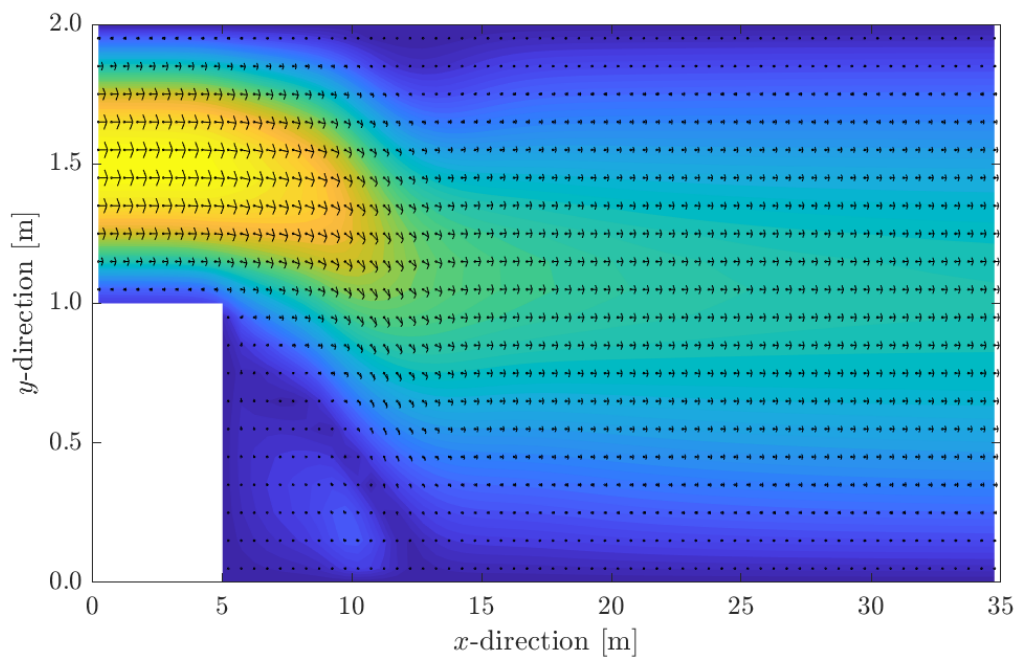


Figure 5.35: Velocity vector plot for the backwards facing step model with $Re = 400$.

5.2.2.2 Comparison of Recirculation Zone

Figure 5.36 show zoomed in versions of the velocity vector plots given in section 5.2.2.1. The plots show the recirculation zones after the backwards facing step for the same set of low Reynolds numbers as used by Biswas et al. [4]. The section shown is the flow between $x = 5$ to five times the step height at $x = 10$. The length of the arrows is scaled 3 times in comparison to the arrows in figures 5.28-5.35.

Figure 5.37 show the same zoomed in versions of the velocity vector plots as in figure 5.36 with the addition of two higher Reynolds numbers of 200 and 400 as given in table 4.7. The section shown is the flow between the step at $x = 5$ to 7.5 times the step height at $x = 12.5$. The length of the arrows is scaled 3 times in comparison to the arrows in figures 5.28-5.35.

As can be seen from figures 5.36 and 5.37, there is seemingly a slight flow out from the wall of the step to the very left of the figure. This is especially apparent from the northernmost point east of the step, which can also be seen in figure 5.27. This behaviour is not physical, as there should be no flow through the wall. The point in question is not located directly at the wall and a nonzero velocity value here would be feasible. It appears that the velocity is not affected by the v -velocity component at all in any of the cases. This may mean that there is an error in the implementation of the boundary condition at this western wall. Although there is seemingly a slight velocity out from the wall here, it should not be a large problem, since the magnitude of the velocity is very small compared to the rest of the channel.

Figure 5.38 shows the flow plots from Biswas et al. [4] for comparison to the results for the Reynolds number study from Biswas et al. [4] who also used the Finite Volume method for the results and the SIMPLE algorithm for obtaining the pressure. The whole height of the domain are shown, but in x -direction the plots are cropped to include 1 m of the inlet section before the expansion and 3 meters after the expansion. The origin of the coordinate system is located at the corner of the backwards facing step, so that $x = 3$ in figure 5.38 corresponds to $x = 8$ in figure 5.36.

Due to the coarseness of the grid used in the simulations in this thesis, the recirculation in the corner for the Reynolds numbers lower than 10 are not visible in figure 5.36. For $Re = 50$ and $Re = 100$ the recirculation can be seen, and the reattachment lengths are in accordance with the results in figure 5.38. The reattachment length is the length of the recirculation zone from the step and until the end of the zone, where the flow no longer curves back towards the step at the southern wall. The agreement of the results are discussed further in chapter 6.

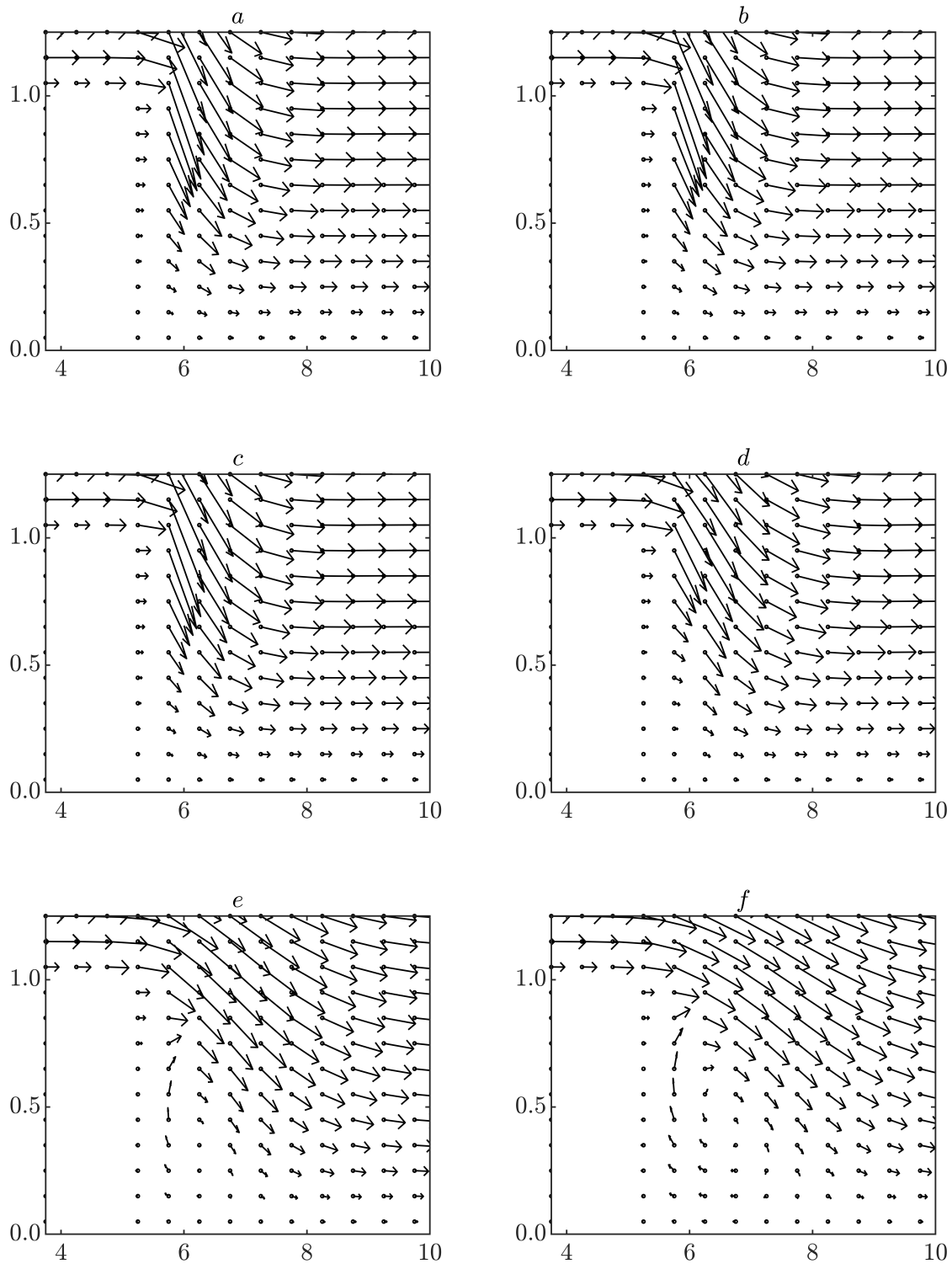


Figure 5.36: Comparison of the recirculation zone over the backwards facing step for different Reynolds numbers. *a*) $Re = 0.0001$, *b*) $Re = 0.1$, *c*) $Re = 1$, *d*) $Re = 10$, *e*) $Re = 50$ and *f*) $Re = 100$.

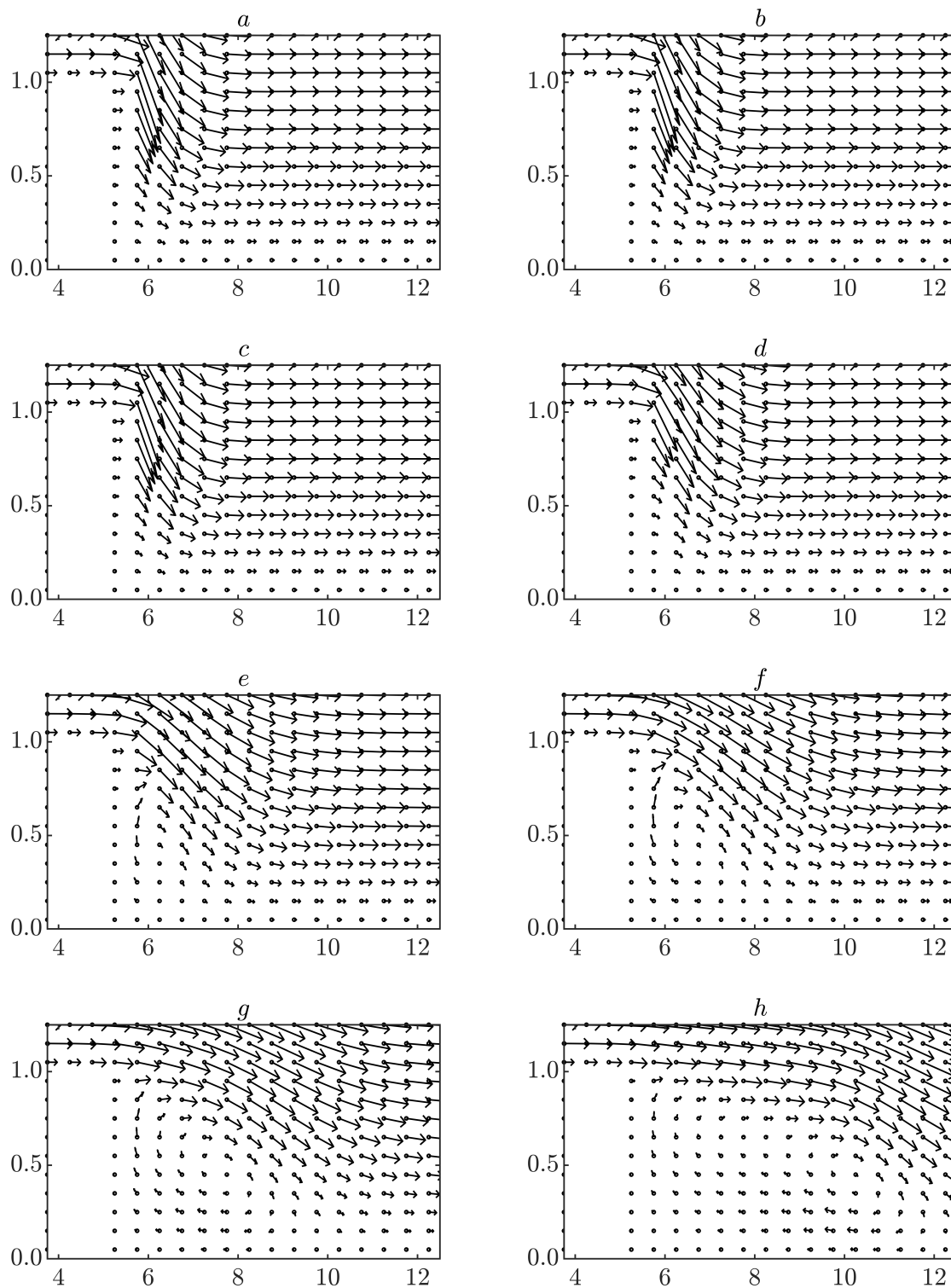


Figure 5.37: Comparison of the recirculation zone over the backwards facing step for different Reynolds numbers. *a)* $Re = 0.0001$, *b)* $Re = 0.1$, *c)* $Re = 1$, *d)* $Re = 10$, *e)* $Re = 50$, *f)* $Re = 100$, *g)* $Re = 200$ and *h)* $Re = 400$.

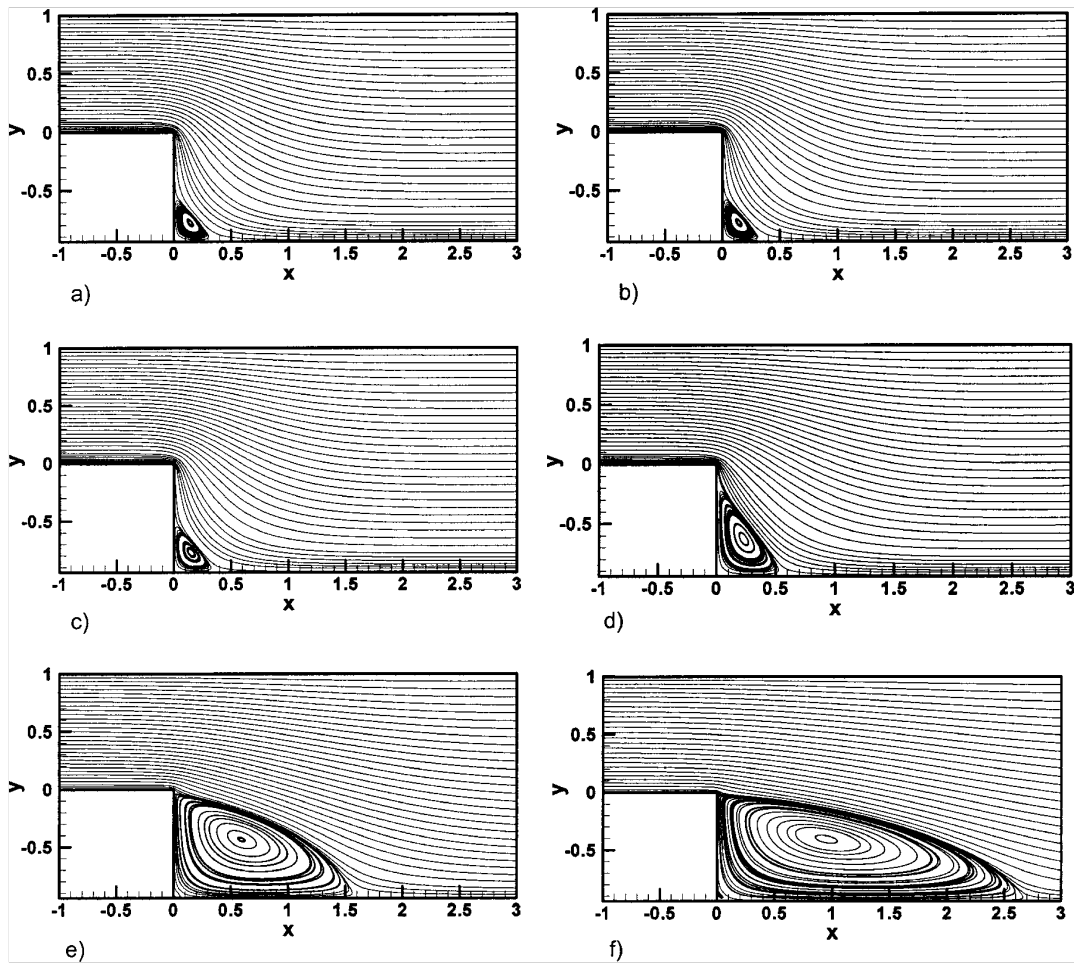


Figure 5.38: Flow over the step as found by Biswas et al. [4]. a) $Re = 0.0001$, b) $Re = 0.1$, c) $Re = 1$, d) $Re = 10$, e) $Re = 50$, f) $Re = 100$.

6

Discussion

In this section, the results as presented in chapter 5 are further discussed, the accuracy of the models that were developed during the work with this thesis are assessed and the improvements from the models developed in the previous project on the topic are discussed.

6.1 Straight Channel Model

The two dimensional model yields good results that fit the expectations. The profiles are symmetrical around the centre of the channel due to the lack of gravity in the modelled dimensions. There are some minor inaccuracies at the outlet when the flow is not fully developed, which is visible from the pressure correction profile. The boundary conditions applied are tailored to fully developed flow, so for this domain a longer channel is needed to obtain the correct results at the outlet.

6.2 Backwards Facing Step Model

In this section, the results from the backwards facing step models are discussed further, and the differences between the results and the findings by Biswas et al. [4] are discussed.

The flow becomes fully developed in all the simulations that were performed. In the simulations with a constant inlet velocity, the Reynolds number is quite high and close to the turbulent transition region, depending on the definition of this region. According to the definition in equation (2.1.11), the Reynolds numbers are well within the laminar range, but according to the definition in equation (2.1.12) only the four lowest Reynolds numbers in section 5.2.2 are in the laminar range. This may mean that the results in section 5.2.1 are less accurate than the results in section 5.2.2, which are obtained for a range of low Reynolds numbers.

Armaly et al. [7] and Biswas et al. [4] state that the flow over the backwards facing step is of two dimensional behaviour for Reynolds numbers below 400. In the first domain, the Reynolds numbers were chosen to be higher than this, which means that they might be inaccurate due to the lack of impact from the third dimension. The results from the second domain are all obtained for Reynolds numbers lower than and including 400 and should therefore be more accurate. It can be seen from the plots in section 5.2.1 that the recirculation zone is not as smoothly represented as for the plots in section 5.2.2.

6.2.1 Convergence

The convergence times for the models can be seen from tables 5.2 and 5.3. In the first simulations with a constant inlet velocity, the computational time increases from $Re = 1120$ to $Re = 560$. The under-relaxation factors had to be halved for the simulations with $Re = 560$ to converge, which is probably the main reason for this. Another possible reason could be that because the recirculation zone is smaller for the lower Reynolds number, but less computational nodes are available in the area of the zone. This means that the model is struggling to determine the properties at each point because there are too few discrete points in the domain to accurately describe the behaviour.

In the second set of simulations, with the parabolic inlet profile, the convergence times are significantly lower for the lowest Reynolds numbers than in the first domain. At $Re = 200$ and 400 , the under-relaxation factors were halved to achieve convergence, which partly can explain why the convergence times peak at these Reynolds numbers. Looking at the trend from the lowest Reynolds numbers and to the higher, it is clear that the computational time is increasing with the increased Reynolds numbers. This might be due to the apparent lack of recirculation zone for the lowest Reynolds numbers, and the streamline behaviour of the flow makes it easy for the model to determine the properties in each node. At the Reynolds numbers where the recirculation starts to appear, the computational time increases. Like for the first simulation domain, the coarseness of the grid due to the relatively few computational nodes might mean that the model struggles to place the recirculation at the discrete points.

In general, the reason for the longer convergence times for the first backwards facing step domain may be that the step height is equal to a half of the inlet height. With the resolution used, the section below the step is only represented by 6 scalar node points in the y -direction, which might not be enough to represent the recirculation accurately, making the model struggle to determine the values at each points. In the second backwards facing step domain, the step is of the same height as the inlet, and is represented by 10 scalar node points in y -direction. This might relax the model since there is less need to force the behaviour of many points into a small set of points.

A higher resolution was not possible to obtain as the models would not converge, or the under-relaxation factors had to be decreased to minuscule values, yielding very long computational times.

6.2.2 Under-relaxation

As specified in section 4, the under-relaxation factors are generally around the magnitude of 0.01 for the backwards facing step and the straight channel models. The factors had to be halved for $Re = 560$ in the first domain, and for the highest Reynolds numbers of 200 and 400 in the second domain.

0.01 is a low value, but higher choices of under-relaxation factors lead to divergence. This could for example have been because the initial guesses were too far away from the solution, or it could be affected by the number of computational nodes. In general it was found that for the straight channel, the under-relaxation factors could be increased when the number of computational nodes was decreased. For the much simpler one-dimensional model as presented in appendix B, the under-relaxation of the velocity is set to 1 and 0.05 for the pressure.

As mentioned in section 2, a suggested relation for the choice of under-relaxation factors are given by equation (2.2.22), where $\alpha_u + \alpha_p = 1$, ideally with α_p and α_u equal to approximately 0.2 and 0.8 respectively. This suggestion is very far away from what was a feasible choice of under-relaxation for the two dimensional models in this thesis, but fits better for the one-dimensional model.

6.2.3 Accuracy of Results

As can be seen by comparing figures 5.36 and 5.38, the recirculation zones after the backwards facing step are consistent with what was found by Biswas et al. [4]. The reattachment lengths for $Re = 100$ and $Re = 400$ are stated in the text in the article to be 2.6 and 7.708 times the step height respectively. This fits well with the profiles in 5.37, where 2.6 times the step height corresponds to $x = 7.6$ and 7.708 times the step height corresponds to $x = 12.7$ which is just outside the edge of the plot.

On the other hand, by comparing figures 5.28-5.34 to figure 5.38, it is apparent that the flow changes directions over the step in a sharper manner than the literature result. That is, the v -velocity component is larger in magnitude than expected in this area. Differences in the solution method or model setup may mean that the results are not directly comparable. The dimensions of the domain as well as the fluid parameters were matched to the specifications given by Biswas et al. [4], but there are other differences that may explain the deviations.

Since the Upwind Difference Scheme is used as the discretisation scheme for the model equations in this thesis, the model is first order accurate. This was chosen because of the simplicity and the stable solution. Biswas et al. [4] instead used a central differencing scheme, which is second order accurate. This means that the results found in this thesis are more stable, but have been more smoothed out and are less precise. As described in section 2.3, the Upwind Differencing Scheme is prone to false diffusion in the results for flows that do not align with one of the coordinate vectors. At the step, the flow takes a more diagonal direction into the expanded section, which may explain the difference in the flow over the step. As mentioned, this effect is worst at low resolutions, which is also the case for this thesis.

Biswas et al. [4] are using a much larger number of computational nodes, which is why the results are able to show the recirculation zones also for the lowest Reynolds numbers. It is specified that approximately 44000 control volumes were used for the corresponding case, with 160 control volumes in y -direction. Also a local grid refining

technique is used in the corner after the step, yielding a more finely meshed grid here.

Another difference is that the channel is rotated 90 degrees around the x -axis in [4], so that the y -direction is the natural choice if gravity is included. It is not specified in the article if gravity is implemented, but if it is this may cause some differences in the results since gravity is neglected in this thesis.

6.3 Model Improvements from Specialisation Project

As mentioned in the introduction, this thesis is a continuation of work done in the fall specialisation project. Models for the one-dimensional and two dimensional straight channels as well as the backwards facing step flow was developed in this project, and the improvements done to the models are discussed in this section. Debugging and troubleshooting of the MATLAB models took up a vast amount of the time during the course of this thesis work.

The issues with the previous code were mainly that the convergence time was vast and that the fluid properties could not be varied, which made it clear that something was wrong in the model.

The large convergence times were due to the fact that the SIMPLE-algorithm was wrongly implemented. The main issue was that the pressure correction was obtained not by using the velocities from the current iteration, but from the previous, acting like an additional under-relaxation of the solution. In addition, the velocity corrections were not performed correctly. Correcting the algorithm reduced the computational time.

For the backwards facing step model, the domain had been split into two computational sub-domains for simplicity with an artificial boundary located at the step. The velocities and pressure for the narrow and wide section were therefore solved separately. This way the code for the straight channel could be implemented directly for the backwards facing step model with the addition of new boundary conditions for the artificial boundary. This unsophisticated method in addition to the slow and wrong solution algorithm caused the model to take approximately 14 hours to converge with the same resolution as is used in this thesis. Instead, in this thesis, the backwards facing step domain is solved as one globally indexed domain, which reduced the computational time drastically.

The second problem was related to that the fluid parameters had to be kept to a set of values, since the models only ran with $u_{in} = 1$ and $\mu = 1$ without divergence. It became clear that this was related to numerical issues, since the desired low velocity values of a around 10^{-5} were overshadowed by the high pressure values of magnitude 10^5 . The low velocities were then likely rounded off to zero in the computations. As a remedy, an adjusted reference pressure was implemented instead, so that the pressure was scaled to zero at the outlet. This removed the large differences in magnitude between the velocities and the pressure, and allowed for the two dimensional straight channel to run with the desired fluid parameters. This model still did not work properly, and even though it converged, but the v -velocity profile had a visibly wrong spear-like behaviour at the outlet. The backwards facing step model still did not run without divergence.

The solution to these still present numerical issues was to transform the fluid flow equations into their dimensionless form. This way, all the fluid parameters were defined as desired, and scaled to and solved with values close to one, resembling the values used in the functioning model from the fall project. This way, the models became more robust to the choice of fluid parameters and can be solved for a range of different Reynolds numbers.

In the troubleshooting phase to discover the mistakes as discussed above, different tests were performed, some of which are explained in section 4.9.1. Adjustments to the boundary conditions are an example of tests that were employed to locate the mistakes. A typo in the velocity scripts that had occurred during the troubleshooting phase took a lot of time to locate.

7

Grid Generation

In this chapter the basic theory behind grid generation is given. Grid generation is used to obtain a mesh in the domain for use when solving the same models as described earlier in this thesis in generalised curvilinear coordinates instead of Cartesian coordinates. The discretisation of these grid generation equations is also given. The discretised governing equations formulated in generalised curvilinear coordinates are stated. The implementation of the grid generation equations in `MATLAB` is explained, and the results are given.

7.1 Theory

This section includes the theory behind grid generation for use when solving fluid flow in generalised curvilinear coordinates. The equations used are for two dimensions.

7.1.1 Generalised Curvilinear Coordinates

Curvilinear coordinates are coordinates that may be located on curved lines. Generalised coordinates are coordinates that are defined relative to coordinates in a simpler reference domain [32]. The reference domain, for example a square, can be divided into points in a simple matter, for example defined by a Cartesian approach. Each point in the reference domain then has a mapping to a point in the physical domain defined by the general coordinates. These mappings across the whole domain creates a non-uniform grid in the physical space.

Equation (7.1.1) shows the mapping from the curvilinear coordinates q^1, q^2 to the Cartesian coordinates x, y .

$$(q^1, q^2) \rightarrow (x, y) \quad (7.1.1)$$

7.1.2 Grid generation

To produce the grid in the physical domain, a grid generator is needed [3]. The solution method for the fluid flow is not dependent on this grid generation. The function of the

grid generator is to make an automatic distribution of grid lines which section off the control volumes in the domain and to provide a connection between the computational and physical domain. Then the corner points of the control volumes can be transformed back into regular control volumes in the physical domain for solving. The properties of the generated grid effects the accuracy, stability and convergence rate of the model, and some models may be more sensitive to the choice of grid generator than other models.

A *structured* grid will be produced with the equations chosen in this work. This means that the curvilinear mesh in the physical domain is generated so that for each curvilinear coordinate, one coordinate line coincides with the boundary of the physical domain [32][33]. A two dimensional structured grid consists of quadrilateral cells, while an unstructured grid consists of triangles [34].

The first step to produce a structured grid is to distribute the boundary grid points for the domain. After this the inner grid points can be obtained. The grid inside the domain is called the volume grid [33]. The volume grid can be found algebraically or by using PDEs, commonly elliptic or hyperbolic equations. When using PDEs to generate the grid, a valid grid is needed for an initial guess. This grid can be generated by use of an algebraic method.

By using an algebraic generator, the transformation between the physical and computational space is described by a direct function. The Transfinite Interpolation (TFI) technique is the most common algebraic grid generator and was first introduced by Gordon and Hall [35]. By first defining the computational points along the boundary of the domain, the central points are obtained by interpolation.

Elliptic equations are most common to use when using PDE generators and were first introduced by Thompson et al. [36]. A smooth grid will be created for the whole domain. There is also flexibility in the use in that it is possible to adjust grid spacing and expansion ratio near the boundaries, and the angle between the grid lines and the boundary can be controlled. A few disadvantages to the method are that the computational time is higher than other methods, and that there are numerical difficulties associated with the method [33].

Figure 7.1 shows an example of a structured grid that has been obtained using the elliptic grid generation equations as described above. The figure is taken from Mohebbi [34].

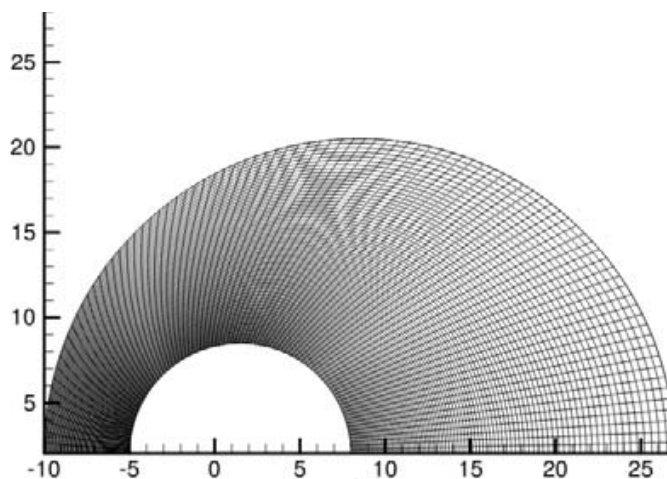


Figure 7.1: Example of a structured grid obtained by use of the elliptic grid generation.

7.1.3 Procedure and Equations

For the grid generation in this thesis, the TFI method is used to obtain an initial grid which serves as an initial grid for an elliptic grid generator. This grid generation can generally be described by the following steps:

1. Define where the corner or boundary points in the physical domain are located in the computational domain
2. Find the location of the boundary points in the physical domain using the TFI method
3. Find the inner computational points of the physical domain using the TFI method
4. Iterate using the elliptic equation with the inner points from the previous step as an initial guess to generate a better grid

These four steps and the equations used are described below.

7.1.3.1 Map corners

Figure 7.2 shows the physical and computational domain and the position of the corner points of the physical domain in the computational domain.

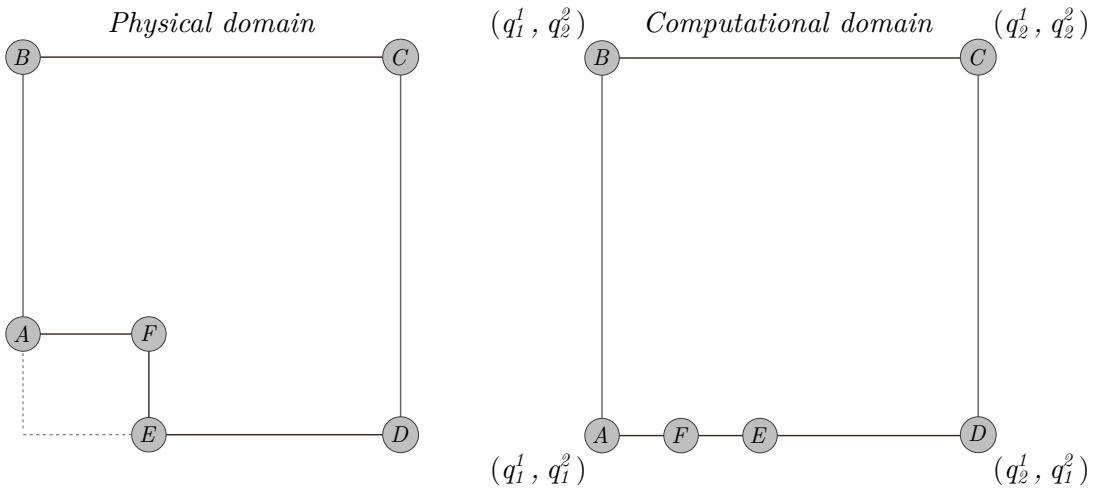


Figure 7.2: Transformation between the physical and the computational domain when using a grid generator.

Equations (7.1.2) to (7.1.5) shows the coordinates of the boundary points in the computational domain as seen in figure 7.2.

$$\text{Line segment } \textcircled{A} \textcircled{B}: \quad q^1 = q_1^1 \quad (7.1.2)$$

$$\text{Line segment } \textcircled{B} \textcircled{C}: \quad q^2 = q_2^2 \quad (7.1.3)$$

$$\text{Line segment } \textcircled{C} \textcircled{D}: \quad q^1 = q_2^1 \quad (7.1.4)$$

$$\text{Line segment } \textcircled{D} \textcircled{A}: \quad q^2 = q_1^2 \quad (7.1.5)$$

In this thesis, the grid spacing δq^1 and δq^2 for both dimensions in the computational domain are chosen to be equal to unity. The span of values of the curvilinear coordinates q^1 and q^2 can be chosen freely, and setting both the width and the height of the grid in the computational space to unity yields a square mesh over the whole square computational domain [37].

7.1.3.2 TFI - Define Boundary Points and Internal Points

The boundary points are defined using Transfinite Interpolation. Equations (7.1.6) and (7.1.7) are the linear Lagrange interpolation functions written individually for q^1 and q^2 respectively. These equations are used to distribute points on the boundary in the computational domain in figure 7.2 [3]. The boundary is defined by lines where either q^1 or q^2 is constant.

$$\mathbf{r}(q^1, q^2) = \sum_{n=1}^2 \phi_n \left(\frac{q^1}{I} \right) \mathbf{r}(q_n^1, q^2) \quad (7.1.6)$$

$$\mathbf{r}(q^1, q^2) = \sum_{m=1}^2 \psi_m \left(\frac{q^2}{J} \right) \mathbf{r}(q^1, q_m^2) \quad (7.1.7)$$

\mathbf{r} is the position vector and I and J are the maximum values of q^1 and q^2 respectively. ϕ and ψ are Lagrange interpolation polynomials, also known as blending functions [3][32][33].

Equation (7.1.8) provides the internal grid points.

$$\begin{aligned} \mathbf{r}(q^1, q^2) = & \sum_{n=1}^2 \phi_n \left(\frac{q^1}{I} \right) \mathbf{r}(q_n^1, q^2) + \sum_{m=1}^2 \psi_m \left(\frac{q^2}{J} \right) \mathbf{r}(q^1, q_m^2) \\ & - \sum_{n=1}^2 \sum_{m=1}^2 \phi_n \left(\frac{q^1}{I} \right) \psi_m \left(\frac{q^2}{J} \right) \mathbf{r}(q_n^1, q_m^2) \end{aligned} \quad (7.1.8)$$

7.1.3.3 Iterate using Elliptic Generation System

The elliptic generation system generates an elliptic grid by solving partial differential equations. Equation (7.1.9) is a system of Poisson equations where the curvilinear coordinates q^1 and q^2 are then the dependent variables and the Cartesian coordinates x and y are the independent variables [33]. This equation is discretised and iterated until the satisfactory grid is achieved.

$$g^{ij} \frac{\partial^2 \mathbf{r}}{\partial q^i \partial q^j} + P^j \frac{\partial \mathbf{r}}{\partial q^j} = 0 \quad (7.1.9)$$

g^{ij} are the contravariant tensor components and P^j are the control functions. Einstein summation notation is used [38][16]. The discretisation of equation (7.1.9) is given in section 7.4. It is common to use second-order central finite differences, which yields a set of linear algebraic equations that is easy to solve [33].

7.2 Governing Equations in General Coordinates

In this section, the governing equations in generalised curvilinear coordinates are stated. These equations can be used to solve the fluid flow problem over the backwards facing step domain in generalised curvilinear coordinates after a grid is obtained using the methods described in this chapter. They are included in this thesis to provide an impression of what the next step is after the grid generation in order to achieve the finished fluid flow model using generalised coordinates. The equations are taken from Melaaen [3], where the procedure with details is explained in sections 3.2 - 3.3. The equations are stated with the notation used by Melaaen [3] since the steps and the meaning behind all symbols are stated there. ξ^i corresponds to q^i above.

Equation (7.2.1) is Navier-Stokes equation in Cartesian coordinates.

$$\frac{\partial}{\partial t} (\rho u_k) + \frac{\partial}{\partial x_i} (\rho u_i u_k) = \frac{\partial}{\partial x_i} \left(\mu \frac{\partial u_k}{\partial x_i} \right) + S_{u_k} \quad (7.2.1)$$

The source term S_{u_k} is defined as in equation (7.2.2).

$$S_{u_k} = \frac{\partial}{\partial x_i} \left(-p \delta_{ik} + \mu \frac{\partial u_i}{\partial x_k} - \frac{2}{3} \mu \delta_{ik} \frac{\partial u_l}{\partial x_l} \right) + B_k \quad (7.2.2)$$

When the source term S_{u_k} in equation (7.2.2) is integrated over the control volume CV , the pressure term in the source term becomes equation (7.2.3).

$$\int_{\delta V} -\frac{\partial p}{\partial x^k} dV = - \left(\frac{\partial p}{\partial x^k} \right)_P \delta V_P = - \left(A_k^j \frac{\partial p}{\partial \xi^j} \right)_P \quad (7.2.3)$$

The second term in the source term S_{u_k} in equation (7.2.2) becomes equation (7.2.5)

$$\int_{\delta V} \nabla \cdot \left(\mu \frac{\partial U}{\partial x^k} \right) dV = \int_{\delta A} \mu \frac{\partial U}{\partial x^k} \cdot d\mathbf{A} \quad (7.2.4)$$

$$= \left[\mu \frac{\partial U}{\partial x^k} \cdot \mathbf{A} \right]_w^e + \left[\mu \frac{\partial U}{\partial x^k} \cdot \mathbf{A} \right]_s^n \quad (7.2.5)$$

where the last terms are given in equation (7.2.6).

$$\mu \frac{\partial U}{\partial x^k} \cdot \mathbf{A} \Big|_{nn} = \mu \frac{A_m^i A_k^j}{J} \frac{\partial u_m}{\partial \xi^j} \Big|_{nn} = \mu \frac{A_m^i}{J} \frac{\partial u_m}{\partial \xi^j} A_k^j \Big|_{nn} \quad (7.2.6)$$

This yields the discretised equation in equation (7.2.7).

$$a_P u_{kP} = \sum_{nb} a_{nb} u_{knb} + b_{u_n} - \left(A_k^j \frac{\partial p}{\partial \xi^j} \right)_P + a_P^0 u_{kP}^0 \quad (7.2.7)$$

with

$$b_{u_k} = b_{NO} + \bar{S}_{1P} + \int_{\delta V} \nabla \cdot \left(\mu \frac{\partial U}{\partial x^k} \right) dV \quad (7.2.8)$$

7.3 Discretisation of the Grid Generation Equations

The two sets of equations needed to produce the grid are discretised in this section in two dimensions. Some parts are written out in three dimensions in appendix D.

7.3.1 Transfinite Interpolation

7.3.1.1 Boundary Points

Equations (7.3.1) and (7.3.2) are the linear Lagrange interpolation functions written individually for q^1 and q^2 respectively.

$$\mathbf{r}(q^1, q^2) = \phi_1 \left(\frac{q^1}{q_1^1} \right) \mathbf{r}(q_1^1, q^2) + \phi_2 \left(\frac{q^1}{q_2^1} \right) \mathbf{r}(q_2^1, q^2) \quad (7.3.1)$$

$$\mathbf{r}(q^1, q^2) = \psi_1 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q^1, q_1^2) + \psi_2 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q^1, q_2^2) \quad (7.3.2)$$

q_1^1 and q_1^2 are the minimum values of q^1 and q^2 respectively and q_2^1 and q_2^2 are the maximum values of q^1 and q^2 respectively, as seen from figure 7.2. The functions ϕ and ψ are Lagrange interpolation polynomials and are defined in equations (7.3.17)-(7.3.20) [32]. The position vector \mathbf{r} is given in equation (7.3.3) for Cartesian coordinates in two dimensions.

$$\mathbf{r} = x\mathbf{e}_x + y\mathbf{e}_y \quad (7.3.3)$$

Equation (7.3.1) is used for the line segments $\textcircled{B}\textcircled{C}$ and $\textcircled{A}\textcircled{D}$ in figure 7.2, and equation (7.3.2) is used for the line segments $\textcircled{A}\textcircled{B}$ and $\textcircled{D}\textcircled{C}$ in figure 7.2. Note that the line segments should always be considered in the positive direction for the coordinate. For instance, the line segments $\textcircled{A}\textcircled{D}$ goes from \textcircled{A} to \textcircled{D} and not the other way around.

The constant coordinate for each line segment as specified in equations (7.1.2)-(7.1.5) can be inserted into equation (7.3.1) or (7.3.2) depending on the line segment as specified in the above paragraph.

Equation (7.3.4) applies to line segment $\textcircled{A}\textcircled{B}$, where q^1 in equation (7.3.2) has been replaced with q_1^1 .

$$\mathbf{r}(q_1^1, q^2) = \psi_1 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_1^1, q_1^2) + \psi_2 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_1^1, q_2^2) \quad (7.3.4)$$

Equation (7.3.5) applies to line segment $\textcircled{B}\textcircled{C}$, where q^2 in equation (7.3.1) has been replaced with q_2^2 .

$$\mathbf{r}(q^1, q_2^2) = \phi_1 \left(\frac{q^1}{q_1^1} \right) \mathbf{r}(q_1^1, q_2^2) + \phi_2 \left(\frac{q^1}{q_1^1} \right) \mathbf{r}(q_2^1, q_2^2) \quad (7.3.5)$$

Equation (7.3.6) applies to line segment $\textcircled{D}\textcircled{C}$, where q^1 in equation (7.3.2) has been replaced with q_2^1 .

$$\mathbf{r}(q_2^1, q^2) = \psi_1 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_2^1, q_1^2) + \psi_2 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_2^1, q_2^2) \quad (7.3.6)$$

Equation (7.3.7) applies to line segment $\textcircled{A}\textcircled{D}$, where q^2 in equation (7.3.1) has been replaced with q_1^2 .

$$\mathbf{r}(q^1, q_1^2) = \phi_1 \left(\frac{q^1}{q_1^1} \right) \mathbf{r}(q_1^1, q_1^2) + \phi_2 \left(\frac{q^1}{q_1^1} \right) \mathbf{r}(q_2^1, q_1^2) \quad (7.3.7)$$

Equations (7.3.4)-(7.3.7) can be written component wise for the Cartesian components x and y by inserting equation (7.3.3) for \mathbf{r} and multiplying with the unit vectors \mathbf{e}_x

and \mathbf{e}_y respectively. The results are given in equations (7.3.8)-(7.3.15).

$$\textcircled{A}\textcircled{B}: x(q_1^1, q^2) = \psi_1\left(\frac{q^2}{q_2^2}\right) x(q_1^1, q_1^2) + \psi_2\left(\frac{q^2}{q_2^2}\right) x(q_1^1, q_2^2) \quad (7.3.8)$$

$$y(q_1^1, q^2) = \psi_1\left(\frac{q^2}{q_2^2}\right) y(q_1^1, q_1^2) + \psi_2\left(\frac{q^2}{q_2^2}\right) y(q_1^1, q_2^2) \quad (7.3.9)$$

$$\textcircled{B}\textcircled{C}: x(q^1, q_2^2) = \phi_1\left(\frac{q^1}{q_1^1}\right) x(q_1^1, q_2^2) + \phi_2\left(\frac{q^1}{q_1^1}\right) x(q_2^1, q_2^2) \quad (7.3.10)$$

$$y(q^1, q_2^2) = \phi_1\left(\frac{q^1}{q_1^1}\right) y(q_1^1, q_2^2) + \phi_2\left(\frac{q^1}{q_1^1}\right) y(q_2^1, q_2^2) \quad (7.3.11)$$

$$\textcircled{D}\textcircled{C}: x(q_2^1, q^2) = \psi_1\left(\frac{q^2}{q_2^2}\right) x(q_2^1, q_1^2) + \psi_2\left(\frac{q^2}{q_2^2}\right) x(q_2^1, q_2^2) \quad (7.3.12)$$

$$y(q_2^1, q^2) = \psi_1\left(\frac{q^2}{q_2^2}\right) y(q_2^1, q_1^2) + \psi_2\left(\frac{q^2}{q_2^2}\right) y(q_2^1, q_2^2) \quad (7.3.13)$$

$$\textcircled{A}\textcircled{D}: x(q^1, q_1^2) = \phi_1\left(\frac{q^1}{q_1^1}\right) x(q_1^1, q_1^2) + \phi_2\left(\frac{q^1}{q_1^1}\right) x(q_2^1, q_1^2) \quad (7.3.14)$$

$$y(q^1, q_1^2) = \phi_1\left(\frac{q^1}{q_1^1}\right) y(q_1^1, q_1^2) + \phi_2\left(\frac{q^1}{q_1^1}\right) y(q_2^1, q_1^2) \quad (7.3.15)$$

In equations (7.3.8)-(7.3.15) the x - and y -points on the right hand side correspond to the corner points in figure 7.2, and are known values that can be inserted.

The functions ϕ and ψ are Lagrange interpolation polynomials, and are defined by equation (7.3.16) [32].

$$\phi_n\left(\frac{q^i}{q_{max}^i}\right) = \prod_{k=1, k \neq n}^N \frac{q^i - q_k^i}{q_n^i - q_k^i} \quad (k \neq n) \quad (7.3.16)$$

The functions ϕ and ψ are chosen to be linear functions as given in equations (7.3.17)-(7.3.20). This yields equally spaced points on the boundaries [3]. ϕ is applied for q^1 and ψ is applied for q^2 .

$$\phi_1\left(\frac{q^1}{q_2^1}\right) = 1 - \frac{q^1}{q_2^1} \quad (7.3.17)$$

$$\phi_2\left(\frac{q^1}{q_2^1}\right) = \frac{q^1}{q_2^1} \quad (7.3.18)$$

$$\psi_1\left(\frac{q^2}{q_2^2}\right) = 1 - \frac{q^2}{q_2^2} \quad (7.3.19)$$

$$\psi_2\left(\frac{q^2}{q_2^2}\right) = \frac{q^2}{q_2^2} \quad (7.3.20)$$

More complex functions can also be used, Melaaen [3] suggests use of Lagrangian interpolation polynomials, which makes it possible to have more control over the distance between the grid lines.

ϕ and ψ can then be inserted into equations (7.3.8)-(7.3.15) to yield equations (7.3.21)-(7.3.28).

$$\textcircled{A}\textcircled{B}: x(q_1^1, q_2^2) = \left(1 - \frac{q_2^2}{q_2^2}\right) x(q_1^1, q_1^2) + \frac{q_2^2}{q_2^2} x(q_1^1, q_2^2) \quad (7.3.21)$$

$$y(q_1^1, q_2^2) = \left(1 - \frac{q_2^2}{q_2^2}\right) y(q_1^1, q_1^2) + \frac{q_2^2}{q_2^2} y(q_1^1, q_2^2) \quad (7.3.22)$$

$$\textcircled{B}\textcircled{C}: x(q_1^1, q_2^2) = \left(1 - \frac{q_1^1}{q_2^1}\right) x(q_1^1, q_2^2) + \frac{q_1^1}{q_2^1} x(q_2^1, q_2^2) \quad (7.3.23)$$

$$y(q_1^1, q_2^2) = \left(1 - \frac{q_1^1}{q_2^1}\right) y(q_1^1, q_2^2) + \frac{q_1^1}{q_2^1} y(q_2^1, q_2^2) \quad (7.3.24)$$

$$\textcircled{D}\textcircled{C}: x(q_2^1, q_2^2) = \left(1 - \frac{q_2^2}{q_2^2}\right) x(q_2^1, q_1^2) + \frac{q_2^2}{q_2^2} x(q_2^1, q_2^2) \quad (7.3.25)$$

$$y(q_2^1, q_2^2) = \left(1 - \frac{q_2^2}{q_2^2}\right) y(q_2^1, q_1^2) + \frac{q_2^2}{q_2^2} y(q_2^1, q_2^2) \quad (7.3.26)$$

$$\textcircled{A}\textcircled{D}: x(q_1^1, q_2^2) = \left(1 - \frac{q_1^1}{q_2^1}\right) x(q_1^1, q_1^2) + \frac{q_1^1}{q_2^1} x(q_2^1, q_1^2) \quad (7.3.27)$$

$$y(q_1^1, q_2^2) = \left(1 - \frac{q_1^1}{q_2^1}\right) y(q_1^1, q_1^2) + \frac{q_1^1}{q_2^1} y(q_2^1, q_1^2) \quad (7.3.28)$$

The x - and y -points in equations (7.3.21)-(7.3.28) can be written on the form x_{AB} as in equations (7.3.29)-(7.3.36).

$$\textcircled{A}\textcircled{B}: x_{AB} = \left(1 - \frac{q_2^2}{q_2^2}\right) x_A + \frac{q_2^2}{q_2^2} x_B \quad (7.3.29)$$

$$y_{AB} = \left(1 - \frac{q_2^2}{q_2^2}\right) y_A + \frac{q_2^2}{q_2^2} y_B \quad (7.3.30)$$

$$\textcircled{B}\textcircled{C}: x_{BC} = \left(1 - \frac{q_1^1}{q_2^1}\right) x_B + \frac{q_1^1}{q_2^1} x_C \quad (7.3.31)$$

$$y_{BC} = \left(1 - \frac{q_1^1}{q_2^1}\right) y_B + \frac{q_1^1}{q_2^1} y_C \quad (7.3.32)$$

$$\textcircled{D}\textcircled{C}: x_{DC} = \left(1 - \frac{q_2^2}{q_2^2}\right) x_D + \frac{q_2^2}{q_2^2} x_C \quad (7.3.33)$$

$$y_{DC} = \left(1 - \frac{q_2^2}{q_2^2}\right) y_D + \frac{q_2^2}{q_2^2} y_C \quad (7.3.34)$$

$$\textcircled{A}\textcircled{D}: x_{AD} = \left(1 - \frac{q_1^1}{q_2^1}\right) x_A + \frac{q_1^1}{q_2^1} x_D \quad (7.3.35)$$

$$y_{AD} = \left(1 - \frac{q_1^1}{q_2^1}\right) y_A + \frac{q_1^1}{q_2^1} y_D \quad (7.3.36)$$

7.3.1.2 Internal Points

Equation (7.1.8), written out in equation (7.3.37) yields the distribution of grid points inside the domain when the boundary points are known from equations (7.3.1) and (7.3.2) above.

$$\begin{aligned} \mathbf{r}(q^1, q^2) = & \phi_1 \left(\frac{q^1}{q_2^1} \right) \mathbf{r}(q_1^1, q^2) + \phi_2 \left(\frac{q^1}{q_2^1} \right) \mathbf{r}(q_2^1, q^2) + \psi_1 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q^1, q_1^2) + \psi_2 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q^1, q_2^2) \\ & + \phi_1 \left(\frac{q^1}{q_2^1} \right) \psi_1 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_1^1, q_1^2) + \phi_1 \left(\frac{q^1}{q_2^1} \right) \psi_2 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_1^1, q_2^2) \\ & + \phi_2 \left(\frac{q^1}{q_2^1} \right) \psi_1 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_2^1, q_1^2) + \phi_2 \left(\frac{q^1}{q_2^1} \right) \psi_2 \left(\frac{q^2}{q_2^2} \right) \mathbf{r}(q_2^1, q_2^2) \end{aligned} \quad (7.3.37)$$

The components of equation (7.3.37) can be obtained like for the boundary points equations above, by replacing the position vector \mathbf{r} with its definition in equation (7.3.3) and multiplying with the unit vectors \mathbf{e}_x and \mathbf{e}_y to obtain the x - and y -component respectively as given in equations (7.3.38) and (7.3.39).

$$\begin{aligned} x(q^1, q^2) = & \phi_1 \left(\frac{q^1}{q_2^1} \right) x(q_1^1, q^2) + \phi_2 \left(\frac{q^1}{q_2^1} \right) x(q_2^1, q^2) + \psi_1 \left(\frac{q^2}{q_2^2} \right) x(q^1, q_1^2) + \psi_2 \left(\frac{q^2}{q_2^2} \right) x(q^1, q_2^2) \\ & + \phi_1 \left(\frac{q^1}{q_2^1} \right) \psi_1 \left(\frac{q^2}{q_2^2} \right) x(q_1^1, q_1^2) + \phi_1 \left(\frac{q^1}{q_2^1} \right) \psi_2 \left(\frac{q^2}{q_2^2} \right) x(q_1^1, q_2^2) \\ & + \phi_2 \left(\frac{q^1}{q_2^1} \right) \psi_1 \left(\frac{q^2}{q_2^2} \right) x(q_2^1, q_1^2) + \phi_2 \left(\frac{q^1}{q_2^1} \right) \psi_2 \left(\frac{q^2}{q_2^2} \right) x(q_2^1, q_2^2) \end{aligned} \quad (7.3.38)$$

$$\begin{aligned} y(q^1, q^2) = & \phi_1 \left(\frac{q^1}{q_2^1} \right) y(q_1^1, q^2) + \phi_2 \left(\frac{q^1}{q_2^1} \right) y(q_2^1, q^2) + \psi_1 \left(\frac{q^2}{q_2^2} \right) y(q^1, q_1^2) + \psi_2 \left(\frac{q^2}{q_2^2} \right) y(q^1, q_2^2) \\ & + \phi_1 \left(\frac{q^1}{q_2^1} \right) \psi_1 \left(\frac{q^2}{q_2^2} \right) y(q_1^1, q_1^2) + \phi_1 \left(\frac{q^1}{q_2^1} \right) \psi_2 \left(\frac{q^2}{q_2^2} \right) y(q_1^1, q_2^2) \\ & + \phi_2 \left(\frac{q^1}{q_2^1} \right) \psi_1 \left(\frac{q^2}{q_2^2} \right) y(q_2^1, q_1^2) + \phi_2 \left(\frac{q^1}{q_2^1} \right) \psi_2 \left(\frac{q^2}{q_2^2} \right) y(q_2^1, q_2^2) \end{aligned} \quad (7.3.39)$$

The same functions ϕ and ψ in equations (7.3.17)-(7.3.20) are inserted, yielding equations (7.3.40) and (7.3.41).

$$\begin{aligned} x(q^1, q^2) = & \left(1 - \frac{q^1}{q_2^1} \right) x(q_1^1, q^2) + \frac{q^1}{q_2^1} x(q_2^1, q^2) + \left(1 - \frac{q^2}{q_2^2} \right) x(q^1, q_1^2) + \frac{q^2}{q_2^2} x(q^1, q_2^2) \\ & + \left(1 - \frac{q^1}{q_2^1} \right) \left(1 - \frac{q^2}{q_2^2} \right) x(q_1^1, q_1^2) + \left(1 - \frac{q^1}{q_2^1} \right) \frac{q^2}{q_2^2} x(q_1^1, q_2^2) \\ & + \frac{q^1}{q_2^1} \left(1 - \frac{q^2}{q_2^2} \right) x(q_2^1, q_1^2) + \frac{q^1}{q_2^1} \frac{q^2}{q_2^2} x(q_2^1, q_2^2) \end{aligned} \quad (7.3.40)$$

$$\begin{aligned} y(q^1, q^2) = & \left(1 - \frac{q^1}{q_2^1} \right) y(q_1^1, q^2) + \frac{q^1}{q_2^1} y(q_2^1, q^2) + \left(1 - \frac{q^2}{q_2^2} \right) y(q^1, q_1^2) + \frac{q^2}{q_2^2} y(q^1, q_2^2) \\ & + \left(1 - \frac{q^1}{q_2^1} \right) \left(1 - \frac{q^2}{q_2^2} \right) y(q_1^1, q_1^2) + \left(1 - \frac{q^1}{q_2^1} \right) \frac{q^2}{q_2^2} y(q_1^1, q_2^2) \\ & + \frac{q^1}{q_2^1} \left(1 - \frac{q^2}{q_2^2} \right) y(q_2^1, q_1^2) + \frac{q^1}{q_2^1} \frac{q^2}{q_2^2} y(q_2^1, q_2^2) \end{aligned} \quad (7.3.41)$$

The x - and y -points in equations (7.3.40)-(7.3.41) can be written on the form x_{AB} as in equations (7.3.42)-(7.3.43).

$$\begin{aligned} x &= \left(1 - \frac{q^1}{q_2^1}\right) x_{AB} + \frac{q^1}{q_2^1} x_{DC} + \left(1 - \frac{q^2}{q_2^2}\right) x_{AD} + \frac{q^2}{q_2^2} x_{BC} \\ &+ \left(1 - \frac{q^1}{q_2^1}\right) \left(1 - \frac{q^2}{q_2^2}\right) x_A + \left(1 - \frac{q^1}{q_2^1}\right) \frac{q^2}{q_2^2} x_B + \frac{q^1}{q_2^1} \left(1 - \frac{q^2}{q_2^2}\right) x_D + \frac{q^1 q^2}{q_2^1 q_2^2} x_C \end{aligned} \quad (7.3.42)$$

$$\begin{aligned} y &= \left(1 - \frac{q^1}{q_2^1}\right) y_{AB} + \frac{q^1}{q_2^1} y_{DC} + \left(1 - \frac{q^2}{q_2^2}\right) y_{AD} + \frac{q^2}{q_2^2} y_{BC} \\ &+ \left(1 - \frac{q^1}{q_2^1}\right) \left(1 - \frac{q^2}{q_2^2}\right) y_A + \left(1 - \frac{q^1}{q_2^1}\right) \frac{q^2}{q_2^2} y_B + \frac{q^1}{q_2^1} \left(1 - \frac{q^2}{q_2^2}\right) y_D + \frac{q^1 q^2}{q_2^1 q_2^2} y_C \end{aligned} \quad (7.3.43)$$

7.3.2 Elliptic Generation System

The equation to be discretised to obtain the improved grid is equation (7.3.44) [3].

$$g^{ij} \frac{\partial^2 \mathbf{r}}{\partial q^i \partial q^j} + P^j \frac{\partial \mathbf{r}}{\partial \xi^j} = 0 \quad (7.3.44)$$

g^{ij} is the contravariant tensor components, $P^j = \nabla^2 q^j$ are the control functions. Einstein summation notation is used [16][38].

$$g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial \mathbf{r}}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial \mathbf{r}}{\partial q^j} = \mathbf{0} \quad (7.3.45)$$

The position vector \mathbf{r} is given in equation (7.3.3) for Cartesian coordinates in two dimensions. The \mathbf{r} -vector is inserted into equation (7.3.45) and simplified to yield equation (7.3.46). The derivative of the base vectors \mathbf{e}_x and \mathbf{e}_y are zero.

$$\begin{aligned} &g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial}{\partial q^j} (x\mathbf{e}_x + y\mathbf{e}_y) \right) + \nabla^2 q^j \frac{\partial}{\partial q^j} (x\mathbf{e}_x + y\mathbf{e}_y) = \mathbf{0} \\ &g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial}{\partial q^j} (x\mathbf{e}_x) \right) + g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial}{\partial q^j} (y\mathbf{e}_y) \right) + \nabla^2 q^j \frac{\partial}{\partial q^j} (x\mathbf{e}_x) + \nabla^2 q^j \frac{\partial}{\partial q^j} (y\mathbf{e}_y) = \mathbf{0} \\ &g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial x}{\partial q^j} \right) \mathbf{e}_x + g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial y}{\partial q^j} \right) \mathbf{e}_y + \nabla^2 q^j \frac{\partial x}{\partial q^j} \mathbf{e}_x + \nabla^2 q^j \frac{\partial y}{\partial q^j} \mathbf{e}_y = \mathbf{0} \end{aligned} \quad (7.3.46)$$

The x -component of equation (7.3.46) can then be obtained by taking the dot product with \mathbf{e}_x . The result is equation (7.3.48).

$$\begin{aligned} &g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial x}{\partial q^j} \right) \mathbf{e}_x \cdot \mathbf{e}_x + g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial y}{\partial q^j} \right) \mathbf{e}_y \cdot \mathbf{e}_x \\ &+ \nabla^2 q^j \frac{\partial x}{\partial q^j} \mathbf{e}_x \cdot \mathbf{e}_x + \nabla^2 q^j \frac{\partial y}{\partial q^j} \mathbf{e}_y \cdot \mathbf{e}_x = \mathbf{0} \cdot \mathbf{e}_x \end{aligned} \quad (7.3.47)$$

$$g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial x}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial x}{\partial q^j} = 0 \quad (7.3.48)$$

Similarly, the y -component of equation (7.3.46) is obtained by taking the dot product with \mathbf{e}_y . The result is equation (7.3.50).

$$g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial x}{\partial q^j} \right) \mathbf{e}_x \cdot \mathbf{e}_y + g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial y}{\partial q^j} \right) \mathbf{e}_y \cdot \mathbf{e}_y + \nabla^2 q^j \frac{\partial x}{\partial q^j} \mathbf{e}_x \cdot \mathbf{e}_y + \nabla^2 q^j \frac{\partial y}{\partial q^j} \mathbf{e}_y \cdot \mathbf{e}_y = \mathbf{0} \cdot \mathbf{e}_y \quad (7.3.49)$$

$$g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial y}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial y}{\partial q^j} = 0 \quad (7.3.50)$$

The above equations are written using Einstein's summation notation, and these summations as shown in equations (7.3.51) and (7.3.52).

$$\sum_{i=1}^2 \sum_{j=1}^2 \left(g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial x}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial x}{\partial q^j} \right) = 0 \quad (7.3.51)$$

$$\sum_{i=1}^2 \sum_{j=1}^2 \left(g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial y}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial y}{\partial q^j} \right) = 0 \quad (7.3.52)$$

Taking the sums yields equations (7.3.53) and (7.3.54) for the x - and y -component respectively.

$$g^{11} \frac{\partial}{\partial q^1} \left(\frac{\partial x}{\partial q^1} \right) + g^{12} \frac{\partial}{\partial q^1} \left(\frac{\partial x}{\partial q^2} \right) + g^{21} \frac{\partial}{\partial q^2} \left(\frac{\partial x}{\partial q^1} \right) + g^{22} \frac{\partial}{\partial q^2} \left(\frac{\partial x}{\partial q^2} \right) + \nabla^2 q^1 \frac{\partial x}{\partial q^1} + \nabla^2 q^2 \frac{\partial x}{\partial q^2} = 0 \quad (7.3.53)$$

$$g^{11} \frac{\partial}{\partial q^1} \left(\frac{\partial y}{\partial q^1} \right) + g^{12} \frac{\partial}{\partial q^1} \left(\frac{\partial y}{\partial q^2} \right) + g^{21} \frac{\partial}{\partial q^2} \left(\frac{\partial y}{\partial q^1} \right) + g^{22} \frac{\partial}{\partial q^2} \left(\frac{\partial y}{\partial q^2} \right) + \nabla^2 q^1 \frac{\partial y}{\partial q^1} + \nabla^2 q^2 \frac{\partial y}{\partial q^2} = 0 \quad (7.3.54)$$

7.3.2.1 Central differencing

The derivatives in equations (7.3.53) and (7.3.54) are approximated with central differences. This differencing will be given first before the rest of the unknown terms g^{ij} and $\nabla^2 q^i$ are specified.

The central differences for discretising the derivatives are given by equations (7.3.55)-(7.3.59).

$$\left. \frac{\partial \varphi}{\partial q^1} \right|_{i,j} = \frac{\varphi_{i+1,j} - \varphi_{i-1,j}}{2\delta q^1} \quad (7.3.55)$$

$$\left. \frac{\partial \varphi}{\partial q^2} \right|_{i,j} = \frac{\varphi_{i,j+1} - \varphi_{i,j-1}}{2\delta q^2} \quad (7.3.56)$$

$$\left. \frac{\partial^2 \varphi}{(\partial q^1)^2} \right|_{i,j} = \frac{\varphi_{i+1,j} + \varphi_{i-1,j} - 2\varphi_{i,j}}{(\delta q^1)^2} \quad (7.3.57)$$

$$\left. \frac{\partial^2 \varphi}{(\partial q^2)^2} \right|_{i,j} = \frac{\varphi_{i,j+1} + \varphi_{i,j-1} - 2\varphi_{i,j}}{(\delta q^2)^2} \quad (7.3.58)$$

$$\left. \frac{\partial^2 \varphi}{\partial q^1 \partial q^2} \right|_{i,j} = \frac{\varphi_{i+1,j+1} + \varphi_{i-1,j-1} - \varphi_{i+1,j-1} - \varphi_{i-1,j+1}}{4\delta q^1 \delta q^2} \quad (7.3.59)$$

δq^1 and δq^2 are the length and width of the control volumes in the computational domain. In this case, they are set equal to unity since the grid spacing is chosen to be one for both dimensions. This yields equations (7.3.60)- (7.3.64).

$$\left. \frac{\partial \varphi}{\partial q^1} \right|_{i,j} = \frac{\varphi_{i+1,j} - \varphi_{i-1,j}}{2} \quad (7.3.60)$$

$$\left. \frac{\partial \varphi}{\partial q^2} \right|_{i,j} = \frac{\varphi_{i,j+1} - \varphi_{i,j-1}}{2} \quad (7.3.61)$$

$$\left. \frac{\partial^2 \varphi}{(\partial q^1)^2} \right|_{i,j} = (\varphi_{i+1,j} + \varphi_{i-1,j} - 2\varphi_{i,j}) \quad (7.3.62)$$

$$\left. \frac{\partial^2 \varphi}{(\partial q^2)^2} \right|_{i,j} = (\varphi_{i,j+1} + \varphi_{i,j-1} - 2\varphi_{i,j}) \quad (7.3.63)$$

$$\left. \frac{\partial^2 \varphi}{\partial q^1 \partial q^2} \right|_{i,j} = \frac{\varphi_{i+1,j+1} + \varphi_{i-1,j-1} - \varphi_{i+1,j-1} - \varphi_{i-1,j+1}}{4} \quad (7.3.64)$$

Equations (7.3.60)- (7.3.64) inserted into equations (7.3.53) and (7.3.54) with φ being x and y respectively, this yields equations (7.3.65) and (7.3.66).

$$\begin{aligned} g^{11} (x_{i+1,j} + x_{i-1,j} - 2x_{i,j}) + g^{12} \frac{x_{i+1,j+1} + x_{i-1,j-1} - x_{i+1,j-1} - x_{i-1,j+1}}{4} \\ + g^{21} \frac{x_{i+1,j+1} + x_{i-1,j-1} - x_{i+1,j-1} - x_{i-1,j+1}}{4} + g^{22} (x_{i,j+1} + x_{i,j-1} - 2x_{i,j}) \\ + \nabla^2 q^1 \frac{y_{i+1,j} - y_{i-1,j}}{2} + \nabla^2 q^2 \frac{y_{i,j+1} - y_{i,j-1}}{2} = 0 \quad (7.3.65) \end{aligned}$$

$$\begin{aligned} g^{11} (y_{i+1,j} + y_{i-1,j} - 2y_{i,j}) + g^{12} \frac{y_{i+1,j+1} + y_{i-1,j-1} - y_{i+1,j-1} - y_{i-1,j+1}}{4} \\ + g^{21} \frac{y_{i+1,j+1} + y_{i-1,j-1} - y_{i+1,j-1} - y_{i-1,j+1}}{4} + g^{22} (y_{i,j+1} + y_{i,j-1} - 2y_{i,j}) \\ + \nabla^2 q^1 \frac{y_{i+1,j} - y_{i-1,j}}{2} + \nabla^2 q^2 \frac{y_{i,j+1} - y_{i,j-1}}{2} = 0 \quad (7.3.66) \end{aligned}$$

Rearranged to gather the same terms, equations (7.3.65) and (7.3.66) become equations (7.3.67) and (7.3.68).

$$\begin{aligned}
& x_{i,j} \left(-2g^{11} - 2g^{22} \right) + x_{i+1,j} \left(g^{11} + \frac{\nabla^2 q^1}{2} \right) + x_{i-1,j} \left(g^{11} - \frac{\nabla^2 q^1}{2} \right) \\
& \quad + x_{i,j+1} \left(g^{22} + \frac{\nabla^2 q^2}{2} \right) + x_{i,j-1} \left(g^{22} - \frac{\nabla^2 q^2}{2} \right) \\
& \quad + x_{i+1,j+1} \left(\frac{g^{12}}{4} + \frac{g^{21}}{4} \right) + x_{i-1,j+1} \left(-\frac{g^{12}}{4} - \frac{g^{21}}{4} \right) \\
& \quad + x_{i+1,j-1} \left(-\frac{g^{12}}{4} - \frac{g^{21}}{4} \right) + x_{i-1,j-1} \left(\frac{g^{12}}{4} + \frac{g^{21}}{4} \right) = 0 \quad (7.3.67)
\end{aligned}$$

$$\begin{aligned}
& y_{i,j} \left(-2g^{11} - 2g^{22} \right) + y_{i+1,j} \left(g^{11} + \frac{\nabla^2 q^1}{2} \right) + y_{i-1,j} \left(g^{11} - \frac{\nabla^2 q^1}{2} \right) \\
& \quad + y_{i,j+1} \left(g^{22} + \frac{\nabla^2 q^2}{2} \right) + y_{i,j-1} \left(g^{22} - \frac{\nabla^2 q^2}{2} \right) \\
& \quad + y_{i+1,j+1} \left(\frac{g^{12}}{4} + \frac{g^{21}}{4} \right) + y_{i-1,j+1} \left(-\frac{g^{12}}{4} - \frac{g^{21}}{4} \right) \\
& \quad + y_{i+1,j-1} \left(-\frac{g^{12}}{4} - \frac{g^{21}}{4} \right) + y_{i-1,j-1} \left(\frac{g^{12}}{4} + \frac{g^{21}}{4} \right) = 0 \quad (7.3.68)
\end{aligned}$$

Equations (7.3.67) and (7.3.68) can be written in coefficient form for simplicity. Equation (7.3.69) shows the discretised elliptic grid generation for the x -component.

$$\begin{aligned}
& c_{i,j}^x x_{i,j} + c_{i+1,j}^x x_{i+1,j} + c_{i-1,j}^x x_{i-1,j} + c_{i,j+1}^x x_{i,j+1} + c_{i,j-1}^x x_{i,j-1} \\
& \quad + c_{i+1,j+1}^x x_{i+1,j+1} + c_{i-1,j+1}^x x_{i-1,j+1} + c_{i+1,j-1}^x x_{i+1,j-1} + c_{i-1,j-1}^x x_{i-1,j-1} = 0 \quad (7.3.69)
\end{aligned}$$

with

$$c_{i,j}^x = -2g^{11} - 2g^{22} \quad (7.3.70)$$

$$c_{i+1,j}^x = g^{11} + \frac{\nabla^2 q^1}{2} \quad (7.3.71)$$

$$c_{i-1,j}^x = g^{11} - \frac{\nabla^2 q^1}{2} \quad (7.3.72)$$

$$c_{i,j+1}^x = g^{22} + \frac{\nabla^2 q^2}{2} \quad (7.3.73)$$

$$c_{i,j-1}^x = g^{22} - \frac{\nabla^2 q^2}{2} \quad (7.3.74)$$

$$c_{i+1,j+1}^x = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.75)$$

$$c_{i-1,j+1}^x = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.76)$$

$$c_{i+1,j-1}^x = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.77)$$

$$c_{i-1,j-1}^x = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.78)$$

Equation (7.3.79) shows the discretised elliptic grid generation for the y -component.

$$c_{i,j}^y y_{i,j} + c_{i+1,j}^y y_{i+1,j} + c_{i-1,j}^y y_{i-1,j} + c_{i,j+1}^y y_{i,j+1} + c_{i,j-1}^y y_{i,j-1} \\ + c_{i+1,j+1}^y y_{i+1,j+1} + c_{i-1,j+1}^y y_{i-1,j+1} + c_{i+1,j-1}^y y_{i+1,j-1} + c_{i-1,j-1}^y y_{i-1,j-1} = 0 \quad (7.3.79)$$

with

$$c_{i,j}^y = -2g^{11} - 2g^{22} \quad (7.3.80)$$

$$c_{i+1,j}^y = g^{11} + \frac{\nabla^2 q^1}{2} \quad (7.3.81)$$

$$c_{i-1,j}^y = g^{11} - \frac{\nabla^2 q^1}{2} \quad (7.3.82)$$

$$c_{i,j+1}^y = g^{22} + \frac{\nabla^2 q^2}{2} \quad (7.3.83)$$

$$c_{i,j-1}^y = g^{22} - \frac{\nabla^2 q^2}{2} \quad (7.3.84)$$

$$c_{i+1,j+1}^y = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.85)$$

$$c_{i-1,j+1}^y = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.86)$$

$$c_{i+1,j-1}^y = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.87)$$

$$c_{i-1,j-1}^y = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.88)$$

The contravariant tensor components g^{ij} and the Poisson equations $\nabla^2 q^i$ still need defining, which is given in the next section.

7.3.2.2 Contravariant Tensor Components

The next step is to obtain an expression for the contravariant tensor components g^{ij} , which is given by equation (7.3.89).

$$g^{ij} = \frac{\mathbf{A}^{(i)} \cdot \mathbf{A}^{(j)}}{J^2} \quad (7.3.89)$$

Below follow some definitions of the parameters that make up this equation. $\mathbf{A}^{(i)}$ is given first and J is given from equation (7.3.117).

$\mathbf{A}^{(i)}$ is the face area vector and contains the face areas of the cells in the grid in the physical domain [3]. It is necessary to define $\mathbf{A}^{(i)}$ using all three dimensions, and the expressions for $\mathbf{A}^{(i)}$ will be simplified to two dimensions after the expressions are obtained.

$\mathbf{A}^{(i)}$ is given by equation (7.3.90) [39].

$$\mathbf{A}^{(k)} = A_j^k \mathbf{e}_j = \mathbf{g}_l \times \mathbf{g}_m \quad (7.3.90)$$

where \mathbf{e}_j is the Cartesian base vector and \mathbf{g}_l and \mathbf{g}_l are general base vectors. ε_{klm} is the permutation symbol and is given by equation (7.3.91)[40].

$$\varepsilon_{klm} = \begin{cases} +1 \rightarrow klm = 123, 231 \text{ or } 312 \\ -1 \rightarrow klm = 321, 213 \text{ or } 132 \\ 0 \rightarrow \text{any indices are the same} \end{cases} \quad (7.3.91)$$

k , l and m in equation (7.3.90) are cyclic which means that the order of the indices cannot be interchanged and still produce the same result [27][30][41]. k , l and m in equation (7.3.90) are cyclic and follow the order of the positive value of the permutation symbol as given in equation (7.3.91). This means that klm take the values 123, 231 or 312.

The general base vector \mathbf{g}_i is defined as in equation (7.3.92).

$$\mathbf{g}_i = \frac{\partial x^j}{\partial q^i} \mathbf{e}_j \quad (7.3.92)$$

where $\frac{\partial x^j}{\partial q^i}$ can also be noted J_i^j as defined by equation (7.3.93).

$$J_i^j = \frac{\partial x^j}{\partial q^i} \quad (7.3.93)$$

Equation (7.3.90) can then be rewritten to yield equation (7.3.94) by use of equation (7.3.92).

$$\mathbf{A}^{(k)} = \frac{\partial x^p}{\partial q^l} \mathbf{e}_p \times \frac{\partial x^q}{\partial q^m} \mathbf{e}_q \quad (7.3.94)$$

The indices p and q are selected for x^j in equation (7.3.92) as j does not take the same index for \mathbf{g}_l and \mathbf{g}_m . Writing out the cross product yields equation (7.3.95). [40] [42]

$$\begin{aligned} \mathbf{A}^{(k)} &= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \mathbf{e}_p \times \mathbf{e}_q \\ &= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \varepsilon_{pqr} \mathbf{e}_r \end{aligned} \quad (7.3.95)$$

ε_{pqr} is the permutation symbol as given in equation (7.3.91) and r is the third possible index for x not equal to p or q . Now the components of $\mathbf{A}^{(k)}$ in Cartesian coordinates can be found by taking the dot product with each unit vector \mathbf{e}_i where i is equal to 1, 2, 3, as shown in equation (7.3.96), which comes from equation (7.3.90).

$$A_i^{(k)} = \mathbf{A}^{(k)} \cdot \mathbf{e}_i \quad (7.3.96)$$

This yields equations (7.3.97), (7.3.98) and (7.3.99) for the three components.

$$\begin{aligned} A_1^{(k)} &= \mathbf{A}^{(k)} \cdot \mathbf{e}_1 \\ &= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \varepsilon_{pqr} \mathbf{e}_r \cdot \mathbf{e}_1 \\ &= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \varepsilon_{pq1} \\ &= \frac{\partial x^2}{\partial q^l} \frac{\partial x^3}{\partial q^m} - \frac{\partial x^3}{\partial q^l} \frac{\partial x^2}{\partial q^m} \end{aligned} \quad (7.3.97)$$

$$\begin{aligned} A_2^{(k)} &= \mathbf{A}^{(k)} \cdot \mathbf{e}_2 \\ &= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \varepsilon_{pqr} \mathbf{e}_r \cdot \mathbf{e}_2 \\ &= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \varepsilon_{pq2} \\ &= \frac{\partial x^3}{\partial q^l} \frac{\partial x^1}{\partial q^m} - \frac{\partial x^1}{\partial q^l} \frac{\partial x^3}{\partial q^m} \end{aligned} \quad (7.3.98)$$

$$\begin{aligned}
A_3^{(k)} &= \mathbf{A}^{(k)} \cdot \mathbf{e}_3 \\
&= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \varepsilon_{pqr} \mathbf{e}_r \cdot \mathbf{e}_3 \\
&= \frac{\partial x^p}{\partial q^l} \frac{\partial x^q}{\partial q^m} \varepsilon_{pq3} \\
&= \frac{\partial x^1}{\partial q^l} \frac{\partial x^2}{\partial q^m} - \frac{\partial x^2}{\partial q^l} \frac{\partial x^1}{\partial q^m}
\end{aligned} \tag{7.3.99}$$

Further, all the nine components of the three area vectors are given by equations (7.3.100)-(7.3.108), which are obtained by filling in the cyclic values of klm which are 123, 231 or 312.

$$A_1^1 = \frac{\partial x^2}{\partial q^2} \frac{\partial x^3}{\partial q^3} - \frac{\partial x^3}{\partial q^2} \frac{\partial x^2}{\partial q^3} \tag{7.3.100}$$

$$A_1^2 = \frac{\partial x^2}{\partial q^3} \frac{\partial x^3}{\partial q^1} - \frac{\partial x^3}{\partial q^3} \frac{\partial x^2}{\partial q^1} \tag{7.3.101}$$

$$A_1^3 = \frac{\partial x^2}{\partial q^1} \frac{\partial x^3}{\partial q^2} - \frac{\partial x^3}{\partial q^1} \frac{\partial x^2}{\partial q^2} \tag{7.3.102}$$

$$A_2^1 = \frac{\partial x^3}{\partial q^2} \frac{\partial x^1}{\partial q^3} - \frac{\partial x^1}{\partial q^2} \frac{\partial x^3}{\partial q^3} \tag{7.3.103}$$

$$A_2^2 = \frac{\partial x^3}{\partial q^3} \frac{\partial x^1}{\partial q^1} - \frac{\partial x^1}{\partial q^3} \frac{\partial x^3}{\partial q^1} \tag{7.3.104}$$

$$A_2^3 = \frac{\partial x^3}{\partial q^1} \frac{\partial x^1}{\partial q^2} - \frac{\partial x^1}{\partial q^1} \frac{\partial x^3}{\partial q^2} \tag{7.3.105}$$

$$A_3^1 = \frac{\partial x^1}{\partial q^2} \frac{\partial x^2}{\partial q^3} - \frac{\partial x^2}{\partial q^2} \frac{\partial x^1}{\partial q^3} \tag{7.3.106}$$

$$A_3^2 = \frac{\partial x^1}{\partial q^3} \frac{\partial x^2}{\partial q^1} - \frac{\partial x^2}{\partial q^3} \frac{\partial x^1}{\partial q^1} \tag{7.3.107}$$

$$A_3^3 = \frac{\partial x^1}{\partial q^1} \frac{\partial x^2}{\partial q^2} - \frac{\partial x^2}{\partial q^1} \frac{\partial x^1}{\partial q^2} \tag{7.3.108}$$

For simplification to two dimensions, all derivatives $\frac{\partial x^3}{\partial q^3}$ are equal to one, and all derivatives of the form $\frac{\partial x^3}{\partial q^i}$ and $\frac{\partial x^i}{\partial q^3}$ where $i \neq 3$ are zero. x is inserted for x^1 and y is inserted for x^2 . This yields equations (7.3.109)-(7.3.112).

$$A_1^1 = \frac{\partial y}{\partial q^2} \tag{7.3.109}$$

$$A_1^2 = -\frac{\partial y}{\partial q^1} \tag{7.3.110}$$

$$A_2^1 = -\frac{\partial x}{\partial q^2} \tag{7.3.111}$$

$$A_2^2 = \frac{\partial x}{\partial q^1} \tag{7.3.112}$$

The area components are discretised using the central differences as given in equations

(7.3.60)- (7.3.64). This yields equations (7.3.113)-(7.3.116).

$$A_1^1 = \frac{y_{i,j+1} - y_{i,j-1}}{2} \quad (7.3.113)$$

$$A_1^2 = -\frac{y_{i+1,j} - y_{i-1,j}}{2} \quad (7.3.114)$$

$$A_2^1 = -\frac{x_{i,j+1} - x_{i,j-1}}{2} \quad (7.3.115)$$

$$A_2^2 = \frac{x_{i+1,j} - x_{i-1,j}}{2} \quad (7.3.116)$$

Now that the area components are accounted for, J in equation (7.3.89) needs to be defined. J is the Jacobi determinant and is given by equation (7.3.117).

$$J = \det \left(J_i^j \right) \quad (7.3.117)$$

$$= \begin{vmatrix} \frac{\partial x}{\partial q^1} & \frac{\partial x}{\partial q^2} \\ \frac{\partial y}{\partial q^1} & \frac{\partial y}{\partial q^2} \end{vmatrix} \quad (7.3.118)$$

$$= \frac{\partial x}{\partial q^1} \frac{\partial y}{\partial q^2} - \frac{\partial y}{\partial q^1} \frac{\partial x}{\partial q^2} \quad (7.3.119)$$

The derivatives in equation (7.3.119) are then discretised with central differences as given in equations (7.3.60)- (7.3.64). This yields equation (7.3.120).

$$J = \frac{1}{4} (x_{i+1,j} - x_{i-1,j}) (y_{i,j+1} - y_{i,j-1}) - \frac{1}{4} (y_{i+1,j} - y_{i-1,j}) (x_{i,j+1} - x_{i,j-1}) \quad (7.3.120)$$

Equation (7.3.89) defining g^{ij} can be written out to yield equation (7.3.121).

$$g^{ij} = \frac{\mathbf{A}^i \cdot \mathbf{A}^j}{J^2} = \frac{A_k^i \mathbf{e}_k \cdot A_l^j \mathbf{e}_l}{J^2} = \frac{A_k^i A_l^j \delta_{kl}}{J^2} = \frac{A_k^i A_k^j}{J^2} \quad (7.3.121)$$

Now all the components of g^{ij} can be written out as in equations (7.3.122)-(7.3.125).

$$g^{11} = \frac{A_k^1 A_k^1}{J^2} = \frac{A_1^1 A_1^1 + A_2^1 A_2^1}{J^2} \quad (7.3.122)$$

$$g^{21} = \frac{A_k^2 A_k^1}{J^2} = \frac{A_1^2 A_1^1 + A_2^2 A_2^1}{J^2} \quad (7.3.123)$$

$$g^{12} = \frac{A_k^1 A_k^2}{J^2} = \frac{A_1^1 A_1^2 + A_2^1 A_2^2}{J^2} \quad (7.3.124)$$

$$g^{22} = \frac{A_k^2 A_k^2}{J^2} = \frac{A_1^2 A_1^2 + A_2^2 A_2^2}{J^2} \quad (7.3.125)$$

7.3.2.3 Control Functions in the Poisson Equations

The choice of the control functions in equation 7.3.126 affects the generated grid and can be used to control the density of generated nodes around one specific point [34]. They can be taken as a constant number or found by use of relation.

$$P^i = \nabla^2 q^i \quad i = 1, 2 \quad (7.3.126)$$

Mohebbi [34] has done a comparison with different values for the control functions.

7.4 Implementation

7.4.1 Initialisation

The settings of the grid needs to be specified first. An initialisation of the coordinates q^1 and q^2 is done as shown in equations (7.4.1) and (7.4.2).

$$q1 = 0:N \quad (7.4.1)$$

$$q2 = 0:M \quad (7.4.2)$$

N is the number of points in q^1/x -direction and M is the number of points in q^2/y -direction. The dimensions of the physical domain are needed before the values of x and y at each corner point can be specified.

$$x_max = 35 \quad (7.4.3)$$

$$y_max = 2 \quad (7.4.4)$$

$$step_h = 1 \quad (7.4.5)$$

$$step_w = 5 \quad (7.4.6)$$

where x_max is L in figure 1.3, the total length of the physical domain including the step, y_max is H in figure 1.3, the total height of the physical domain including step and $step_h$ and $step_w$ are h and l in figure 1.3, the height and length of the step.

The values of x and y at each corner point in figure 7.2 are specified as in equations (7.4.7)-(7.4.18).

$$xA = 0 \quad (7.4.7) \quad yA = step_h \quad (7.4.13)$$

$$xB = 0 \quad (7.4.8) \quad yB = y_max \quad (7.4.14)$$

$$xC = x_max \quad (7.4.9) \quad yC = y_max \quad (7.4.15)$$

$$xD = x_max \quad (7.4.10) \quad yD = 0 \quad (7.4.16)$$

$$xE = step_w \quad (7.4.11) \quad yE = 0 \quad (7.4.17)$$

$$xF = step_w \quad (7.4.12) \quad yF = step_h \quad (7.4.18)$$

7.4.2 Transfinite Interpolation

Equations (7.4.19)-(7.4.24) are implemented in **MATLAB** to yield the x - and y -points in the line segments $\textcircled{A}\textcircled{B}$, $\textcircled{B}\textcircled{C}$ and $\textcircled{D}\textcircled{C}$ in figure 7.2.

$$\textcircled{A}\textcircled{B}: x_{AB} = \left(1 - \frac{q^2}{q_2^2}\right) x_A + \frac{q^2}{q_2^2} x_B \quad (7.4.19)$$

$$y_{AB} = \left(1 - \frac{q^2}{q_2^2}\right) y_A + \frac{q^2}{q_2^2} y_B \quad (7.4.20)$$

$$\textcircled{B}\textcircled{C}: x_{BC} = \left(1 - \frac{q^1}{q_1^1}\right) x_B + \frac{q^1}{q_1^1} x_C \quad (7.4.21)$$

$$y_{BC} = \left(1 - \frac{q^1}{q_1^1}\right) y_B + \frac{q^1}{q_1^1} y_C \quad (7.4.22)$$

$$\textcircled{C}\textcircled{D}: x_{DC} = \left(1 - \frac{q^2}{q_2^2}\right) x_D + \frac{q^2}{q_2^2} x_C \quad (7.4.23)$$

$$y_{DC} = \left(1 - \frac{q^2}{q_2^2}\right) y_D + \frac{q^2}{q_2^2} y_C \quad (7.4.24)$$

The line segment $\textcircled{A}\textcircled{D}$ needs to be split into the three line segments $\textcircled{A}\textcircled{F}$, $\textcircled{F}\textcircled{E}$ and $\textcircled{E}\textcircled{D}$ for this calculation. The placement of the points \textcircled{E} and \textcircled{F} in the computational domain determines how the boundary points between \textcircled{A} and \textcircled{D} are distributed between the three sub-line segments. The variables `AFpoints` and `FEpoints` specify how many points go in these respective segments out of the `N` points in total for $\textcircled{A}\textcircled{D}$. Vectors of coordinates q^1 for the line segments $\textcircled{A}\textcircled{F}$, $\textcircled{F}\textcircled{E}$ and $\textcircled{E}\textcircled{D}$ noted q_{AF}^1 , q_{FE}^1 and q_{ED}^1 were then created as shown in equations (7.4.25)-(7.4.27).

$$\text{q1AF} = 0:\text{AFpoints} \quad (7.4.25)$$

$$\text{q1FE} = 0:\text{FEpoints} \quad (7.4.26)$$

$$\text{q1ED} = 0:(\text{N}-\text{AFpoints}-\text{FEpoints}) \quad (7.4.27)$$

Note that each new vector of q^1 -coordinates line segment starts from zero. This is because the coordinates are unique to the line segment in question, as the fraction q^1/q_2^1 in equations (7.3.35) and (7.3.36) should go from 0 to 1 along the line segment. The boundary points for the line segments $\textcircled{A}\textcircled{F}$, $\textcircled{F}\textcircled{E}$ and $\textcircled{E}\textcircled{D}$ are then found by equations (7.4.28)-(7.4.33).

$$\textcircled{A}\textcircled{F}: x_{AF} = \left(1 - \frac{q_{AF}^1}{q_{AF,2}^1}\right) x_A + \frac{q_{AF}^1}{q_{AF,2}^1} x_F \quad (7.4.28)$$

$$y_{AF} = \left(1 - \frac{q_{AF}^1}{q_{AF,2}^1}\right) y_A + \frac{q_{AF}^1}{q_{AF,2}^1} y_F \quad (7.4.29)$$

$$\textcircled{F}\textcircled{E}: x_{FE} = \left(1 - \frac{q_{FE}^1}{q_{FE,2}^1}\right) x_F + \frac{q_{FE}^1}{q_{FE,2}^1} x_E \quad (7.4.30)$$

$$y_{FE} = \left(1 - \frac{q_{FE}^1}{q_{FE,2}^1}\right) y_F + \frac{q_{FE}^1}{q_{FE,2}^1} y_E \quad (7.4.31)$$

$$\textcircled{E}\textcircled{D}: x_{ED} = \left(1 - \frac{q_{ED}^1}{q_{ED,2}^1}\right) x_E + \frac{q_{ED}^1}{q_{ED,2}^1} x_D \quad (7.4.32)$$

$$y_{ED} = \left(1 - \frac{q_{ED}^1}{q_{ED,2}^1}\right) y_E + \frac{q_{ED}^1}{q_{ED,2}^1} y_D \quad (7.4.33)$$

For the calculation of the centre points, the line segment $\textcircled{A}\textcircled{D}$ must be put back together. This is done as shown in equations (7.4.34) and (7.4.35).

$$\mathbf{x}_{AD} = [\mathbf{x}_{AF} \ \mathbf{x}_{FE}(2:\text{end}-1) \ \mathbf{x}_{ED}] \quad (7.4.34)$$

$$\mathbf{y}_{AD} = [\mathbf{y}_{AF} \ \mathbf{y}_{FE}(2:\text{end}-1) \ \mathbf{y}_{ED}] \quad (7.4.35)$$

Note that since each sub-line segment go from a corner point to another, points \textcircled{E} and \textcircled{F} are overlapped. This is solved by taking only the points from position 2 to $\text{end}-1$ for the line segment $\textcircled{E}\textcircled{F}$. \mathbf{x}_{AD} and \mathbf{y}_{AD} are then N long and can be used to calculate the centre points of the domain.

Equations (7.4.36)-(7.4.37) are implemented in MATLAB to yield the points in the centre of the domain.

$$\begin{aligned} x = & \left(1 - \frac{q^1}{q_2^1}\right) x_{AB} + \frac{q^1}{q_2^1} x_{DC} + \left(1 - \frac{q^2}{q_2^2}\right) x_{AD} + \frac{q^2}{q_2^2} x_{BC} \\ & + \left(1 - \frac{q^1}{q_2^1}\right) \left(1 - \frac{q^2}{q_2^2}\right) x_A + \left(1 - \frac{q^1}{q_2^1}\right) \frac{q^2}{q_2^2} x_B + \frac{q^1}{q_2^1} \left(1 - \frac{q^2}{q_2^2}\right) x_D + \frac{q^1 q^2}{q_2^1 q_2^2} x_C \end{aligned} \quad (7.4.36)$$

$$\begin{aligned} y = & \left(1 - \frac{q^1}{q_2^1}\right) y_{AB} + \frac{q^1}{q_2^1} y_{DC} + \left(1 - \frac{q^2}{q_2^2}\right) y_{AD} + \frac{q^2}{q_2^2} y_{BC} \\ & + \left(1 - \frac{q^1}{q_2^1}\right) \left(1 - \frac{q^2}{q_2^2}\right) y_A + \left(1 - \frac{q^1}{q_2^1}\right) \frac{q^2}{q_2^2} y_B + \frac{q^1}{q_2^1} \left(1 - \frac{q^2}{q_2^2}\right) y_D + \frac{q^1 q^2}{q_2^1 q_2^2} y_C \end{aligned} \quad (7.4.37)$$

A double for loop shown below over the indices of q^1 and q^2 is used to calculate the points, where the first index i runs for the q^1 -coordinate direction and the second index j runs for the q^2 -coordinate direction.

```

1  for j = 1:length(q2)
2      for i = 1:length(q1)
3          x(j,i) = (1-q1(i)/q1(end))*xAB(j) + (q1(i)/q1(end))*xDC(j)...
4              + (1-q2(j)/q2(end))*xAD(i) + (q2(j)/q2(end))*xBC(i)...
5              - (1-q1(i)/q1(end))*(1-q2(j)/q2(end))*xA...
6              - (1-q1(i)/q1(end))*(q2(j)/q2(end))*xB...
7              - (q1(i)/q1(end))*(1-q2(j)/q2(end))*xD...
8              - (q1(i)/q1(end))*(q2(j)/q2(end))*xC;
9          y(j,i) = (1-q1(i)/q1(end))*yAB(j) + (q1(i)/q1(end))*yDC(j)...
10             + (1-q2(j)/q2(end))*yAD(i) + (q2(j)/q2(end))*yBC(i)...
11             - (1-q1(i)/q1(end))*(1-q2(j)/q2(end))*yA...
12             - (1-q1(i)/q1(end))*(q2(j)/q2(end))*yB...
13             - (q1(i)/q1(end))*(1-q2(j)/q2(end))*yD...
14             - (q1(i)/q1(end))*(q2(j)/q2(end))*yC;
15     end %for
16 end %for

```

q^1 , x_{BC} , x_{AD} , y_{BC} and y_{AD} in equations (7.4.36) and (7.4.37) are therefore indexed with i , while q^2 , x_{AB} , x_{DC} , y_{AB} and y_{DC} are indexed with j . The rest of the code is shown in appendix E.6.

The points along the boundary of the domain as found by equations (7.4.19)-(7.4.33) do not need to be inserted in the matrices x and y above. The boundary points will in addition to the centre points be inserted into the matrices by use of equations (7.4.36) and (7.4.37) since the boundary values of q^1 and q^2 are included in the for loop.

7.4.3 Elliptic Grid Generator

The discretised elliptic grid generation equation for x and y in equations (7.3.69) and (7.3.79) are implemented in MATLAB the same way as the Momentum equations (3.2.57) and (3.2.59) and solved iteratively. The discretised equations are represented on the form $Xx = b_x$ and $Yy = b_y$ and are solved using the *divided into* operator in MATLAB as described in section 2.6.

7.4.3.1 Initial guess

The initial guess is the algebraic grid obtained from the Transfinite interpolation equation.

7.4.3.2 Boundary Conditions

The source terms b_x and b_y are equal to zero from equations (7.3.69) and (7.3.79). At any of the four boundaries *east*, *west*, *north* or *south*, boundary conditions are applied. The values of x and y are known at all boundaries from the TFI grid. The known x - and y -values noted x_{edge} and y_{edge} are then multiplied with the appropriate coefficient c^x and c^y and moved to the source term as seen in equations (7.4.38) and (7.4.39).

$$c_{i,j}^x x_{i,j} + \sum c_{nb}^x x_{nb} = \sum -c_{edge}^x x_{edge} \quad (7.4.38)$$

$$c_{i,j}^y y_{i,j} + \sum c_{nb}^y y_{nb} = \sum -c_{edge}^y y_{edge} \quad (7.4.39)$$

The subscript nb symbolises all the neighbouring nodes, and c^x and c^y are given in equations (7.3.69) and (7.3.79).

7.4.3.3 Control Functions

The values of the control functions P^j can be chosen and adjusted to yield the grid with the desired qualities. $P^j = 0$ reduces the Poisson equation (7.1.9) to the Laplace equation [33].

7.4.3.4 Under-Relaxation

The solution is relaxed at the end of the iteration like shown in equations (7.4.40) and (7.4.41)[34] before x^{new} and y^{new} are passed on to the next iteration.

$$x^{new} = (1 - \alpha)x + \alpha x^\circ \quad (7.4.40)$$

$$y^{new} = (1 - \alpha)y + \alpha y^\circ \quad (7.4.41)$$

The under-relaxation factor *alpha* is set to 0.001.

7.4.3.5 Convergence Criteria

The discretised elliptic grid generation equations (7.3.69) and (7.3.79) are equal to zero, and the solution is converged when this is true.

The convergence criteria used are defined in equations (7.4.42) and (7.4.43).

$$C_x = \max(|x - x^\circ|) \quad (7.4.42)$$

$$C_y = \max(|y - y^\circ|) \quad (7.4.43)$$

x and y are obtained in the current iteration, and x° and y° are the result from the previous iteration. The limits for both C_x and C_y were set to 10^{-3} .

7.4.3.6 Code Setup

A `while` loop is set up running until the solution is converged. Global indexing is used for the entries of the matrices x and y as obtained from the TFI equation.

The area components $A_i^{(k)}$, the Jacobi determinant J and the contravariant tensor components g^{ij} are obtained from x and y at the previous iteration.

A `for` loop runs through all the points in the globally indexed vectors x and y . If a point is at a boundary of the domain, the appropriate boundary condition is applied.

The new points x^{new} and y^{new} are obtained by use of the *devided into* operator `\` in MATLAB, and they are under-relaxed before being passed on to the next iteration.

7.5 Results and Discussion

The results from the grid generation are presented and discussed in this section.

7.5.1 Transfinite Interpolation

The code `transfinite.m` was used to obtain the results for the transfinite interpolation model and is given in appendix E. Figure 7.3 shows the obtained grid by use of the transfinite interpolation equations (7.4.19) - (7.4.37). 72 nodes were used in x -direction and 22 nodes were used in y -direction. Here the points in line segment AD were split

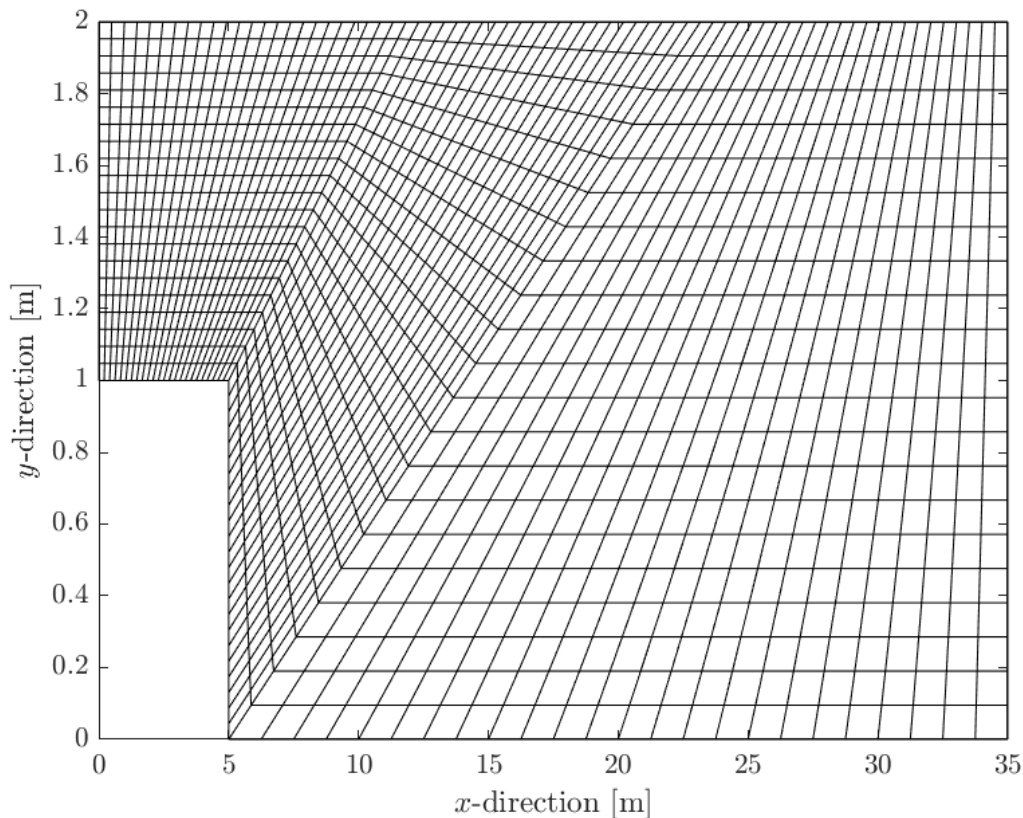


Figure 7.3: Grid obtained by transfinite interpolation.

into three segments for AF FE ED. A different ratio would have yielded more points in the line section ED which may be beneficial.

7.5.2 Elliptic Grid Generation

The code `elliptic.m` is used for the elliptic grid model, and is given in appendix E. The transfinite interpolation grid obtained from `transfinite.m` is used as an initial guess. The code does not work properly and does not yield the desired grid.

Figure 7.4 shows the elliptic grid after 100 iterations. This is around the number of iterations before the solution starts to move away from the domain to a larger extent, yielding node points outside of the domain. The solution diverges after 859 iterations. After 100 iterations, the convergence criteria are still very large.

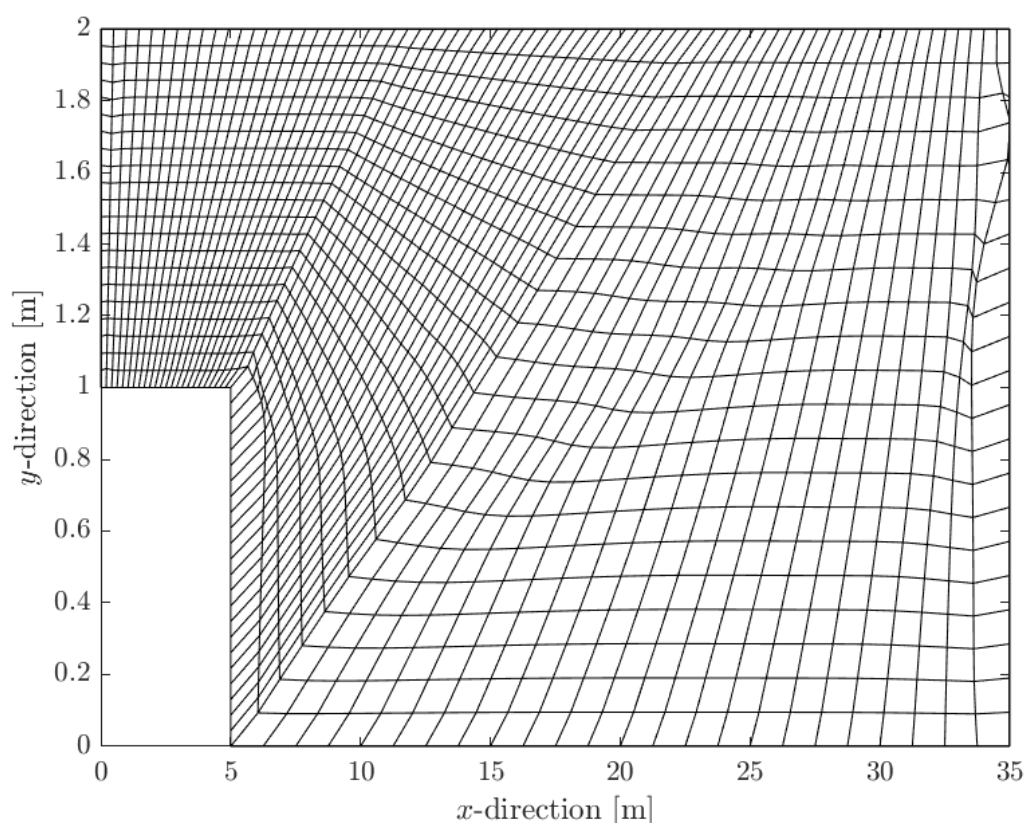


Figure 7.4: Elliptic grid after 100 iterations.

As can be seen, the grid points have started to move slightly, bending some of the lines as expected. The corner of the backwards facing step and the eastern boundary seems to be the locations where the solution is starting to fail. At the corner of the backwards facing step, the node points are starting to seep into the domain. At the eastern boundary, the second last line is starting to oscillate.

Since the model runs and produces a result, the fundamentals of the code are most likely correct. It is therefore likely that the errors in the solution are caused by a small typo in the code or by a mistake in the derivation of the elliptic grid equations. As mentioned in the theory section, there are numerical difficulties associated with the elliptic generation method [33]. There is therefore a slight chance that tuning of the parameters may yield a functioning model, but this is unlikely.

The boundary points on the southern boundary (line segment AD) are not equally spaced in the current implementation. Modifying the transfinite interpolation grid to have equally spaced points on this boundary might help with the inaccuracies around the corner.

8

Conclusion

Modelling the fluid flow with dimensionless equations makes the models more robust to choice of inlet condition, and modelling of a range of different Reynolds numbers is possible. The straight channel model and the backwards facing step models yield expected results. The results for the recirculation zones for the backwards facing step model for Reynolds numbers between 0.0001 and 400 are in agreement with results found in literature. For Reynolds numbers lower than 50, the resolution of the grid is not high enough to represent the recirculation zones. A higher resolution could not be obtained. The flow over the step has a higher magnitude v -velocity than expected, leading to sharper turns in the flow direction over the step. This is likely due to the choice of discretisation scheme, since the Upwind Differencing Scheme is prone to problems with false diffusion. The models are all sensitive to the values of the under-relaxation factors, and the factors are around magnitude 0.01 for all the two dimensional cases. The under-relaxation factors generally had to be lowered for the higher Reynolds numbers. For the models were grids of lower resolutions were possible, the under-relaxation factors could be increased.

The transfinite interpolation technique produces the algebraic grid for use when solving the fluid flow problem formulated in generalised curvilinear coordinates. The code for an elliptic grid using the algebraic grid as an initial guess does not yield the satisfactory grid, most likely due to a mistake in the discretised elliptic grid generation equation or in the code.

8.1 Recommendations for Future Work

- Repeat simulations using a higher order differencing scheme to avoid false diffusion over the backwards facing step
- Modify backwards facing step model to simulate with a higher resolution to accurately represent the recirculation zones for low Reynolds numbers
- Correctly solve the elliptic grid generation equation

- Solve the flow problem formulated in generalised curvilinear coordinates with the obtained elliptic grid

Bibliography

- [1] R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.
- [2] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics; The Finite Volume Method*. Pearson Education Limited, Essex, England, 1995.
- [3] M. C. Melaaen. *Analysis of Curvilinear Non-Orthogonal Coordinates for Numerical Calculation of Fluid Flow in Complex Geometries*. PhD thesis, Norges tekniske høgskole, Trondheim, 1990.
- [4] G. Biswas, M. Breuer, and F. Durst. Backward-facing step flows for various expansion ratios at low and moderate reynolds numbers. *Journal of Fluids Engineering*, 126(3):362–374, 5 2004.
- [5] D. Barkley, M. G. M. Gomes, and R. D. Henderson. Three-dimensional instability in flow over a backward-facing step. *Journal of fluid mechanics*, 473:167–190, 2002.
- [6] D. Ratha and A. Sarkar. Analysis of flow over backward facing step with transition. *Frontiers of Structural and Civil Engineering*, 9(1):71–81, 2015.
- [7] B. F. Armaly, F. Durst, J. C. F. Pereira, and B. Schönung. Experimental and theoretical investigation of backward-facing step flow. *Journal of fluid Mechanics*, 127:473–496, 1983.
- [8] R. J. Goldstein, V. L. Eriksen, R. M. Olson, and E. R. G. Eckert. Laminar separation, reattachment, and transition of the flow over a downstream-facing step. *Journal of Fluids Engineering*, 1970.
- [9] M.K. Denham and M. A. Patrick. Laminar flow over a downstream-facing step in a two-dimensional flow channel. *Trans. Inst. Chem. Engrs*, 52(4):361–367, 1974.
- [10] E. Erturk. Numerical solutions of 2-d steady incompressible flow over a backward-facing step, part i: High reynolds number solutions. *Computers & Fluids*, 37(6): 633–655, 2008.
- [11] I. E. Barton. The entrance effect of laminar flow over a backward-facing step geometry. *International Journal for Numerical Methods in Fluids*, 25(6):633–644, 1997.
- [12] T. Lee and D. Mateescu. Experimental and numerical investigation of 2-d backward-facing step flow. *Journal of Fluids and Structures*, 12(6):703–716, 1998.
- [13] J. H. Nie and B. F. Armaly. Reverse flow regions in three-dimensional backward-

- facing step flow. *International journal of heat and mass transfer*, 47(22):4713–4720, 2004.
- [14] E. Manger. *Modelling and simulation of gas/solids flow in curvilinear coordinates*. PhD thesis, Telemark College, 1996.
- [15] A. Klavenes. The finite volume method for modelling steady, laminar flow, 2019. Specialisation project.
- [16] H.A. Jakobsen. *Chemical Reactor Modeling; Multiphase Reactive Flows*. Springer, Switzerland, 2014.
- [17] C. J. Geankoplis. *Transport Processes and Separation Principles*. Pearson, India, 2015.
- [18] F. P. Incropera, A. S. Lavine, T. L. Bergman, and D. P. DeWitt. *Fundamentals of heat and mass transfer*. Wiley, 2007.
- [19] Y. Cengel. *Heat and mass transfer: fundamentals and applications*. McGraw-Hill Higher Education, 2014.
- [20] D. J. Wollkind and B. J. Dichone. *Comprehensive Applied Mathematical Modeling in the Natural and Engineering Sciences*. Springer, 2017.
- [21] R. B. Bird, W. E. Stewart, E. N. Lightfoot, and D. J. Klingenberg. *Introductory transport phenomena*, volume 1. Wiley New York, 2015.
- [22] S. V. Patankar and D. B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15(10):1787–1806, 1972.
- [23] M. Peric. *A Finite Volume Method for the Prediction of Three-Dimensional Fluid Flow in Complex Ducts*. PhD thesis, University of London, Imperial College, 1985.
- [24] M. Peric, R. Kessler, and G. Scheuerer. Comparison of finite-volume numerical methods with staggered and colocated grids. *Computers & Fluids*, 16(4):389–403, 7 1988.
- [25] S. Patankar. *Numerical heat transfer and fluid flow*. Taylor & Francis, 2018.
- [26] R. Salvi. *Navier-Stokes equations: theory and numerical methods*, volume 388. CRC Press, 1998.
- [27] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley & Sons, Hoboken, 2006.
- [28] S. Attaway. *Matlab: a practical introduction to programming and problem solving*. Butterworth-Heinemann, 2013.
- [29] R. W. Fox, A.T. McDonald, and P.J. Pitchard. *Introduction to Fluid Mechanics*. John Wiley & Sons, Inc, 2004.
- [30] R. A. Adams and C. Essex. *Calculus 2: selected chapters from: Calculus: a complete course*. Pearson Education Limited, Essex, 2013.
- [31] A. Blackman, L. R. Gahan, G. H. Aylward, and T. J. V. Findlay. *Aylward and Findlay's SI Chemical Data*. John Wiley & Sons, Milton, 2014.
- [32] T. Cebeci, J. P. Shao, F. Kafyeke, and E. Laurendeau. *Computational fluid dynamics for engineers*. Springer Berlin Heidelberg, 2005.

- [33] J. Blazek. *Computational fluid dynamics: principles and applications*. Butterworth-Heinemann, 2015.
- [34] F. Mohebbi. *Optimal shape design based on body-fitted grid generation*. PhD thesis, University of Canterbury. Mechanical Engineering, 2014.
- [35] W. J. Gordon and C. A. Hall. Construction of curvilinear co-ordinate systems and applications to mesh generation. *International Journal for Numerical Methods in Engineering*, 7(4):461–477, 1973.
- [36] J. F. Thompson, Z. U. A. Warsi, and C. W. Mastin. *Numerical Grid Generation, Foundations and Applications*. Elsevier Science Publishing Co., New York, 1985.
- [37] T. Cebeci, J. R. Shao, F. Kafyeke, and E. Laurendeau. *Computational Fluid Dynamics for Engineers*. Horizons Publishing, Long Beach, 2005.
- [38] R. B. Stull. *An introduction to boundary layer meteorology*. Springer, Netherlands, 1988.
- [39] M. C. Melaaen. Analysis of fluid flow in constricted tubes and ducts using body-fitted non-staggered grids. *International Journal for Numerical Methods in Fluids*, 15(8):895–923, 1991.
- [40] F. Irgens. *Tensor Analysis*. Springer, Cham, 2019.
- [41] F. S. Roberts and B. Tesman. *Applied Combinatorics*. CRC Press, Boca Raton, 2009.
- [42] D. E. Neuenschwander. *Tensor calculus for physics : a concise guide*. Johns Hopkins University Press, Baltimore, 2014.

Appendix

Table of Contents

A	Governing Equations	131
A.1	The Mass Based Equation of Continuity	131
A.2	The Equation of Motion	131
A.3	Other Equations and Theorems	132
B	One Dimensional Model	133
B.1	Discretisation	133
B.2	Boundary Conditions	137
B.3	Implementation	138
B.4	Results and Discussion	139
C	Detailed Two Dimensional Discretisation	143
C.1	Continuity Equation	143
C.2	Momentum equation	144
C.3	SIMPLE-Equations	152
D	Elliptic Grid Generation in Three Dimensions	157
D.1	Elliptic Grid Generation Equation	157
D.2	Expression for the Contravariant Tensor Components	158
E	MATLAB Code	161
E.1	Codes Sorted by Model	161
E.2	List of MATLAB parameters	162
E.3	One Dimensional Straight Channel	172
E.4	Two Dimensional Straight Channel	176
E.4.1	Codes	176
E.4.1.1	channel_2D.m	176
E.4.1.2	plot_2D.m	190
E.5	Backwards Facing Step Model	194
E.5.1	Constant Inlet Velocity	194
E.5.1.1	channel_BFS.m	194
E.5.1.2	BFS_u_velocity.m	199
E.5.1.3	BFS_v_velocity.m	205
E.5.1.4	BFS_pressurecorrection.m	211
E.5.1.5	plot_BFS.m	215
E.5.1.6	plotVelocityQuiver.m	217
E.5.1.7	plotColoredQuiver.m	220
E.5.1.8	plotVelocityCorrection.m	222
E.5.1.9	plotIntermediates.m	223
E.5.1.10	plot_BFS_iterations.m	225
E.5.1.11	plotVelInts_BFS_iterations.m	227

E.5.1.12	isWide.m	230
E.5.1.13	getRowNumber.m	231
E.5.1.14	getRowUnder.m	231
E.5.1.15	getRowOver.m	231
E.5.1.16	global2matrix.m	231
E.5.2	Parabolic Inlet Velocity Profile	232
E.5.2.1	channel_BFS_parabolic.m	232
E.5.2.2	BFS_u_velocity_parabolic.m	236
E.5.2.3	BFS_v_velocity_parabolic.m	242
E.5.2.4	BFS_pressurecorrection_parabolic.m	247
E.5.2.5	plotColoredQuiver_parabolic.m	251
E.6	Grid Generation	253
E.6.1	Codes	254
E.6.1.1	transfinite.m	254
E.6.1.2	elliptic.m	255
E.6.1.3	getCol.m	261
E.6.1.4	getRow.m	261
E.6.1.5	matrix2global.m	262
E.6.1.6	global2matrix.m	262



Governing Equations

A.1 The Mass Based Equation of Continuity

The continuity equation in vector form is shown in equation A.1.1 [16].

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \quad (\text{A.1.1})$$

A.2 The Equation of Motion

The momentum equation in vector form is shown in equation A.2.1 [16].

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \mathbf{u}) = -\nabla p - \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} \quad (\text{A.2.1})$$

The x -component of the two dimensional momentum equation is shown in equation A.2.2.

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}(\rho u u) + \frac{\partial}{\partial y}(\rho v u) = -\frac{\partial p}{\partial x} - \frac{\partial \sigma_{xx}}{\partial x} - \frac{\partial \sigma_{yx}}{\partial y} + \rho g_x \quad (\text{A.2.2})$$

The y -component of the two dimensional momentum equation is shown in equation A.2.3.

$$\frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x}(\rho u v) + \frac{\partial}{\partial y}(\rho v v) = -\frac{\partial p}{\partial y} - \frac{\partial \sigma_{xy}}{\partial x} - \frac{\partial \sigma_{yy}}{\partial y} + \rho g_y \quad (\text{A.2.3})$$

The stress tensors $\boldsymbol{\sigma}$ for two dimensional systems are shown in equations A.2.4, A.2.5 and A.2.6.

$$\sigma_{xx} = -\mu \left[2 \frac{\partial u}{\partial x} - \frac{2}{3} (\nabla \cdot \mathbf{v}) \right] \quad (\text{A.2.4})$$

$$\sigma_{yy} = -\mu \left[2 \frac{\partial v}{\partial y} - \frac{2}{3} (\nabla \cdot \mathbf{v}) \right] \quad (\text{A.2.5})$$

$$\sigma_{xy} = \sigma_{yx} = -\mu \left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right] \quad (\text{A.2.6})$$

A.3 Other Equations and Theorems

Gauss' theorem

Gauss' theorem is shown in equation A.3.1 [2].

$$\int_{CV} \nabla \cdot \phi \, dV = \int_A \mathbf{n} \cdot \phi \, dA \quad (\text{A.3.1})$$

where \mathbf{n} is normal to ϕ .

The Fundamental Theorem of Calculus

The Fundamental Theorem of Calculus is shown in equation A.3.2 [30].

$$\int_a^b \frac{d}{dx} f(x) dx = f(b) - f(a) \quad (\text{A.3.2})$$

B

One Dimensional Model

This chapter includes all discretisation, properties of the flow and results for the one dimensional straight channel model. The one dimensional model was developed for learning how the Finite Volume method is used for governing fluid flow equations. The expected result is a linear profile for both the velocity and pressure.

B.1 Discretisation

In this section, the discretisation of the Continuity equation, the Momentum equation and the SIMPLE-equations in one dimension is given. The steps are explained in short comments.

Continuity Equation

Continuity equation with the transient term deleted is integrated over the control volume CV . Gauss' theorem in equation A.3.1 is applied, and the resulting surface integral is split into the two control volume surfaces e and w .

$$\begin{aligned}\int_{CV} \nabla \cdot (\rho \mathbf{u}) dV &= 0 \\ \int_A \mathbf{n} \cdot (\rho \mathbf{u}) dA &= 0 \\ \int_{A_e} \mathbf{n} \cdot (\rho \mathbf{u}) dA_e + \int_{A_w} \mathbf{n} \cdot (\rho \mathbf{u}) dA_w &= 0 \\ (\rho u A)_e - (\rho u A)_w &= 0\end{aligned}$$

The convective mass flux per unit area F is

$$F^c = \rho u$$

and is defined at the pressure node cell faces which coincide with the velocity nodes so that no approximation of F^c is needed. The Continuity equation becomes:

$$F_e A_e - F_w A_w = 0$$

Momentum Equation

Left Side

$$\nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \mathbf{RHS}$$

$$\int_{CV} \nabla \cdot (\rho \mathbf{u}\mathbf{u}) dV = \mathbf{RHS}$$

$$\int_A \mathbf{n} \cdot (\rho \mathbf{u}\mathbf{u}) dA = \mathbf{RHS}$$

$$\int_{A_e} \mathbf{n} \cdot (\rho \mathbf{u}\mathbf{u}) dA_e + \int_{A_w} \mathbf{n} \cdot (\rho \mathbf{u}\mathbf{u}) dA_w = \mathbf{RHS}$$

$$(\rho u u A)_e - (\rho u u A)_w = \mathbf{RHS}$$

The upwind differencing scheme is used for one of the velocity terms. The other term is used with the Continuity equation to determine the flow direction.

For eastgoing flow:

$$\phi_w = \phi_W \quad \text{and} \quad \phi_e = \phi_P$$

For westgoing flow:

$$\phi_w = \phi_P \quad \text{and} \quad \phi_e = \phi_E$$

Left hand side of the momentum equation for eastgoing flow:

$$F_e A_e u_P - F_w A_w u_W = \mathbf{RHS}$$

Left hand side of the momentum equation for westgoing flow:

$$F_e A_e u_E - F_w A_w u_P = \mathbf{RHS}$$

Result:

$$\begin{aligned} & \left(\max(F_w A_w, 0) + \max(0, -F_e A_e) + F_e A_e - F_w A_w \right) u_P \\ & \quad - \max(0, -F_e A_e) u_E - \max(F_w A_w, 0) u_W = \mathbf{RHS} \end{aligned}$$

$$\begin{aligned} & \left(\max(F_w A_w, 0) + \max(0, -F_e A_e) + F_e A_e - F_w A_w \right) u_i \\ & \quad - \max(0, -F_e A_e) u_{i+1} - \max(F_w A_w, 0) u_{i-1} = \mathbf{RHS} \end{aligned}$$

Right Side

$\nabla \cdot \mathbf{u}$ is zero for incompressible flow from the Continuity equation. This means that $\frac{\partial u}{\partial x}$ is zero for the one dimensional problem, but is kept for practice, since there would be no equation to solve if the term is not kept.

$$\begin{aligned} \text{LHS} &= -\nabla p - \sum_i \frac{\partial \sigma_i}{\partial x_i} \\ \text{LHS} &= -\nabla p - \frac{\partial \sigma_x}{\partial x} \\ \text{LHS} &= \mathbf{e}_x \cdot \left(-\nabla p - \frac{\partial \sigma_x}{\partial x} \right) \\ \text{LHS} &= -\frac{\partial p}{\partial x} - \frac{\partial \sigma_{xx}}{\partial x} \\ \text{LHS} &= -\frac{\partial p}{\partial x} - \frac{\partial}{\partial x} \left(-\mu \left[2 \frac{\partial u}{\partial x} - \frac{2}{3} (\nabla \cdot \mathbf{u}) \right] \right) \\ \text{LHS} &= -\frac{\partial p}{\partial x} - \frac{\partial}{\partial x} \left(-2\mu \frac{\partial u}{\partial x} \right) \\ \text{LHS} &= -\int_{CV} \frac{\partial p}{\partial x} dV - \int_A \int_{\delta x} \frac{\partial}{\partial x} \left(-2\mu \frac{\partial u}{\partial x} \right) dA dx \\ \text{LHS} &= -\frac{\partial p}{\partial x} \Delta V - \int_{\delta x} \frac{\partial}{\partial x} \left(-2\mu \frac{\partial u}{\partial x} \right) A dx \\ \text{LHS} &= -\frac{\partial p}{\partial x} \Delta V - \left(-2\mu \frac{\partial u}{\partial x} A \right)_e + \left(-2\mu \frac{\partial u}{\partial x} A \right)_w \\ \text{LHS} &= -\frac{\partial p}{\partial x} \delta x A + 2\mu \left(\frac{\partial u}{\partial x} A \right)_e - 2\mu \left(\frac{\partial u}{\partial x} A \right)_w \end{aligned}$$

The derivatives are approximated using central differences:

$$\begin{aligned} \left. \frac{\partial p}{\partial x} \right|_i &= \frac{p_I - p_{I-1}}{\delta x} \\ \left. \frac{\partial u}{\partial x} \right|_e &= \frac{u_{i+1} - u_i}{\delta x} \\ \left. \frac{\partial u}{\partial x} \right|_w &= \frac{u_i - u_{i-1}}{\delta x} \end{aligned}$$

The convective mass flux per unit area F and the diffusion conductance D :

$$F = \rho u \qquad D = \frac{\mu}{\delta x}$$

F at the velocity cell faces is approximated with linear interpolation:

$$F_e = \rho \frac{u_i + u_{i+1}}{2} \qquad F_w = \rho \frac{u_{i-1} + u_i}{2}$$

F_e and F_w are taken as known from the previous iteration. A_e and A_w are equal and are noted A . D_e and D_w are then noted D since all the node distances are equal. Inserting the central differences as well as F and D into the right side of the equation:

$$\text{LHS} = -\left(\frac{p_I - p_{I-1}}{\delta x}\right) \delta x A + 2\mu A \left(\frac{u_{i+1} - u_i}{\delta x}\right) - 2\mu A \left(\frac{u_i - u_{i-1}}{\delta x}\right)$$

$$\text{LHS} = -(p_I - p_{I-1}) A + \frac{2\mu A}{\delta x} (u_{i+1} - u_i) - \frac{2\mu A}{\delta x} (u_i - u_{i-1})$$

$$\text{LHS} = -(p_I - p_{I-1}) A + 2D_e A (u_{i+1} - u_i) - 2D_w A (u_i - u_{i-1})$$

$$\text{LHS} = 2D_e A u_{i+1} - 2D_e A u_i - 2D_w A u_i + 2D_w A u_{i-1} - (p_I - p_{I-1}) A$$

$$\text{LHS} = (-2D_e A - 2D_w A) u_i + 2D_e A u_{i+1} + 2D_w A u_{i-1} - (p_I - p_{I-1}) A$$

Final Discretised Momentum Equation

$$\begin{aligned} & \left(4AD + \max(F_w A, 0) + \max(0, -F_e A) + F_e A - F_w A\right) u_i + \\ & \left(-2AD - \max(0, -F_e A)\right) u_{i+1} + \left(-2AD - \max(F_w A, 0)\right) u_{i-1} \\ & \hspace{15em} = -A(p_I - p_{I-1})_{cs} \end{aligned}$$

Coefficient form

$$a_i u_i + a_{i-1} u_{i-1} + a_{i+1} u_{i+1} = b_i$$

with

$$a_i = -a_{i-1} - a_{i+1} + F_e A - F_w A$$

$$a_{i+1} = -2AD - \max(0, -F_e A)$$

$$a_{i-1} = -2AD - \max(F_w A, 0)$$

$$b_i = -A(p_I - p_{I-1})$$

SIMPLE-Equations

Velocity Correction Equation

The Momentum equation with the variables replaced with their "guessed" variables labelled * are subtracted from the Momentum equation. u^* is the velocity obtained from the Momentum equation earlier in the solution algorithm, and the guessed pressure p° is the pressure at the previous iteration. The velocity corrections are then omitted for all the neighbouring nodes.

$$\begin{aligned} a_i(u_i - u_i^*) + a_{i-1}(u_{i-1} - u_{i-1}^*) + a_{i+1}(u_{i+1} - u_{i+1}^*) = \\ \left(- (p_I - p_{I-1}) + (p_I^* - p_{I-1}^*)\right) A + \cancel{b_i} \end{aligned}$$

$$\begin{aligned} a_i(u_i - u_i^*) + a_{i-1}(u_{i-1} - u_{i-1}^*) + a_{i+1}(u_{i+1} - u_{i+1}^*) = \\ \left(- p_I + p_{I-1} + p_I^* - p_{I-1}^*\right) A \end{aligned}$$

$$a_i(u_i - u_i^*) + \cancel{a_{i-1} u_{i-1}^*} + \cancel{a_{i+1} u_{i+1}^*} \xrightarrow{\text{omit}} = - (p_I' - p_{I-1}') A$$

$$u_i = u_i^* - \frac{A}{a_i^{centre}} (p'_I - p'_{I-1})$$

Pressure Correction Equation

The pressure correction equation is obtained from the continuity equation, by inserting the velocity correction equation for unknown velocity nodes. The "guessed" velocity u^* is obtained from the Momentum equation.

$$\begin{aligned} \rho A \left(u_{i+1}^* - \frac{A}{a_{i+1}^{centre}} (p'_{I+1} - p'_I) \right) - \rho A \left(u_i^* - \frac{A_i}{a_i^{centre}} (p'_I - p'_{I-1}) \right) &= 0 \\ \rho A u_{i+1}^* - \frac{\rho A^2}{a_{i+1}^{centre}} (p'_{I+1} - p'_I) - \rho A u_i^* + \frac{\rho A^2}{a_i^{centre}} (p'_I - p'_{I-1}) &= 0 \\ -\frac{\rho A^2}{a_{i+1}^{centre}} (p'_{I+1} - p'_I) + \frac{\rho A^2}{a_i^{centre}} (p'_I - p'_{I-1}) + \rho A u_{i+1}^* - \rho A u_i^* &= 0 \\ p'_I \frac{\rho A^2}{a_{i+1}^{centre}} - p'_{I+1} \frac{\rho A^2}{a_{i+1}^{centre}} + p'_I \frac{\rho A^2}{a_i^{centre}} - p'_{I-1} \frac{\rho A^2}{a_i^{centre}} + \rho A u_{i+1}^* - \rho A u_i^* &= 0 \end{aligned}$$

Pressure correction equation:

$$p'_I \nu_I + p'_{I+1} \nu_{I+1} + p'_{I-1} \nu_{I-1} = \beta_I$$

Coefficients:

$$\nu_I = -\nu_{I+1} - \nu_{I-1} \quad (\text{B.1.1})$$

$$\nu_{I+1} = -\frac{\rho A^2}{a_{i+1}^{centre}} \quad (\text{B.1.2})$$

$$\nu_{I-1} = -\frac{\rho A^2}{a_i^{centre}} \quad (\text{B.1.3})$$

$$\beta_I = -\rho A u_{i+1}^* + \rho A u_i^* \quad (\text{B.1.4})$$

B.2 Boundary Conditions

Inlet

In the Momentum equation, the western node is the known inlet velocity u_{in}

$$a_i u_i + a_{i+1} u_{i+1} = b_i$$

with

$$a_i = -a_{i-1} - a_{i+1} + F_e A - F_w A + 2AD + \max(F_w A, 0)$$

$$a_{i+1} = -2AD - \max(0, -F_e A)$$

$$b_i = -A(p_I - p_{I-1}) + (2AD + \max(F_w A, 0))u_{in}$$

In the pressure correction equation, the western node is the known inlet velocity u_{in} which can be inserted directly during the derivation of the equation. No link is created for the western node.

$$\rho A \left(u_{i+1}^* - \frac{A}{a_{i+1}^{centre}} (p'_{I+1} - p'_I) \right) - \rho A u_{in} = 0$$

Rearranged, this yields

$$\nu_I p'_I + \nu_{I+1} p'_{I+1} = \beta_I$$

with

$$\begin{aligned}\nu_I &= -\nu_2 \\ \nu_{I+1} &= -\frac{\rho A^2}{a_{I+1}^{centre}} \\ \beta_I &= -\rho A u_{i+1}^* + \rho u_{in}\end{aligned}$$

Outlet

At the outlet the pressure is known, and the eastern velocity coefficient $a_E = aN + 1$ in the Momentum equation is set equal to zero to break the connection. This yields

$$a_i u_i + a_{i-1} u_{i-1} = b_i$$

with

$$\begin{aligned}a_i &= -a_{i-1} + F_e A - F_w A \\ a_{i-1} &= -2AD - \max(F_w A, 0) \\ b_i &= -A(p_{out} - p_{I-1})\end{aligned}$$

F_e is set to be equal to F_w .

In the pressure correction equation, the pressure correction at the eastern known is zero because the velocity is known. This yields

$$p'_I \nu_I + p'_{I-1} \nu_{I-1} = \beta_I$$

with

$$\begin{aligned}\nu_I &= \frac{\rho A^2}{a_{i+1}^{centre}} - \nu_{I-1} \\ \nu_{I-1} &= -\frac{\rho A^2}{a_i^{centre}} \\ \beta_I &= -A \rho u_{i+1}^* + A \rho u^* F_i\end{aligned}$$

B.3 Implementation

Properties of the Flow and the Domain

The modelled fluid is water and the fluid properties are taken to be constant with the values given in equation (4.1.1). Gravity is assumed to be effective in y - or z -direction and is therefore not modelled in the one-dimensional case.

The channel is taken to be 3 m long. The values for the known inlet velocity and the outlet pressure are

$$u_{in} = 1 \cdot 10^{-3} \quad p_{out} = 1 \cdot 10^5 \quad (\text{B.3.1})$$

$$\alpha_u = 1 \quad \alpha_p = 0.05 \quad (\text{B.3.2})$$

Initial Guesses

The initial u -velocity and pressure are both set to a constant value across the domain. The initial guesses are shown in equation (B.3.3).

$$u^\circ = 1.5 \cdot 10^{-3} \left[\text{m/s} \right] \text{ for all } u \quad p^\circ = 1.5 \cdot 10^5 \left[\text{Pa} \right] \text{ for all } p \quad (\text{B.3.3})$$

Convergence criteria

The convergence criteria used are

$$C_1 < 10^{-6} \quad (\text{B.3.4})$$

$$C_3 < 10^{-6} \quad (\text{B.3.5})$$

$$C_4 < 10^{-6} \quad (\text{B.3.6})$$

The definitions of C_1 , C_3 and C_4 are given in section 4.7. The convergence criteria C_1 and C_4 have been normalised with respect to the inlet velocity u_{in} for the one dimensional model.

B.4 Results and Discussion

The results for the one dimensional model are given in this section. The one dimensional model is not made dimensionless because it worked with the desired Reynolds number.

Table B.1 shows the convergence times and number of iterations needed to solve the one dimensional model.

N	Iterations	Time
10	972	1 sec
50	984	2 sec
100	947	3 sec
400	3202	36 sec

Table B.1: Different convergence times for different numbers of computational nodes for the one dimensional model.

The maximum amount of node points with these settings is approximately 415 node points.

Figure B.1 shows the one-dimensional u -velocity profile, figure B.2 shows the pressure profile and figure B.3 shows the pressure correction, all with 400 computational points. Note that the scale is 10^{-8} Pa, and that the order of magnitude of the pressure in figure B.2 is 10^5 . As can be seen, the pressure correction goes to zero towards the outlet. The known outlet pressure is the next node outside of the domain and not plotted in figures in figure B.2. Therefore the exact point where the pressure correction is zero is not included in figure B.3. The velocity and pressure profiles are both flat and equal to the known value at the inlet or outlet. This is expected, since the density is constant and the gradient $\frac{\partial u}{\partial x}$ must then be zero from Continuity. This means that the velocity is constant over the whole domain, and as a consequence of this the pressure is constant also. The pressure correction is close to zero across the whole domain which is the case when the Continuity equation is fulfilled and convergence is reached.

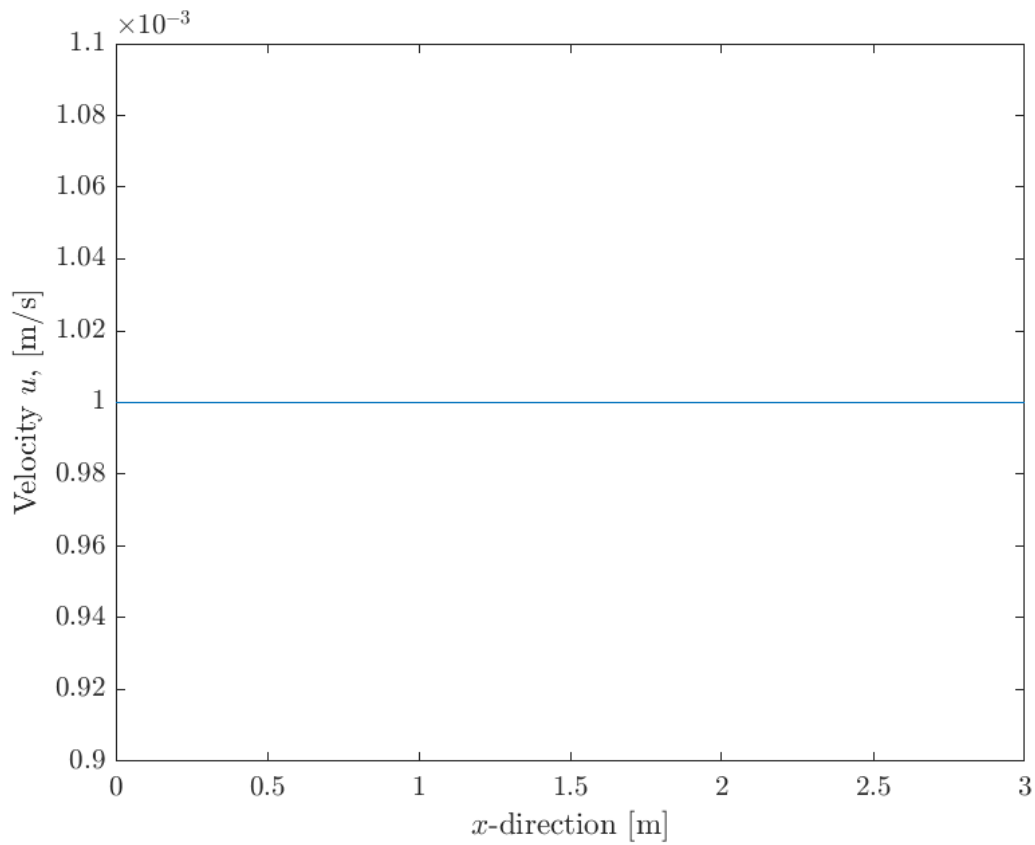


Figure B.1: Velocity profile for the one dimensional model.

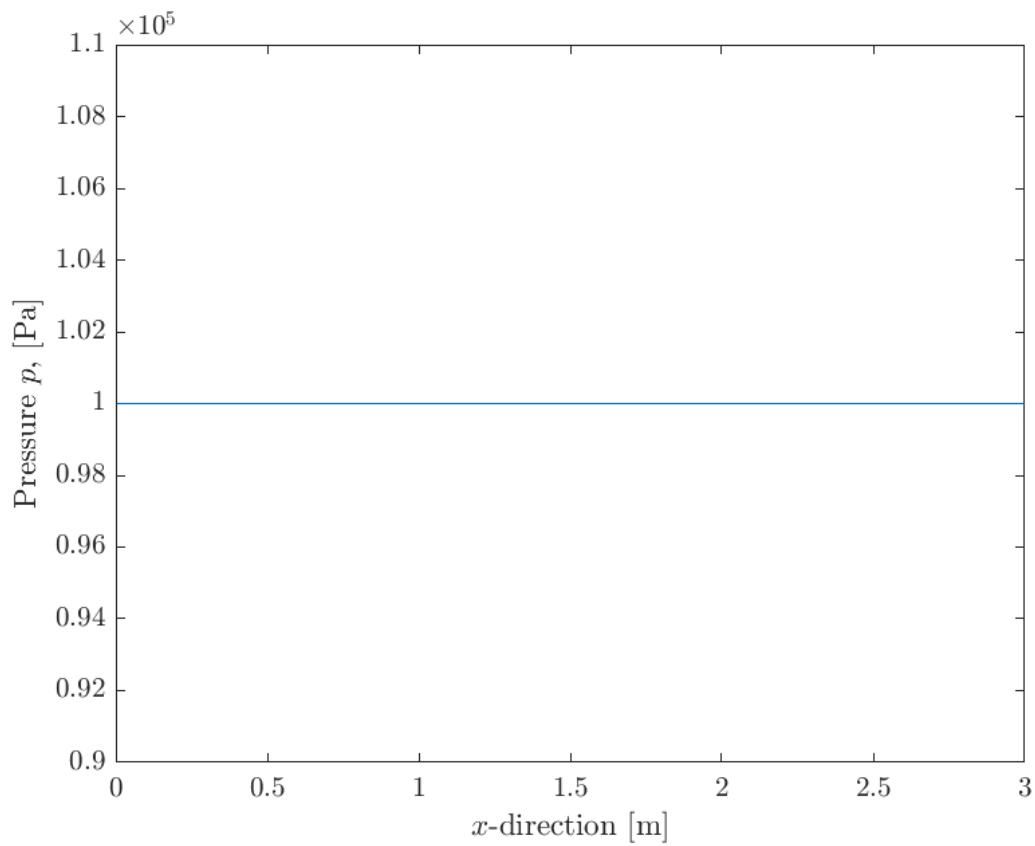


Figure B.2: Pressure profile for the one dimensional model.

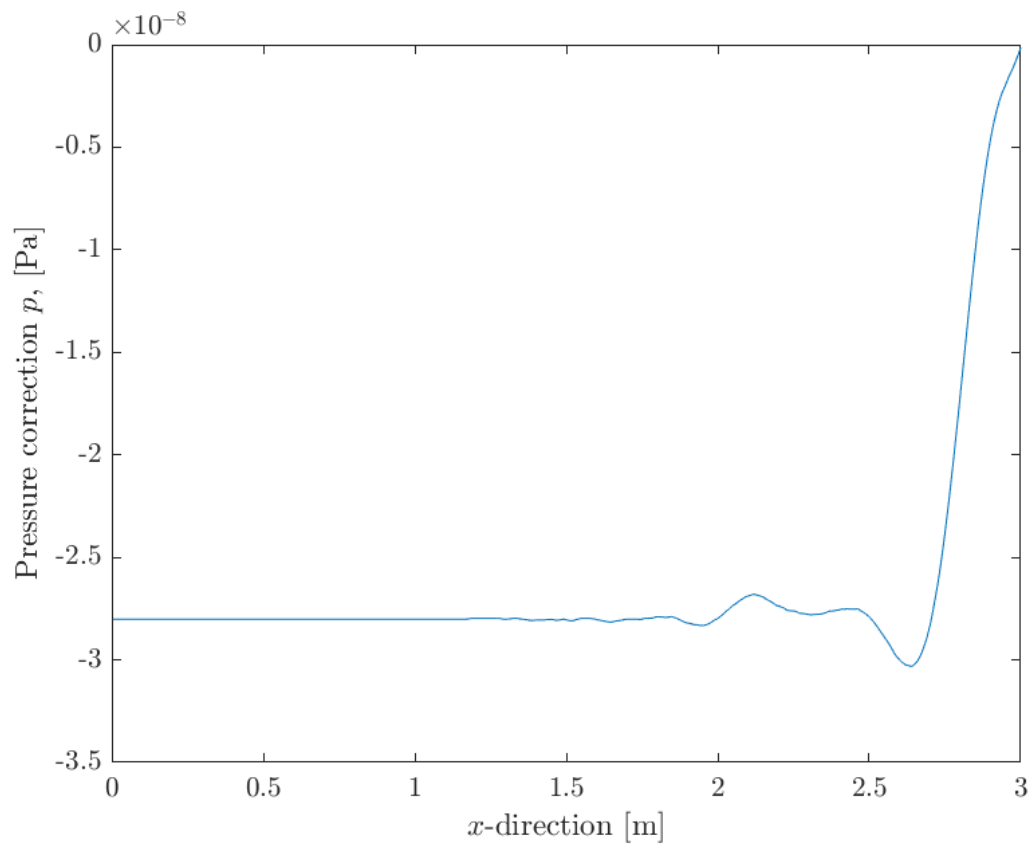


Figure B.3: Pressure correction for the one dimensional model.

C

Detailed Two Dimensional Discretisation

In this chapter, some supplements to the discretisation in the main document are given. The discretisation of the Continuity and Momentum equations as given in section 3 are repeated with all the intermediate steps included.

C.1 Continuity Equation

The transient term is neglected. The equation is integrated over the control volume CV , and Gauss theorem in equation (A.3.1) is applied. u is the x -velocity component, v is the y -velocity component.

$$\begin{aligned}\nabla \cdot (\rho \mathbf{u}) &= 0 \\ \int_{CV} \nabla \cdot (\rho \mathbf{u}) dV &= 0 \\ \int_A \mathbf{n} \cdot (\rho \mathbf{u}) dA &= 0 \\ \int_{A_{x,e}} \rho \mathbf{e}_x \cdot \mathbf{u} dA + \int_{A_{x,w}} \rho (-\mathbf{e}_x) \cdot \mathbf{u} dA + \int_{A_{y,n}} \rho \mathbf{e}_y \cdot \mathbf{u} dA + \int_{A_{y,s}} \rho (-\mathbf{e}_y) \cdot \mathbf{u} dA &= 0 \\ \int_{A_{x,e}} \rho u dA - \int_{A_{x,w}} \rho u dA + \int_{A_{y,n}} \rho v dA - \int_{A_{y,s}} \rho v dA &= 0 \\ \rho u_e A_{x,e} - \rho u_w A_{x,w} + \rho v_n A_{y,n} - \rho v_s A_{y,s} &= 0\end{aligned}$$

C.2 Momentum equation

The transient term is neglected, and the vector form Momentum equation is then

$$\nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p - \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g}$$

Left Hand Side

The equation is integrated over the control volume CV , and Gauss theorem in equation (A.3.1) is applied. Taking the dot product with the unit vector \mathbf{e}_x or \mathbf{e}_y yields the component x and y -components of the equation. u is the x -velocity component, v is the y -velocity component.

$$\nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \text{RHS}$$

$$\int_{CV} \nabla \cdot (\rho \mathbf{u}\mathbf{u}) dV = \text{RHS}$$

$$\int_A \mathbf{n} \cdot (\rho \mathbf{u}\mathbf{u}) dA = \text{RHS}$$

$$\int_{A_{x,e}} \mathbf{e}_x \cdot \rho \mathbf{u}\mathbf{u} dA + \int_{A_{x,w}} -\mathbf{e}_x \cdot \rho \mathbf{u}\mathbf{u} dA + \int_{A_{y,n}} \mathbf{e}_y \cdot \rho \mathbf{u}\mathbf{u} dA + \int_{A_{y,s}} -\mathbf{e}_y \cdot \rho \mathbf{u}\mathbf{u} dA = \text{RHS}$$

$$\int_{A_{x,e}} \rho u\mathbf{u} dA - \int_{A_{x,w}} \rho u\mathbf{u} dA + \int_{A_{y,n}} \rho v\mathbf{u} dA - \int_{A_{y,s}} \rho v\mathbf{u} dA = \text{RHS}$$

$$\rho (u\mathbf{u})_e A_{x,e} - \rho (u\mathbf{u})_w A_{x,w} + \rho (v\mathbf{u})_n A_{y,n} - \rho (v\mathbf{u})_s A_{y,s} = \text{RHS}$$

x -component

$$\mathbf{e}_x \cdot \left(\rho (u\mathbf{u})_e A_{x,e} - \rho (u\mathbf{u})_w A_{x,w} + \rho (v\mathbf{u})_n A_{y,n} - \rho (v\mathbf{u})_s A_{y,s} \right) = \text{RHS}$$

$$\rho (uu)_e A_{x,e} - \rho (uu)_w A_{x,w} + \rho (vu)_n A_{y,n} - \rho (vu)_s A_{y,s} = \text{RHS}$$

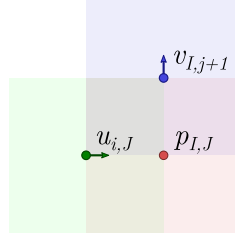
$$F_{x,e} u_e A_{x,e} - F_{x,w} u_w A_{x,w} + F_{x,n} u_n A_{y,n} - F_{x,s} u_s A_{y,s} = \text{RHS}$$

y -component

$$\mathbf{e}_y \cdot \left(\rho (u\mathbf{u})_e A_{x,e} - \rho (u\mathbf{u})_w A_{x,w} + \rho (v\mathbf{u})_n A_{y,n} - \rho (v\mathbf{u})_s A_{y,s} \right) = \text{RHS}$$

$$\rho (uv)_e A_{x,e} - \rho (uv)_w A_{x,w} + \rho (vv)_n A_{y,n} - \rho (vv)_s A_{y,s} = \text{RHS}$$

$$F_{y,e} v_e A_{x,e} - F_{y,w} v_w A_{x,w} + F_{y,n} v_n A_{y,n} - F_{y,s} v_s A_{y,s} = \text{RHS}$$

Upwind Differencing**Positive x -flow, Positive y -flow**

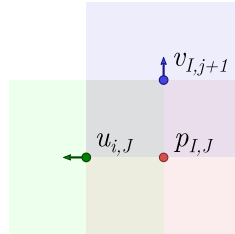
$$\begin{aligned}
 u_e &= u_P \text{ and } u_w = u_W \\
 u_n &= u_P \text{ and } u_s = u_S \\
 v_e &= v_P \text{ and } v_w = v_W \\
 v_n &= v_P \text{ and } v_s = v_S
 \end{aligned}$$

 x -component is:

$$F_{x,e}u_P A_{x,e} - F_{x,w}u_W A_{x,w} + F_{y,n}u_P A_{y,n} - F_{y,s}u_S A_{y,s} = \mathbf{RHS}$$

 y -component is:

$$F_{x,e}v_P A_{x,e} - F_{x,w}v_W A_{x,w} + F_{y,n}v_P A_{y,n} - F_{y,s}v_S A_{y,s} = \mathbf{RHS}$$

Negative x -flow, Positive y -flow

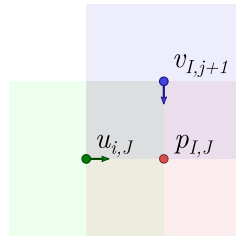
$$\begin{aligned}
 u_e &= u_E \text{ and } u_w = u_P \\
 u_n &= u_P \text{ and } u_s = u_S \\
 v_e &= v_E \text{ and } v_w = v_P \\
 v_n &= v_P \text{ and } v_s = v_S
 \end{aligned}$$

 x -component is:

$$F_{x,e}u_E A_{x,e} - F_{x,w}u_P A_{x,w} + F_{y,n}u_P A_{y,n} - F_{y,s}u_S A_{y,s} = \mathbf{RHS}$$

 y -component is:

$$F_{x,e}v_E A_{x,e} - F_{x,w}v_P A_{x,w} + F_{y,n}v_P A_{y,n} - F_{y,s}v_S A_{y,s} = \mathbf{RHS}$$

Positive x -flow, Negative y -flow

$$u_e = u_P \text{ and } u_w = u_W$$

$$u_n = u_N \text{ and } u_s = u_P$$

$$v_e = v_P \text{ and } v_w = v_W$$

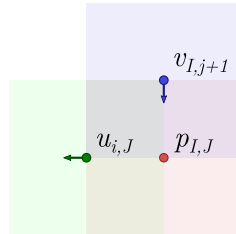
$$v_n = v_N \text{ and } v_s = v_P$$

x -component is:

$$F_{x,e}u_P A_{x,e} - F_{x,w}u_W A_{x,w} + F_{y,n}u_N A_{y,n} - F_{y,s}u_P A_{y,s} = \mathbf{RHS}$$

y -component is:

$$F_{x,e}v_P A_{x,e} - F_{x,w}v_W A_{x,w} + F_{y,n}v_N A_{y,n} - F_{y,s}v_P A_{y,s} = \mathbf{RHS}$$

Negative x -flow, Negative y -flow

$$u_e = u_E \text{ and } u_w = u_P$$

$$u_n = u_N \text{ and } u_s = u_P$$

$$v_e = v_E \text{ and } v_w = v_P$$

$$v_n = v_N \text{ and } v_s = v_P$$

x -component is:

$$F_{x,e}u_E A_{x,e} - F_{x,w}u_P A_{x,w} + F_{y,n}u_N A_{y,n} - F_{y,s}u_P A_{y,s} = \mathbf{RHS}$$

y -component is:

$$F_{x,e}v_E A_{x,e} - F_{x,w}v_P A_{x,w} + F_{y,n}v_N A_{y,n} - F_{y,s}v_P A_{y,s} = \mathbf{RHS}$$

All Flow Directions*x*-component:

$$\begin{aligned}
& \left(\max(0, -F_{x,e}A_{x,e}) + \max(F_{x,w}A_{x,w}, 0) + \max(0, -F_{y,n}A_{y,n}) + \max(F_{y,s}A_{y,s}, 0) \right. \\
& \quad \left. + F_{x,e}A_{x,e} - F_{x,w}A_{x,w} + F_{y,n}A_{y,n} - F_{y,s}A_{y,s} \right) u_P \\
& + \left(-\max(0, -F_{x,e}A_{x,e}) \right) u_E + \left(-\max(F_{x,w}A_{x,w}, 0) \right) u_W \\
& + \left(-\max(0, -F_{y,n}A_{y,n}) \right) u_N + \left(-\max(F_{y,s}A_{y,s}, 0) \right) u_S = \mathbf{RHS}
\end{aligned}$$

y-component:

$$\begin{aligned}
& \left(\max(0, -F_{x,e}A_{x,e}) + \max(F_{x,w}A_{x,w}, 0) + \max(0, -F_{y,n}A_{y,n}) + \max(F_{y,s}A_{y,s}, 0) \right. \\
& \quad \left. + F_{x,e}A_{x,e} - F_{x,w}A_{x,w} + F_{y,n}A_{y,n} - F_{y,s}A_{y,s} \right) v_P \\
& + \left(-\max(0, -F_{x,e}A_{x,e}) \right) v_E + \left(-\max(F_{x,w}A_{x,w}, 0) \right) v_W \\
& + \left(-\max(0, -F_{y,n}A_{y,n}) \right) v_N + \left(-\max(F_{y,s}A_{y,s}, 0) \right) v_S = \mathbf{RHS}
\end{aligned}$$

Right Hand Side

The right hand side of the Momentum equation is rearranged, and the *x*- and *y*-components of the equation are obtained by taking the dot product with the unit vectors \mathbf{e}_x or \mathbf{e}_y before the integration over the control volume *CV*. The gravity term is neglected. The area integral is taken first, and Fundamental Theorem of Algebra is applied to the remaining integral. $\nabla \cdot \mathbf{u}$ is zero from Continuity.

x-component

$$\begin{aligned}
\mathbf{LHS} &= \mathbf{e}_x \cdot \left(-\nabla p - \frac{\partial \boldsymbol{\sigma}_x}{\partial x} - \frac{\partial \boldsymbol{\sigma}_y}{\partial y} \right) \\
\mathbf{LHS} &= -\frac{\partial p}{\partial x} - \frac{\partial \sigma_{xx}}{\partial x} - \frac{\partial \sigma_{xy}}{\partial y} \\
\mathbf{LHS} &= -\frac{\partial p}{\partial x} - \left(-\frac{\partial}{\partial x} \mu \left[2\frac{\partial u}{\partial x} - \frac{2}{3}(\nabla \cdot \mathbf{u}) \right] \right) - \left(-\frac{\partial}{\partial y} \mu \left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right] \right) \\
\mathbf{LHS} &= -\frac{\partial p}{\partial x} - \left(-\frac{\partial}{\partial x} \mu \left[2\frac{\partial u}{\partial x} \right] \right) - \left(-\frac{\partial}{\partial y} \mu \left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right] \right) \\
\mathbf{LHS} &= -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x} \left(2\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial x} \right) \\
\mathbf{LHS} &= -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x} \left(2\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial y} \right)
\end{aligned}$$

$$\begin{aligned}
\text{LHS} &= -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial x} \left(\mu \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \right) \\
\text{LHS} &= -\frac{\partial p}{\partial x} + \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) \\
\text{LHS} &= -\int_{CV} \frac{\partial p}{\partial x} dV + \int_{CV} \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) dV + \int_{CV} \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) dV \\
\text{LHS} &= -\frac{\partial p}{\partial x} \Delta V + \int_{\delta x} \int_{A_x} \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) dA dx + \int_{\delta y} \int_{A_y} \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) dA dy \\
\text{LHS} &= -\frac{\partial p}{\partial x} \Delta V + \int_{\delta x} \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial x} \right) A_x dx + \int_{\delta y} \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial y} \right) A_y dy \\
\text{LHS} &= -\frac{\partial p}{\partial x} \Delta V + \left(\mu \frac{\partial u}{\partial x} A_x \right)_e - \left(\mu \frac{\partial u}{\partial x} A_x \right)_w + \left(\mu \frac{\partial u}{\partial y} A_y \right)_n - \left(\mu \frac{\partial u}{\partial y} A_y \right)_s \\
\text{LHS} &= -\frac{\partial p}{\partial x} \Big|_{i,J} \Delta V + \mu \frac{\partial u}{\partial x} \Big|_e A_{x,e} - \mu \frac{\partial u}{\partial x} \Big|_w A_{x,w} + \mu \frac{\partial u}{\partial y} \Big|_n A_{y,n} - \mu \frac{\partial u}{\partial y} \Big|_s A_{y,s} \\
\text{LHS} &= -\frac{\partial p}{\partial x} \Big|_{i,J} \delta x A_x + \mu \frac{\partial u}{\partial x} \Big|_e A_{x,e} - \mu \frac{\partial u}{\partial x} \Big|_w A_{x,w} + \mu \frac{\partial u}{\partial y} \Big|_n A_{y,n} - \mu \frac{\partial u}{\partial y} \Big|_s A_{y,s}
\end{aligned}$$

The derivative terms above are approximated with the following central differences:

$$\frac{\partial p}{\partial x} \Big|_{i,J} = \frac{p_{I,J} - p_{I-1,J}}{\delta x}$$

$$\frac{\partial u}{\partial x} \Big|_e = \frac{u_{i+1,J} - u_{i,J}}{\delta x}$$

$$\frac{\partial u}{\partial x} \Big|_w = \frac{u_{i,J} - u_{i-1,J}}{\delta x}$$

$$\frac{\partial u}{\partial y} \Big|_n = \frac{u_{i,J+1} - u_{i,J}}{\delta y}$$

$$\frac{\partial u}{\partial y} \Big|_s = \frac{u_{i,J} - u_{i,J-1}}{\delta y}$$

The diffusion conductances $D_x = \frac{\mu}{\delta x}$ and $D_y = \frac{\mu}{\delta y}$ are introduced. For a rectangular control volume, $A_x = A_{x,w} = A_x$ and $A_{y,n} = A_{y,s} = A_y$. Inserting this and the finite

differences yields:

$$\begin{aligned} \text{LHS} &= -\frac{p_{I,J} - p_{I-1,J}}{\delta x} \delta x A_x + \mu \frac{u_{i+1,J} - u_{i,J}}{\delta x} A_x - \mu \frac{u_{i,J} - u_{i-1,J}}{\delta x} A_x \\ &\quad + \mu \frac{u_{i,J+1} - u_{i,J}}{\delta y} A_y - \mu \frac{u_{i,J} - u_{i,J-1}}{\delta y} A_y \end{aligned}$$

$$\begin{aligned} \text{LHS} &= -(p_{I,J} - p_{I-1,J}) A_x + \frac{\mu A_x}{\delta x} (u_{i+1,J} - u_{i,J}) - \frac{\mu A_x}{\delta x} (u_{i,J} - u_{i-1,J}) \\ &\quad + \frac{\mu A_y}{\delta y} (u_{i,J+1} - u_{i,J}) - \frac{\mu A_y}{\delta y} (u_{i,J} - u_{i,J-1}) \end{aligned}$$

$$\begin{aligned} \text{LHS} &= -(p_{I,J} - p_{I-1,J}) A_x + D_x A_x (u_{i+1,J} - u_{i,J}) - D_x A_x (u_{i,J} - u_{i-1,J}) \\ &\quad + D_y A_y (u_{i,J+1} - u_{i,J}) - D_y A_y (u_{i,J} - u_{i,J-1}) \end{aligned}$$

$$\begin{aligned} \text{LHS} &= D_x A_x u_{i+1,J} - D_x A_x u_{i,J} - D_x A_x u_{i,J} + D_x A_x u_{i-1,J} \\ &\quad + D_y A_y u_{i,J+1} - D_y A_y u_{i,J} - D_y A_y u_{i,J} + D_y A_y u_{i,J-1} - (p_{I,J} - p_{I-1,J}) A_x \end{aligned}$$

$$\begin{aligned} \text{LHS} &= \left(-D_x A_x - D_x A_x - D_y A_y - D_y A_y \right) u_{i,J} + D_x A_x u_{i+1,J} + D_x A_x u_{i-1,J} \\ &\quad + D_y A_y u_{i,J+1} + D_y A_y u_{i,J-1} - (p_{I,J} - p_{I-1,J}) A_x \end{aligned}$$

y-component

$$\text{LHS} = \mathbf{e}_y \cdot \left(-\nabla p - \frac{\partial \boldsymbol{\sigma}_x}{\partial x} - \frac{\partial \boldsymbol{\sigma}_y}{\partial y} \right)$$

$$\text{LHS} = -\frac{\partial p}{\partial y} - \frac{\partial \sigma_{yx}}{\partial x} - \frac{\partial \sigma_{yy}}{\partial y}$$

$$\text{LHS} = -\frac{\partial p}{\partial y} - \left(-\frac{\partial}{\partial x} \mu \left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right] \right) - \left(-\frac{\partial}{\partial y} \mu \left[2\frac{\partial v}{\partial y} - \frac{2}{3}(\nabla \cdot \mathbf{u}) \right] \right)$$

$$\text{LHS} = -\frac{\partial p}{\partial y} - \left(-\frac{\partial}{\partial x} \mu \left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right] \right) - \left(-\frac{\partial}{\partial y} \mu \left[2\frac{\partial v}{\partial y} \right] \right)$$

$$\text{LHS} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial x} \left(\mu \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(2\mu \frac{\partial v}{\partial y} \right)$$

$$\text{LHS} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial y} \left(\mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(2\mu \frac{\partial v}{\partial y} \right)$$

$$\text{LHS} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial y} \left(\mu \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right) \right) + \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right)$$

$$\text{LHS} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right)$$

$$\begin{aligned}
\text{LHS} &= - \int_{CV} \frac{\partial p}{\partial y} dV + \int_{CV} \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) dV + \int_{CV} \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) dV \\
\text{LHS} &= - \frac{\partial p}{\partial y} \Delta V + \int_{CV} \frac{\partial}{\partial x} \left(\mu \frac{\partial v}{\partial x} \right) dx A_x + \int_{CV} \frac{\partial}{\partial y} \left(\mu \frac{\partial v}{\partial y} \right) dy A_y \\
\text{LHS} &= - \frac{\partial p}{\partial y} \Delta V + \left(\mu \frac{\partial v}{\partial x} A_x \right)_e - \left(\mu \frac{\partial v}{\partial x} A_x \right)_w + \left(\mu \frac{\partial v}{\partial y} A_y \right)_n - \left(\mu \frac{\partial v}{\partial y} A_y \right)_s \\
\text{LHS} &= - \frac{\partial p}{\partial y} \Big|_{I,j} \Delta V + \mu \frac{\partial v}{\partial x} \Big|_e A_{x,e} - \mu \frac{\partial v}{\partial x} \Big|_w A_{x,w} + \mu \frac{\partial v}{\partial y} \Big|_n A_{y,n} - \mu \frac{\partial v}{\partial y} \Big|_s A_{y,s} \\
\text{LHS} &= - \frac{\partial p}{\partial y} \Big|_{I,j} \delta y A_y + \mu \frac{\partial v}{\partial x} \Big|_e A_{x,e} - \mu \frac{\partial v}{\partial x} \Big|_w A_{x,w} + \mu \frac{\partial v}{\partial y} \Big|_n A_{y,n} - \mu \frac{\partial v}{\partial y} \Big|_s A_{y,s}
\end{aligned}$$

The derivative terms above are approximated with the following central differences:

$$\begin{aligned}
\frac{\partial p}{\partial y} \Big|_{I,j} &= \frac{p_{I,J} - p_{I,J-1}}{\delta y} \\
\frac{\partial v}{\partial x} \Big|_e &= \frac{v_{I+1,j} - v_{I,j}}{\delta x} \\
\frac{\partial v}{\partial x} \Big|_w &= \frac{v_{I,j} - v_{I-1,j}}{\delta x} \\
\frac{\partial v}{\partial y} \Big|_n &= \frac{v_{I,j+1} - v_{I,j}}{\delta y} \\
\frac{\partial v}{\partial y} \Big|_s &= \frac{v_{I,j} - v_{I,j-1}}{\delta y}
\end{aligned}$$

The diffusion conductances $D_x = \frac{\mu}{\delta x}$ and $D_y = \frac{\mu}{\delta y}$ are introduced. For a rectangular control volume, $A_x = A_{x,w} = A_x$ and $A_{y,n} = A_{y,s} = A_y$. Inserting this and the finite differences yields:

$$\begin{aligned}
\text{LHS} &= - \frac{p_{I,J} - p_{I,J-1}}{\delta y} \delta y A_y + \mu \frac{v_{I+1,j} - v_{I,j}}{\delta x} A_{x,e} - \mu \frac{v_{I,j} - v_{I-1,j}}{\delta x} A_{x,w} \\
&\quad + \mu \frac{v_{I,j+1} - v_{I,j}}{\delta y} A_{y,n} - \mu \frac{v_{I,j} - v_{I,j-1}}{\delta y} A_{y,s} \\
\text{LHS} &= - (p_{I,J} - p_{I,J-1}) A_y + \frac{\mu A_{x,e}}{\delta x} (v_{I+1,j} - v_{I,j}) - \frac{\mu A_{x,w}}{\delta x} (v_{I,j} - v_{I-1,j}) \\
&\quad + \frac{\mu A_{y,n}}{\delta y} (v_{I,j+1} - v_{I,j}) - \frac{\mu A_{y,s}}{\delta y} (v_{I,j} - v_{I,j-1})
\end{aligned}$$

$$\begin{aligned} \text{LHS} &= -(p_{I,J} - p_{I,J-1})A_y + D_x A_{x,e}(v_{I+1,j} - v_{I,j}) - D_x A_{x,w}(v_{I,j} - v_{I-1,j}) \\ &\quad + D_y A_{y,n}(v_{I,j+1} - v_{I,j}) - D_y A_{y,s}(v_{I,j} - v_{I,j-1}) \end{aligned}$$

$$\begin{aligned} \text{LHS} &= D_x A_{x,e}v_{I+1,j} - D_x A_{x,e}v_{I,j} - D_x A_{x,w}v_{I,j} + D_x A_{x,w}v_{I-1,j} \\ &\quad + D_y A_{y,n}v_{I,j+1} - D_y A_{y,n}v_{I,j} - D_y A_{y,s}v_{I,j} + D_y A_{y,s}v_{I,j-1} - (p_{I,J} - p_{I,J-1})A_y \end{aligned}$$

$$\begin{aligned} \text{LHS} &= (-D_x A_{x,e} - D_x A_{x,w} - D_y A_{y,n} - D_y A_{y,s})v_{I,j} D_x A_{x,e}v_{I+1,j} + D_x A_{x,w}v_{I-1,j} \\ &\quad + D_y A_{y,n}v_{I,j+1} + D_y A_{y,s}v_{I,j-1} - (p_{I,J} - p_{I,J-1})A_y \end{aligned}$$

Both Sides Combined

x-component

$$\begin{aligned} &\left(\max(0, -F_{x,e}A_x) + \max(F_{x,w}A_y, 0) + \max(0, -F_{y,n}A_y) + \max(F_{y,s}A_y, 0) \right. \\ &\quad \left. + F_{x,e}A_x - F_{x,w}A_y + F_{y,n}A_y - F_{y,s}A_y + D_x A_x + D_x A_y \right. \\ &\quad \left. + D_y A_y + D_y A_y \right) u_{i,J} + \left(-\max(0, -F_{x,e}A_x) - D_x A_x \right) u_{i+1,J} \\ &+ \left(-\max(F_{x,w}A_y, 0) - D_x A_y \right) u_{i-1,J} + \left(-\max(0, -F_{y,n}A_y) - D_y A_y \right) u_{i,J+1} \\ &\quad + \left(-\max(F_{y,s}A_y, 0) - D_y A_y \right) u_{i,J-1} = -(p_{I,J} - p_{I-1,J})A_x \end{aligned}$$

On coefficient form:

$$a_{i,J}u_{i,J} + a_{i+1,J}u_{i+1,J} + a_{i-1,J}u_{i-1,J} + a_{i,J+1}u_{i,J+1} + a_{i,J-1}u_{i,J-1} = b_{i,J}$$

with

$$a_{i,J} = -a_{i+1,J} - a_{i-1,J} - a_{i,J+1} - a_{i,J-1} + F_{x,e}A_x - F_{x,w}A_y + F_{y,n}A_y - F_{y,s}A_y$$

$$a_{i+1,J} = -\max(0, -F_{x,e}A_x) - D_x A_x$$

$$a_{i-1,J} = -\max(F_{x,w}A_y, 0) - D_x A_y$$

$$a_{i,J+1} = -\max(0, -F_{y,n}A_y) - D_y A_y$$

$$a_{i,J-1} = -\max(F_{y,s}A_y, 0) - D_y A_y$$

$$b_{i,J} = -(p_{I,J} - p_{I-1,J})A_x$$

y-component

$$\begin{aligned}
& \left(\max(0, -F_{x,e}A_x) + \max(F_{x,w}A_y, 0) + \max(0, -F_{y,n}A_y) + \max(F_{y,s}A_y, 0) \right. \\
& \quad + F_{x,e}A_x - F_{x,w}A_y + F_{y,n}A_y - F_{y,s}A_y + D_xA_x + D_xA_y \\
& \quad \left. + D_yA_y + D_yA_y \right) v_{I,j} + \left(-\max(0, -F_{x,e}A_x) - D_xA_x \right) v_{I+1,j} \\
& + \left(-\max(F_{x,w}A_y, 0) - D_xA_y \right) v_{I-1,j} + \left(-\max(0, -F_{y,n}A_y) - D_yA_y \right) v_{I,j+1} \\
& \quad + \left(-\max(F_{y,s}A_y, 0) - D_yA_y \right) v_{I,j-1} = -\left(p_{I,J} - p_{I,J-1} \right) A_x
\end{aligned}$$

On coefficient form:

$$a_{I,j}v_{I,j} + a_{I+1,j}v_{I+1,j} + a_{I-1,j}v_{I-1,j} + a_{I,j+1}v_{I,j+1} + a_{I,j-1}v_{I,j-1} = b_{I,j}$$

with

$$a_{I,j} = -a_{I+1,j} - a_{I-1,j} - a_{I,j+1} - a_{I,j-1} + F_{x,e}A_x - F_{x,w}A_y + F_{y,n}A_y - F_{y,s}A_y$$

$$a_{I+1,j} = -\max(0, -F_{x,e}A_x) - D_xA_x$$

$$a_{I-1,j} = -\max(F_{x,w}A_y, 0) - D_xA_y$$

$$a_{I,j+1} = -\max(0, -F_{y,n}A_y) - D_yA_y$$

$$a_{I,j-1} = -\max(F_{y,s}A_y, 0) - D_yA_y$$

$$b_{I,j} = -\left(p_{I,J} - p_{I,J-1} \right) A_y$$

C.3 SIMPLE-Equations**Velocity Correction Equation****x-component**

The Momentum equation for the correct properties:

$$\begin{aligned}
a_{i,J}u_{i,J} + a_{i+1,J}u_{i+1,J} + a_{i-1,J}u_{i-1,J} + a_{i,J+1}u_{i,J+1} + a_{i,J-1}u_{i,J-1} \\
= -\left(p_{I,J} - p_{I-1,J} \right) A_{x,i,J} + b_{i,J}
\end{aligned}$$

The Momentum equation for the intermediate / guessed properties:

$$\begin{aligned}
a_{i,J}u_{i,J}^* + a_{i+1,J}u_{i+1,J}^* + a_{i-1,J}u_{i-1,J}^* + a_{i,J+1}u_{i,J+1}^* + a_{i,J-1}u_{i,J-1}^* \\
= -\left(p_{I,J}^* - p_{I-1,J}^* \right) A_{x,i,J} + b_{i,J}
\end{aligned}$$

The guessed velocity Momentum equation is subtracted from the correct velocity Momentum equation. The correction terms for all the neighbouring nodes are neglected,

keeping only the correction in the center node:

$$\begin{aligned} & a_{i,J}(u_{i,J} - u_{i,J}^*) + a_{i+1,J}(u_{i+1,J} - u_{i+1,J}^*) + a_{i-1,J}(u_{i-1,J} - u_{i-1,J}^*) \\ & \quad + a_{i,J+1}(u_{i,J+1} - u_{i,J+1}^*) + a_{i,J-1}(u_{i,J-1} - u_{i,J-1}^*) \\ & = \left(-p_{I,J} + p_{I-1,J} + p_{I,J}^* - p_{I-1,J}^* \right) A_{x,i,J} + \cancel{b_{i,J}} - \cancel{b_{i,J}} \end{aligned}$$

$$\begin{aligned} & a_{i,J}^{centre}(u_{i,J} - u_{i,J}^*) + \cancel{a_{i+1,J}u_{i+1,J}'} + \cancel{a_{i-1,J}u_{i-1,J}'} + \cancel{a_{i,J+1}u_{i,J+1}'} + \cancel{a_{i,J-1}u_{i,J-1}'} \\ & = - \left(p'_{I,J} - p'_{I-1,J} \right) A_{x,i,J} \end{aligned}$$

The velocity correction equation is then:

$$u_{i,J} = u_{i,J}^* - \frac{A_{x,i,J}}{a_{i,J}^{centre}} \left(p'_{I,J} - p'_{I-1,J} \right)$$

y-component

The Momentum equation for the correct properties:

$$\begin{aligned} & a_{I,j}v_{I,j} + a_{I+1,j}v_{I+1,j} + a_{I-1,j}v_{I-1,j} + a_{I,j+1}v_{I,j+1} + a_{I,j-1}v_{I,j-1} \\ & = - \left(p_{I,J} - p_{I,J-1} \right) A_{y,I,j} + b_{i,J} \end{aligned}$$

The Momentum equation for the intermediate / guessed properties:

$$\begin{aligned} & a_{I,j}v_{I,j}^* + a_{I+1,j}v_{I+1,j}^* + a_{I-1,j}v_{I-1,j}^* + a_{I,j+1}v_{I,j+1}^* + a_{I,j-1}v_{I,j-1}^* \\ & = - \left(p_{I,J}^* - p_{I,J-1}^* \right) A_{y,I,j} + b_{i,J} \end{aligned}$$

The guessed velocity Momentum equation is subtracted from the correct velocity Momentum equation. The correction terms for all the neighbouring nodes are neglected, keeping only the correction in the center node:

$$\begin{aligned} & a_{I,j}(v_{I,j} - v_{I,j}^*) + a_{I+1,j}(v_{I+1,j} - v_{I+1,j}^*) + a_{I-1,j}(v_{I-1,j} - v_{I-1,j}^*) \\ & \quad + a_{I,j+1}(v_{I,j+1} - v_{I,j+1}^*) + a_{I,j-1}(v_{I,j-1} - v_{I,j-1}^*) \\ & = \left(-p_{I,J} + p_{I,J-1} + p_{I,J}^* - p_{I,J-1}^* \right) A_{y,I,j} + \cancel{b_{I,j}} - \cancel{b_{I,j}} \end{aligned}$$

$$a_{I,j}^{centre}(v_{I,j} - v_{I,j}^*) + \cancel{a_{I+1,j}v_{I+1,j}'} + \cancel{a_{I-1,j}v_{I-1,j}'} + \cancel{a_{I,j+1}v_{I,j+1}'} + \cancel{a_{I,j-1}v_{I,j-1}'} = - \left(p'_{I,J} - p'_{I,J-1} \right) A_{y,I,j}$$

The velocity correction equation is then:

$$v_{I,j} = v_{I,j}^* - \frac{A_{y,I,j}}{a_{I,j}^{centre}} \left(p'_{I,J} - p'_{I,J-1} \right)$$

Pressure Correction Equation

The velocity correction equations and the Continuity equation are used to produce the pressure correction equation. The Continuity equation is:

$$\rho u_{i+1,J} A_{x,i+1,J} - \rho u_{i,J} A_{x,i,J} + \rho v_{I,j+1} A_{y,I,j+1} - \rho v_{I,j} A_{y,I,j} = 0$$

The velocity correction equations are inserted for $u_{i+1,J}$, $u_{i,J}$, $v_{I,j+1}$ and $v_{I,j}$, and the equation is rearranged:

$$\begin{aligned} & \rho A_{x,i+1,J} \left(u_{i+1,J}^* - \frac{A_{x,i+1,J}}{a_{i+1,J}^{\text{centre}}} (p'_{I+1,J} - p'_{I,J}) \right) - \rho A_{x,i,J} \left(u_{i,J}^* - \frac{A_{x,i,J}}{a_{i,J}^{\text{centre}}} (p'_{I,J} - p'_{I-1,J}) \right) \\ & + \rho A_{y,I,j+1} \left(v_{I,j+1}^* - \frac{A_{y,I,j+1}}{a_{I,j+1}^{\text{centre}}} (p'_{I,J+1} - p'_{I,J}) \right) - \rho A_{y,I,j} \left(v_{I,j}^* - \frac{A_{y,I,j}}{a_{I,j}^{\text{centre}}} (p'_{I,J} - p'_{I,J-1}) \right) = 0 \end{aligned}$$

$$\begin{aligned} & \rho A_{x,i+1,J} u_{i+1,J}^* - \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} (p'_{I+1,J} - p'_{I,J}) - \rho A_{x,i,J} u_{i,J}^* + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} (p'_{I,J} - p'_{I-1,J}) \\ & + \rho A_{y,I,j+1} v_{I,j+1}^* - \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} (p'_{I,J+1} - p'_{I,J}) - \rho A_{y,I,j} v_{I,j}^* + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} (p'_{I,J} - p'_{I,J-1}) = 0 \end{aligned}$$

$$\begin{aligned} & - \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} (p'_{I+1,J} - p'_{I,J}) + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} (p'_{I,J} - p'_{I-1,J}) \\ & \quad - \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} (p'_{I,J+1} - p'_{I,J}) + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} (p'_{I,J} - p'_{I,J-1}) \\ & = -\rho A_{x,i+1,J} u_{i+1,J}^* + \rho A_{x,i,J} u_{i,J}^* - \rho A_{y,I,j+1} v_{I,j+1}^* + \rho A_{y,I,j} v_{I,j}^* \end{aligned}$$

$$\begin{aligned} & - \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} p'_{I+1,J} + \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} p'_{I,J} + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} p'_{I,J} - \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} p'_{I-1,J} \\ & \quad - \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} p'_{I,J+1} + \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} p'_{I,J} + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} p'_{I,J} - \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} p'_{I,J-1} \\ & = -\rho A_{x,i+1,J} u_{i+1,J}^* + \rho A_{x,i,J} u_{i,J}^* - \rho A_{y,I,j+1} v_{I,j+1}^* + \rho A_{y,I,j} v_{I,j}^* \end{aligned}$$

$$\begin{aligned} & \left(\frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} + \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} \right) p'_{I,J} \\ & \quad - \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} p'_{I+1,J} - \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} p'_{I-1,J} - \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} p'_{I,J+1} - \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} p'_{I,J-1} \\ & = -\rho A_{x,i+1,J} u_{i+1,J}^* + \rho A_{x,i,J} u_{i,J}^* - \rho A_{y,I,j+1} v_{I,j+1}^* + \rho A_{y,I,j} v_{I,j}^* \end{aligned}$$

$$\begin{aligned} & \left(\frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} + \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} \right) p'_{I,J} \\ & \quad - \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{\text{centre}}} p'_{I+1,J} - \frac{\rho A_{x,i,J}^2}{a_{i,J}^{\text{centre}}} p'_{I-1,J} - \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{\text{centre}}} p'_{I,J+1} - \frac{\rho A_{y,I,j}^2}{a_{I,j}^{\text{centre}}} p'_{I,J-1} \\ & = -A_{x,i+1,J} F_{i+1,J}^* + A_{x,i,J} F_{i,J}^* - A_{y,I,j+1} F_{I,j+1}^* + A_{y,I,j} F_{I,j}^* \end{aligned}$$

The pressure correction equation is:

$$\nu_{I,J}p'_{I,J} + \nu_{I+1,J}p'_{I+1,J} + \nu_{I-1,J}p'_{I-1,J} + \nu_{I,J+1}p'_{I,J+1} + \nu_{I,J-1}p'_{I,J-1} = \beta_{I,J}$$

with

$$\begin{aligned} \nu_{I,J} &= \frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{centre}} + \frac{\rho A_{x,i,J}^2}{a_{i,J}^{centre}} + \frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{centre}} + \frac{\rho A_{y,I,j}^2}{a_{I,j}^{centre}} \\ \nu_{I+1,J} &= -\frac{\rho A_{x,i+1,J}^2}{a_{i+1,J}^{centre}} \\ \nu_{I-1,J} &= -\frac{\rho A_{x,i,J}^2}{a_{i,J}^{centre}} \\ \nu_{I,J+1} &= -\frac{\rho A_{y,I,j+1}^2}{a_{I,j+1}^{centre}} \\ \nu_{I,J-1} &= -\frac{\rho A_{y,I,j}^2}{a_{I,j}^{centre}} \\ \beta_{I,J} &= -A_{x,i+1,J}F_{i+1,J}^* + A_{x,i,J}F_{i,J}^* - A_{y,I,j+1}F_{I,j+1}^* + A_{y,I,j}F_{I,j}^* \end{aligned}$$

D

Elliptic Grid Generation in Three Dimensions

D.1 Elliptic Grid Generation Equation

The equation to be discretised is equation (7.1.9). For a three dimensional system, the summations as shown in equations (D.1.2) (D.1.3) and (D.1.4) are taken, all sums from 1 to 3. The position vector r is expressed as in equation (D.1.1).

$$\mathbf{r} = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z \quad (\text{D.1.1})$$

$$\sum_{i=1}^3 \sum_{j=1}^3 \sum_{k=1}^3 \left(g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial x}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial x}{\partial q^j} \right) = 0 \quad (\text{D.1.2})$$

$$\sum_{i=1}^3 \sum_{j=1}^3 \sum_{k=1}^3 \left(g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial y}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial y}{\partial q^j} \right) = 0 \quad (\text{D.1.3})$$

$$\sum_{i=1}^3 \sum_{j=1}^3 \sum_{k=1}^3 \left(g^{ij} \frac{\partial}{\partial q^i} \left(\frac{\partial z}{\partial q^j} \right) + \nabla^2 q^j \frac{\partial z}{\partial q^j} \right) = 0 \quad (\text{D.1.4})$$

Taking the sums yields equations (D.1.5) (D.1.6) and (D.1.7) for the x -, y - and z components respectively.

$$\begin{aligned} & g^{11} \frac{\partial}{\partial q^1} \left(\frac{\partial x}{\partial q^1} \right) + g^{12} \frac{\partial}{\partial q^1} \left(\frac{\partial x}{\partial q^2} \right) + g^{13} \frac{\partial}{\partial q^1} \left(\frac{\partial x}{\partial q^3} \right) \\ & + g^{21} \frac{\partial}{\partial q^2} \left(\frac{\partial x}{\partial q^1} \right) + g^{22} \frac{\partial}{\partial q^2} \left(\frac{\partial x}{\partial q^2} \right) + g^{23} \frac{\partial}{\partial q^2} \left(\frac{\partial x}{\partial q^3} \right) \\ & + g^{31} \frac{\partial}{\partial q^3} \left(\frac{\partial x}{\partial q^1} \right) + g^{32} \frac{\partial}{\partial q^3} \left(\frac{\partial x}{\partial q^2} \right) + g^{33} \frac{\partial}{\partial q^3} \left(\frac{\partial x}{\partial q^3} \right) \\ & + \nabla^2 q^1 \frac{\partial x}{\partial q^1} + \nabla^2 q^2 \frac{\partial x}{\partial q^2} + \nabla^2 q^3 \frac{\partial x}{\partial q^3} = 0 \quad (\text{D.1.5}) \end{aligned}$$

$$\begin{aligned}
& g^{11} \frac{\partial}{\partial q^1} \left(\frac{\partial y}{\partial q^1} \right) + g^{12} \frac{\partial}{\partial q^1} \left(\frac{\partial y}{\partial q^2} \right) + g^{13} \frac{\partial}{\partial q^1} \left(\frac{\partial y}{\partial q^3} \right) \\
& + g^{21} \frac{\partial}{\partial q^2} \left(\frac{\partial y}{\partial q^1} \right) + g^{22} \frac{\partial}{\partial q^2} \left(\frac{\partial y}{\partial q^2} \right) + g^{23} \frac{\partial}{\partial q^2} \left(\frac{\partial y}{\partial q^3} \right) \\
& + g^{31} \frac{\partial}{\partial q^3} \left(\frac{\partial y}{\partial q^1} \right) + g^{32} \frac{\partial}{\partial q^3} \left(\frac{\partial y}{\partial q^2} \right) + g^{33} \frac{\partial}{\partial q^3} \left(\frac{\partial y}{\partial q^3} \right) \\
& + \nabla^2 q^1 \frac{\partial y}{\partial q^1} + \nabla^2 q^2 \frac{\partial y}{\partial q^2} + \nabla^2 q^3 \frac{\partial y}{\partial q^3} = 0 \quad (\text{D.1.6})
\end{aligned}$$

$$\begin{aligned}
& g^{11} \frac{\partial}{\partial q^1} \left(\frac{\partial z}{\partial q^1} \right) + g^{12} \frac{\partial}{\partial q^1} \left(\frac{\partial z}{\partial q^2} \right) + g^{13} \frac{\partial}{\partial q^1} \left(\frac{\partial z}{\partial q^3} \right) \\
& + g^{21} \frac{\partial}{\partial q^2} \left(\frac{\partial z}{\partial q^1} \right) + g^{22} \frac{\partial}{\partial q^2} \left(\frac{\partial z}{\partial q^2} \right) + g^{23} \frac{\partial}{\partial q^2} \left(\frac{\partial z}{\partial q^3} \right) \\
& + g^{31} \frac{\partial}{\partial q^3} \left(\frac{\partial z}{\partial q^1} \right) + g^{32} \frac{\partial}{\partial q^3} \left(\frac{\partial z}{\partial q^2} \right) + g^{33} \frac{\partial}{\partial q^3} \left(\frac{\partial z}{\partial q^3} \right) \\
& + \nabla^2 q^1 \frac{\partial z}{\partial q^1} + \nabla^2 q^2 \frac{\partial z}{\partial q^2} + \nabla^2 q^3 \frac{\partial z}{\partial q^3} = 0 \quad (\text{D.1.7})
\end{aligned}$$

D.2 Expression for the Contravariant Tensor Components

Area components:

$$A_1^1 = \frac{\partial x^2}{\partial q^2} \frac{\partial x^3}{\partial q^3} - \frac{\partial x^3}{\partial q^2} \frac{\partial x^2}{\partial q^3} \quad (\text{D.2.1})$$

$$A_1^2 = \frac{\partial x^2}{\partial q^3} \frac{\partial x^3}{\partial q^1} - \frac{\partial x^3}{\partial q^3} \frac{\partial x^2}{\partial q^1} \quad (\text{D.2.2})$$

$$A_1^3 = \frac{\partial x^2}{\partial q^1} \frac{\partial x^3}{\partial q^2} - \frac{\partial x^3}{\partial q^1} \frac{\partial x^2}{\partial q^2} \quad (\text{D.2.3})$$

$$A_2^1 = \frac{\partial x^3}{\partial q^2} \frac{\partial x^1}{\partial q^3} - \frac{\partial x^1}{\partial q^2} \frac{\partial x^3}{\partial q^3} \quad (\text{D.2.4})$$

$$A_2^2 = \frac{\partial x^3}{\partial q^3} \frac{\partial x^1}{\partial q^1} - \frac{\partial x^1}{\partial q^3} \frac{\partial x^3}{\partial q^1} \quad (\text{D.2.5})$$

$$A_2^3 = \frac{\partial x^3}{\partial q^1} \frac{\partial x^1}{\partial q^2} - \frac{\partial x^1}{\partial q^1} \frac{\partial x^3}{\partial q^2} \quad (\text{D.2.6})$$

$$A_3^1 = \frac{\partial x^1}{\partial q^2} \frac{\partial x^2}{\partial q^3} - \frac{\partial x^2}{\partial q^2} \frac{\partial x^1}{\partial q^3} \quad (\text{D.2.7})$$

$$A_3^2 = \frac{\partial x^1}{\partial q^3} \frac{\partial x^2}{\partial q^1} - \frac{\partial x^2}{\partial q^3} \frac{\partial x^1}{\partial q^1} \quad (\text{D.2.8})$$

$$A_3^3 = \frac{\partial x^1}{\partial q^1} \frac{\partial x^2}{\partial q^2} - \frac{\partial x^2}{\partial q^1} \frac{\partial x^1}{\partial q^2} \quad (\text{D.2.9})$$

Jacobi determinant:

$$J = \det \left(J_j^i \right) \quad (\text{D.2.10})$$

$$= \begin{vmatrix} \frac{\partial x^1}{\partial q^1} & \frac{\partial x^1}{\partial q^2} & \frac{\partial x^1}{\partial q^3} \\ \frac{\partial x^2}{\partial q^1} & \frac{\partial x^2}{\partial q^2} & \frac{\partial x^2}{\partial q^3} \\ \frac{\partial x^3}{\partial q^1} & \frac{\partial x^3}{\partial q^2} & \frac{\partial x^3}{\partial q^3} \end{vmatrix} \quad (\text{D.2.11})$$

$$= \frac{\partial x^1}{\partial q^1} \begin{vmatrix} \frac{\partial x^2}{\partial q^2} & \frac{\partial x^2}{\partial q^3} \\ \frac{\partial x^3}{\partial q^2} & \frac{\partial x^3}{\partial q^3} \end{vmatrix} - \frac{\partial x^1}{\partial q^2} \begin{vmatrix} \frac{\partial x^2}{\partial q^1} & \frac{\partial x^2}{\partial q^3} \\ \frac{\partial x^3}{\partial q^1} & \frac{\partial x^3}{\partial q^3} \end{vmatrix} + \frac{\partial x^1}{\partial q^3} \begin{vmatrix} \frac{\partial x^2}{\partial q^1} & \frac{\partial x^2}{\partial q^2} \\ \frac{\partial x^3}{\partial q^1} & \frac{\partial x^3}{\partial q^2} \end{vmatrix} \quad (\text{D.2.12})$$

$$= \frac{\partial x^1}{\partial q^1} \left(\frac{\partial x^2}{\partial q^2} \frac{\partial x^3}{\partial q^3} - \frac{\partial x^3}{\partial q^2} \frac{\partial x^2}{\partial q^3} \right) - \frac{\partial x^1}{\partial q^2} \left(\frac{\partial x^2}{\partial q^1} \frac{\partial x^3}{\partial q^3} - \frac{\partial x^3}{\partial q^1} \frac{\partial x^2}{\partial q^3} \right) + \frac{\partial x^1}{\partial q^3} \left(\frac{\partial x^2}{\partial q^1} \frac{\partial x^3}{\partial q^2} - \frac{\partial x^3}{\partial q^1} \frac{\partial x^2}{\partial q^2} \right) \quad (\text{D.2.13})$$

$$= \frac{\partial x^1}{\partial q^1} \frac{\partial x^2}{\partial q^2} \frac{\partial x^3}{\partial q^3} - \frac{\partial x^1}{\partial q^1} \frac{\partial x^3}{\partial q^2} \frac{\partial x^2}{\partial q^3} - \frac{\partial x^1}{\partial q^2} \frac{\partial x^2}{\partial q^1} \frac{\partial x^3}{\partial q^3} + \frac{\partial x^1}{\partial q^2} \frac{\partial x^3}{\partial q^1} \frac{\partial x^2}{\partial q^3} + \frac{\partial x^1}{\partial q^3} \frac{\partial x^2}{\partial q^1} \frac{\partial x^3}{\partial q^2} - \frac{\partial x^1}{\partial q^3} \frac{\partial x^3}{\partial q^1} \frac{\partial x^2}{\partial q^2} \quad (\text{D.2.14})$$

Contravariant tensor components summed over k :

$$g^{ij} = \frac{\mathbf{A}^i \cdot \mathbf{A}^j}{J^2} = \frac{A_k^i \mathbf{e}_k \cdot A_l^j \mathbf{e}_L}{j^2} = \frac{A_k^i A_l^j \delta_{kl}}{J^2} = \frac{A_k^i A_k^j}{J^2} \quad (\text{D.2.15})$$

Components of g^{ij} :

$$g^{11} = \frac{A_k^1 A_k^1}{J^2} = \frac{A_1^1 A_1^1 + A_2^1 A_2^1 + A_3^1 A_3^1}{J^2} \quad (\text{D.2.16})$$

$$g^{21} = \frac{A_k^2 A_k^1}{J^2} = \frac{A_1^2 A_1^1 + A_2^2 A_2^1 + A_3^2 A_3^1}{J^2} \quad (\text{D.2.17})$$

$$g^{31} = \frac{A_k^3 A_k^1}{J^2} = \frac{A_1^3 A_1^1 + A_2^3 A_2^1 + A_3^3 A_3^1}{J^2} \quad (\text{D.2.18})$$

$$g^{12} = \frac{A_k^1 A_k^2}{J^2} = \frac{A_1^1 A_1^2 + A_2^1 A_2^2 + A_3^1 A_3^2}{J^2} \quad (\text{D.2.19})$$

$$g^{22} = \frac{A_k^2 A_k^2}{J^2} = \frac{A_1^2 A_1^2 + A_2^2 A_2^2 + A_3^2 A_3^2}{J^2} \quad (\text{D.2.20})$$

$$g^{32} = \frac{A_k^3 A_k^2}{J^2} = \frac{A_1^3 A_1^2 + A_2^3 A_2^2 + A_3^3 A_3^2}{J^2} \quad (\text{D.2.21})$$

$$g^{13} = \frac{A_k^1 A_k^3}{J^2} = \frac{A_1^1 A_1^3 + A_2^1 A_2^3 + A_3^1 A_3^3}{J^2} \quad (\text{D.2.22})$$

$$g^{23} = \frac{A_k^2 A_k^3}{J^2} = \frac{A_1^2 A_1^3 + A_2^2 A_2^3 + A_3^2 A_3^3}{J^2} \quad (\text{D.2.23})$$

$$g^{33} = \frac{A_k^3 A_k^3}{J^2} = \frac{A_1^3 A_1^3 + A_2^3 A_2^3 + A_3^3 A_3^3}{J^2} \quad (\text{D.2.24})$$

E

MATLAB Code

A list of the names of the parameters as used in **MATLAB** and the codes used to solve the models are given in this chapter. A map of how the scripts and functions are used is given in section 4.9. The appendix Table of Contents is helpful to find a specific code. The use of each code is explained. All the codes can also be found in the attached `.zip` file.

E.1 Codes Sorted by Model

Below follows a grouped list of all the codes used in this thesis. The models are separated into four general groups with the following codes:

1. 1D: The one-dimensional model for the straight channel
 - `channel_1D.m`
2. 2D: The two-dimensional model for the straight channel, dimensionless.
 - `channel_2D.m`
 - `plot_2D.m`
3. BFS: The two-dimensional model for the backwards facing step model, dimensionless with constant and parabolic inlet:
 - `channel_BFS.m`
 - `channel_BFS_parabolc.m`
 - `BFS_u_velocity.m`
 - `BFS_u_velocity_parabolc.m`
 - `BFS_pressurecorrection.m`
 - `BFS_pressurecorrection_parabolc.m`

- BFS_v_velocity.m
- BFS_v_velocity_parabolc.m
- isWide.m
- getRowNumber.m
- getRowOver.m
- getRowUnder.m
- global2matrix.m
- plot_BFS.m
- plot_BFS_parabolc.m
- plotColoredQuiver.m
- plotColoredQuiver_parabolic.m
- plotVelocityCorrection.m
- plotIntermediates.m
- plot_BFS_iterations.m
- plotVelInts_BFS_iterations.m

4. GG: Grid generation codes

- elliptic.m
- getCol.m
- getRow.m
- global2matrix.m
- matrix2global.m
- transfinite.m

E.2 List of MATLAB parameters

A list of MATLAB parameters is given in this section, containing the names used in MATLAB for the fluid flow parameters.

The list includes the names for the parameters used in MATLAB for each group of models, as well as the unit, description, corresponding symbol in derivations if it exists, and which models the parameter appears in.

Parameters solely used for plotting are excluded from the list, as well as some parameters for intermediate calculations.

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
A	Number	m ²	Surface area of control volume in x -direction.	A	✓			
A11	Vector	m ²	Face area component	A_1^1				✓
A12	Vector	m ²	Face area component	A_2^1				✓
A21	Vector	m ²	Face area component	A_1^2				✓
A22	Vector	m ²	Face area component	A_2^2				✓
AM11	Matrix	m ²	Face area component	A_1^1				✓
AM12	Matrix	m ²	Face area component	A_2^1				✓
AM21	Matrix	m ²	Face area component	A_1^2				✓
AM22	Matrix	m ²	Face area component	A_2^2				✓
A_x	Number	-	Dimensionless surface area of control volume in x -direction.	\hat{A}_x		✓	✓	
A_x_true	Number	m ²	Surface area of control volume in x -direction.	A_x		✓	✓	
A_y	Number	-	Dimensionless cross-sectional area in x -direction.	\hat{A}_y		✓	✓	
A_y_true	Number	m ²	Cross-sectional area in y -direction.	A_x		✓	✓	
alpha	-	-	Under-relaxation factor	α				✓
alpha_p	Number	-	Under-relaxation factor for pressure	α_p	✓	✓	✓	
alpha_u	Number	-	Under-relaxation factor for u -velocity	α_u	✓	✓	✓	
alpha_v	Number	-	Under-relaxation factor for v -velocity	α_v		✓	✓	
au	Vector	kg/s	Centre node coefficient for u -velocity	a_u^{centre}	✓			
au	Vector	-	Centre node coefficient for u -velocity	a_u^{centre}		✓	✓	
av	Vector	-	Centre node coefficient for v -velocity	a_v^{centre}		✓	✓	
beta	Vector	Pa	Source term in pressure correction equation	β	✓			
beta	Vector	-	Source term in pressure correction equation	$\hat{\beta}$		✓	✓	
bu	Vector	m/s	Source term in u -velocity equation	$b_{i,J}$	✓			
bu	Vector	-	Source term in u -velocity equation	$\hat{b}_{i,J}$		✓	✓	

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
bv	Vector	-	Source term in v -velocity equation	$\hat{b}_{I,j}$		✓	✓	✓
bx	Vector	-	Source term for x					✓
by	Vector	-	Source term for y					✓
c1	Number	-	Convergence criterion, u -velocity residual	C_1	✓	✓	✓	
c1_diff	Number	-	Distance from value of c1 to limit		✓	✓	✓	
c1_lim	Number	-	c1 limit		✓	✓	✓	
c2	Number	-	Convergence criterion, v -velocity residual	C_2		✓	✓	
c2_diff	Number	-	Distance from value of c2 to limit			✓	✓	
c2_lim	Number	-	c2 limit			✓	✓	
c3	Number	Pa	Convergence criterion, continuity	C_3	✓			
c3	Number	-	Convergence criterion, continuity	C_3		✓	✓	
c3_diff	Number	Pa	Distance from value of c3 to limit		✓			
c3_diff	Number	-	Distance from value of c3 to limit			✓	✓	
c3_lim	Number	-	c3 limit		✓	✓	✓	
c4	Number	-	Convergence criterion, iteration change u -velocity	C_4	✓	✓	✓	
c4_diff	Number	-	Distance from value of c4 to limit		✓	✓	✓	
c4_lim	Number	-	c4 limit		✓	✓	✓	
c5	Number	-	Convergence criterion, iteration change v -velocity	C_5		✓	✓	
c5_diff	Number	-	Distance from value of c5 to limit			✓	✓	
c5_lim	Number	-	c5 limit			✓	✓	
conv	Boolean	-	True if the model is converged		✓	✓	✓	✓
cx	Number	-	Convergence criterion for x	C_x		✓	✓	
cx_lim	Number	-	cx limit			✓	✓	
cy	Number	-	Convergence criterion for y	C_x		✓	✓	

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
cy_lim	Number	-	cy limit			✓	✓	
D_hyd	Number	m	Hydraulic diameter	D_{hyd}		✓	✓	
D	Number	Pa·s/m	Diffusion conductance	D	✓			
D_x	Number	-	Dimensionless diffusion conductance in x -direction	\hat{D}_x		✓	✓	
D_y	Number	-	Dimensionless diffusion conductance in y -direction	\hat{D}_y		✓	✓	
del_x	Number	m	Control volume width	δ_x	✓			
del_x	Number	-	Dimensionless control volume width	$\hat{\delta}_x$		✓	✓	
del_x_true	Number	m	Control volume width	δ_x		✓	✓	
del_y	Number	-	Dimensionless control volume height	$\hat{\delta}_y$		✓	✓	
del_y_true	Number	m	Control volume height	δ_y		✓	✓	
E_coeff	Number	-	Eastern node coefficient in velocity or pressure corr. equation	\hat{a}_E		✓	✓	
eP_coeff	Number	-	Eastern node contribution to centre node			✓	✓	
etest	Boolean	-	True if node point is at eastern boundary			✓	✓	✓
F_e	Vector	-	Dimensionless convective mass flux for u -velocity, east cell face	F_e	✓			
F_xe	Vector	-	Dimensionless convective mass flux for u -velocity, east cell face	$\hat{F}_{x,e}$		✓	✓	
F_xn	Vector	-	Dimensionless convective mass flux for u -velocity, north cell face	$\hat{F}_{x,e}$		✓	✓	
F_xs	Vector	-	Dimensionless convective mass flux for u -velocity, south cell face	$\hat{F}_{x,s}$		✓	✓	
F_xw	Vector	-	Dimensionless convective mass flux for u -velocity, west cell face	$\hat{F}_{x,w}$		✓	✓	
F_ye	Vector	-	Dimensionless convective mass flux for v -velocity, east cell face	$\hat{F}_{y,e}$		✓	✓	
F_yn	Vector	-	Dimensionless convective mass flux for v -velocity, north cell face	$\hat{F}_{y,n}$		✓	✓	
F_ys	Vector	-	Dimensionless convective mass flux for v -velocity, south cell face	$\hat{F}_{y,s}$		✓	✓	
F_yw	Vector	-	Dimensionless convective mass flux for v -velocity, west cell face	$\hat{F}_{y,w}$		✓	✓	
F_w	Vector	kg/sm ²	Dimensionless convective mass flux for u -velocity, west cell face	F_w	✓			
filler	Matrix	-	Filler value placed where the step is in the BFS models				✓	

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
g11	Vector	m ²	Contravariant tensor component	g^{11}				✓
g12	Vector	m ²	Contravariant tensor component	g^{12}				✓
g21	Vector	m ²	Contravariant tensor component	g^{21}				✓
g22	Vector	m ²	Contravariant tensor component	g^{22}				✓
gM11	Matrix	m ²	Contravariant tensor component	g^{11}				✓
gM12	Matrix	m ²	Contravariant tensor component	g^{12}				✓
gM21	Matrix	m ²	Contravariant tensor component	g^{21}				✓
gM22	Matrix	m ²	Contravariant tensor component	g^{22}				✓
h	Number	m	Narrow channel height	h		✓	✓	
h	Number	m	Height of step	h				✓
H	Number	m	Backwards facing step height				✓	
H_total	Number	m	Channel height after step	H			✓	
it	Number	-	Current iteration number		✓	✓	✓	✓
l	Number	m	Narrow channel length	l	✓	✓	✓	✓
L	Number	m	Channel length after the backwards facing step				✓	
L_total	Number	m	Total channel length	L			✓	
M	Number	-	# of scalar nodes in y -dir.			✓		
M	-	-	Length of q^2 -vector					✓
m_narrow	Number	-	# of v -vel. nodes in y -dir. in narrow channel				✓	
M_narrow	Number	-	# of u -vel./pressure corr. nodes in y -dir. in narrow channel				✓	
m_total	Number	-	# of v -vel. nodes in y -dir. in total				✓	
M_total	Number	-	# of u -vel./pressure corr. nodes in y -dir. in total				✓	
m_wide	Number	-	# of v -vel. nodes in y -dir. under step				✓	
M_wide	Number	-	# of u -vel./pressure corr. nodes in y -dir. under step				✓	
maxits	Number	-	Stop if not converged after this number of iterations		✓	✓	✓	✓

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
mu	Number	Pa·s	Viscosity	μ	✓			
mu	Number	-	Dimensionless viscosity	$\hat{\mu}$		✓	✓	
mu_true	Number	Pa·s	Viscosity	μ		✓	✓	
N	Number	-	# of scalar nodes in x -dir.		✓	✓		
N	-	-	Length of q^1 -vector					✓
N_coeff	Number	-	Northern node coefficient in velocity or pressure corr. equation	\hat{a}_N		✓	✓	
N_narrow	Number	-	# of nodes in x -dir. in narrow channel				✓	
N_total	Number	-	# of nodes in x -dir. in total				✓	
N_wide	Number	-	# of nodes in x -dir. after step				✓	
nP_coeff	Number	-	Northern node contribution to centre node			✓	✓	
nptest	Boolean	-	True if node point is at northern boundary			✓	✓	✓
onlyChannel	Boolean	-	True if step section is disabled				✓	
P1	-	-	Poisson control function	P^1				✓
P2	-	-	Poisson control function	P^2				✓
p_atm	Number	Pa	Atmospheric pressure			✓	✓	
p_circ	Vector	Pa	Initial guess for pressure	p°	✓			
p_circ	Vector	-	Initial guess for pressure	\hat{p}°		✓	✓	
p_corr	Vector	Pa	Pressure correction	p'	✓			
p_corr	Vector	-	Pressure correction	\hat{p}'		✓	✓	
p_guess	Vector	Pa	Pressure guess	p°	✓			
p_guess	Vector	-	Pressure guess	p°		✓	✓	
p_new	Vector	Pa	Pressure for next iteration	p^{new}	✓			
p_new	Vector	-	Pressure for next iteration	\hat{p}^{new}		✓	✓	
p_out	Number	Pa	Outlet pressure	\hat{p}_{out}	✓			
p_out	Vector	-	Outlet pressure	\hat{p}_{out}		✓	✓	

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
p_out_tilde	Vector	-	Adjusted outlet pressure	\hat{p}_{out}		✓	✓	
plotinit...	Boolean	-	Option for plotting the initial guess profiles		✓	✓	✓	
q1	Vector	-	Curvilinear coordinate	q^1				✓
q2	Vector	-	Curvilinear coordinate	q^2				✓
Re	Number	-	Reynolds number	Re		✓	✓	
rho	Number	kg/m ³	Density	ρ	✓			
rho	Number	-	Dimensionless density	$\hat{\rho}$		✓	✓	
rho_true	Number	kg/m ³	Density	ρ		✓	✓	
runitera...	Boolean	-	Option for plotting after each iteration		✓	✓	✓	
s	Number		Distribution parameter for line segment AD					✓
S_coeff	Number		Southern node coefficient in velocity or pressure corr. equation	\hat{a}_S		✓	✓	
scorner	Boolean	-	True if node point is at the BFS corner				✓	
sP_coeff	Number		Southern node contribution to centre node			✓	✓	
stest	Boolean	-	True if node point is at southern boundary			✓	✓	✓
sys_width	Number	m	Height and width of the system in 1D	h	✓			
T	Matrix	sm	Coefficients for pressure	T	✓			
T	Matrix	-	Coefficients for pressure	\hat{T}		✓	✓	
totalpoints	Number	-	Total number of scalar points				✓	
totalpoin...	Number	-	Total number of v -velocity nodes				✓	
U	Matrix	kg/s	Coefficients for u -velocity	U	✓			
U	Matrix	-	Coefficients for u -velocity	\hat{U}		✓	✓	
u_bulk	Number	m/s	Bulk inlet u -velocity	u_{avg}			✓	
u_bulk_d...	Number		Dimensionless bulk inlet u -velocity	\hat{u}_{avg}			✓	
u_circ	Vector	m/s	Initial guess for u -velocity	u°	✓			

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
u_circ	Vector	-	Initial guess for u -velocity	\hat{u}°		✓	✓	
u_corr	Vector	Pa	u -velocity correction	u'	✓			
u_corr	Vector	-	u -velocity correction	\hat{u}'		✓	✓	
u_guess	Number	m/s	u -velocity guess		✓	✓	✓	
u_in	Number	m/s	u -velocity at inlet	u_{in}	✓			
u_in	Number	-	Dimensionless u -velocity at inlet	\hat{u}_{in}		✓	✓	
u_in	Vector	-	Dimensionless u -velocity profile at inlet	\hat{u}_{in}			✓	
u_in_true	Number	m/s	u -velocity at inlet	\hat{u}_{in}		✓	✓	
u_in_true	Vector	m/s	u -velocity profile at inlet	\hat{u}_{in}			✓	
u_new	Vector	Pa	u -velocity for next iteration	u^{new}	✓			
u_new	Vector	-	u -velocity for next iteration	\hat{u}^{new}		✓	✓	
u_max	Number	m/s	Max inlet u -velocity	u_{max}			✓	
u_star	Vector	-	u -velocity after matrix inversion	\hat{u}^*		✓	✓	
V	Matrix	-	Coefficients for v -velocity	\hat{U}		✓	✓	
v_circ	Vector	-	Initial guess for v -velocity	\hat{v}°		✓	✓	
v_corr	Vector	-	v -velocity correction	\hat{v}'		✓	✓	
v_guess	Number	m/s	v -velocity guess			✓	✓	
v_in	Number	-	Dimensionless v -velocity at inlet	\hat{v}_{in}		✓	✓	
v_in_true	Number	m/s	v -velocity at inlet	\hat{v}_{in}		✓	✓	
v_new	Vector	-	v -velocity for next iteration	\hat{v}^{new}		✓	✓	
v_star	Vector	-	v -velocity after matrix inversion	\hat{v}^*		✓	✓	
W_coeff	Number		Western node coefficient in velocity or pressure corr. equation	\hat{a}_W		✓	✓	
wP_coeff	Number		Western node contribution to centre node			✓	✓	
wtest	Boolean	-	True if node point is at western boundary			✓	✓	✓

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
wwall	Boolean	-	True if node point is at the wall after the BFS				✓	
X	Matrix	-	Coefficients for x					✓
x	Matrix	-	x -coordinate	x				✓
xA	Number	-	Location of point A , x -coordinate					✓
xAB	Vector	-	Boundary points, x -coordinate	x_{AB}				✓
xAD	Vector	-	Boundary points, x -coordinate	x_{AD}				✓
xAF	Vector	-	Boundary points, x -coordinate	x_{AF}				✓
xB	Number	-	Location of point B , x -coordinate					✓
xBC	Vector	-	Boundary points, x -coordinate	x_{BC}				✓
xC	Number	-	Location of point C , x -coordinate					✓
xD	Number	-	Location of point D , x -coordinate					✓
xDC	Vector	-	Boundary points, x -coordinate	x_{DC}				✓
xE	Number	-	Location of point E , x -coordinate					✓
xED	Vector	-	Boundary points, x -coordinate	x_{ED}				✓
xF	Number	-	Location of point F , x -coordinate					✓
xFE	Vector	-	Boundary points, x -coordinate	x_{FE}				✓
xx	Vector	-	x after matrix inversion	x				✓
x_mat	Matrix	-	x after matrix inversion	x				✓
x_max	Number	-	Total length of physical domain	L				✓
Y	Matrix	-	Coefficients for y					✓
y	Matrix	-	y -coordinate	y				✓
yA	Number	-	Location of point A , y -coordinate					✓
yAB	Vector	-	Boundary points, y -coordinate	y_{AB}				✓
yAD	Vector	-	Boundary points, y -coordinate	y_{AD}				✓
yAF	Vector	-	Boundary points, y -coordinate	y_{AF}				✓

Continued on next page

Continued from previous page

Name	Type	Unit	Description	Symbol	Appears in			
					1D	2D	BFS	GG
yB	Number	-	Location of point B , y -coordinate					✓
yBC	Vector	-	Boundary points, y -coordinate	y_{BC}				✓
yC	Number	-	Location of point C , y -coordinate					✓
yD	Number	-	Location of point D , y -coordinate					✓
yDC	Vector	-	Boundary points, y -coordinate	y_{DC}				✓
yE	Number	-	Location of point E , y -coordinate					✓
yED	Vector	-	Boundary points, y -coordinate	y_{ED}				✓
yF	Number	-	Location of point F , y -coordinate					✓
yFE	Vector	-	Boundary points, y -coordinate	y_{FE}				✓
yy	Vector	-	y after matrix inversion	y				✓
y_mat	Matrix	-	y after matrix inversion	y				✓
y_max	Number	-	Total height of physical domain	H				✓

E.3 One Dimensional Straight Channel

The code `channel_1D.m` solves and plots the solution to the one dimensional flow problem.

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %           One dimensional fluid flow in a straight channel           %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  clc
6  clear
7  close all
8  tic
9
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 %% Solver specifications
12 maxits = 20000;
13 N = 100; % Number of scalar nodal points
14 runiterationwise = 0; % Plots the profiles after each iteration
15 plotinitialprofiles = 0; % Plot the initial guesses
16
17 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
18 %% Initial guesses
19
20 p_out = 1e5; % Pa
21 u_in = 1e-3; % m/s
22 p_guess = 1.5e5; % Pa
23 u_guess = 1.5e-3; % m/s
24
25 % Creating a linear profile for the initial guess of the pressure
26 pprofile = [linspace(p_guess,p_out,N+1), p_out];
27 p_circ = pprofile(2:end-1);
28
29 % Creating a linear profile for the initial guess of the velocity
30 uprofile = linspace(u_in,u_guess,N+1);
31 % Placing the velocities in the staggered grid
32 u_circ = 0.5*(uprofile(2:end)+uprofile(1:end-1));
33
34 alpha_u = 1; % Under-relaxation of the velocity
35 alpha_p = 0.05; % Under-relaxation of the pressure
36 % 1 corresponds to no under-relaxation
37
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 %% Parameters and system specifications
40 L = 3; % Channel length [m]
41
42 x_0 = 0; % Channel start [m]
43 x_N = L; % Channel end [m]
44 mu = 8.90 * 10^-4; % Viscosity [Pa s]
45
46 sys_width = 1; % Height of the channel is set to unity
47 % for the one dimensional model
48
49 del_x = x_N/N; % Width of the control volume [m]
50 A = del_x*sys_width;% Cross-sectional area [m^2]
51
52
53 rho = 1e3; % Density [kg/m^3]
54
55 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
56 %% Initialisation
57
58 p_corr = zeros(1, N);
59 p_new = zeros(1, N);
60
61 u_star = zeros(1, N);
62 u_corr = zeros(1, N);
63 u_new = zeros(1, N);
64
65 F_e = zeros(1, N);
66 F_w = zeros(1, N);
67 D = mu/del_x;
68
69 U = zeros(N, N);
70 bu = zeros(1, N);

```

```

71
72 T = zeros(N, N);
73 beta = zeros(1, N);
74 a_u = zeros(1, N);
75
76 xu_plot = linspace(x_0, x_N, N+1); % staggered grid
77 xp_plot = linspace(x_0+del_x/2, x_N+del_x/2, N+1);
78
79 if plotinitialprofiles == 1
80     figure
81     plot(xu_plot, [u_in, u_circ])
82     hold on
83     plot(xu_plot(1:2), [u_in, u_circ(1)], 'r')
84     title('Initial guess $u$', 'interpreter', 'latex')
85     figure
86     plot(xp_plot, [p_circ, p_out])
87     hold on
88     plot(xp_plot(end-1:end), [p_circ(end), p_out], 'r')
89     title('Initial guess $p$', 'interpreter', 'latex')
90 end %if
91
92 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
93 %% While loop
94 conv = 0;
95 it = 1;
96 while conv == 0
97
98     %% Solve momentum equation
99     % Calculation of coefficients F_e:
100    for i = 1:length(u_circ)-1
101        F_e(i) = rho*1/2*(u_circ(i+1)+u_circ(i));
102    end %for
103    F_e(end) = rho*1/2*(u_circ(end)+u_circ(end-1)); % F_e = F_w
104
105    % Calculation of coefficients F_w:
106    F_w(1) = rho*1/2*(u_circ(1)+u_in);
107    for i = 2:length(u_circ)
108        F_w(i) = rho*1/2*(u_circ(i)+u_circ(i-1));
109    end %for
110
111    % u_2 (u(1))
112    U(1,1) = 4*D*A + max(0, -F_e(1)*A) + max(F_w(1)*A, 0)...
113            + F_e(1)*A - F_w(1)*A;
114    U(1,2) = -2*D*A - max(0, -F_e(1)*A);
115    bu(1) = ( 2*D*A + max(F_w(1)*A, 0))*u_in ...
116            - A*(p_circ(2) - p_circ(1));
117
118
119    % u_centers
120    for j = 2:length(u_circ)-1
121        U(j,j) = 4*D*A + max(0, -F_e(j)*A) + max(F_w(j)*A, 0) ...
122            + F_e(j)*A - F_w(j)*A ;
123        U(j,j+1) = -2*D*A - max(0, -F_e(j)*A);
124        U(j,j-1) = -2*D*A - max(F_w(j)*A, 0);
125        bu(j) = - (p_circ(j+1) - p_circ(j))*A;
126
127
128    end %for
129
130    % u_{n+1} (u(end))
131    U(end,end) = 4*D*A + max(F_w(end)*A, 0) ...
132            + F_e(end)*A - F_w(end)*A ;
133    U(end,end-1) = -2*D*A - max(F_w(end)*A, 0);
134    bu(end) = - (p_out - p_circ(end))*A;
135
136    % Matrix inversion
137    u_star = U\bu';
138
139    %% Solve pressure correction equation
140    % a^center-coefficients in the momentum equation
141    for i = 1:length(U)
142        a_u(i) = U(i,i);
143    end %for
144
145    % p_1
146    T(1,1) = rho*A/a_u(1);

```

```

147 T(1,2) = - rho*A/a_u(1);
148 beta(1) = rho*(-u_star(1) + u_in);
149
150 % p_centers
151 for j = 2:length(p_corr)-1
152     T(j,j) = rho*A*(1/a_u(j) + 1/a_u(j-1));
153     T(j,j+1) = - rho*A/a_u(j);
154     T(j,j-1) = - rho*A/a_u(j-1);
155     beta(j) = rho*(-u_star(j) + u_star(j-1));
156 end %for
157
158 % p_N
159 T(end,end) = rho*A*(1/a_u(end)+1/a_u(end-1));
160 T(end,end-1) = - rho*A/a_u(end);
161 beta(end) = rho*(-u_star(end) + u_star(end-1));
162
163 % Matrix inversion
164 p_corr = T\beta';
165
166 %% Velocity correction
167 for j = 1:length(p_corr)-1
168     u_corr(j) = - A/a_u(j)*(p_corr(j+1)-p_corr(j));
169 end %for
170 u_corr(end) = - A/a_u(end)*(-p_corr(end));
171 % pressure correction is zero for the known outlet pressure
172
173 %% Under-relaxation
174 % Pressure
175 p_new = p_circ + alpha_p* p_corr';
176
177 % Under-relaxation of u
178 u_new = alpha_u*(u_star' + u_corr) + (1-alpha_u)*u_circ;
179
180 %% Check convergence
181 if isnan(rcond(U)) || isnan(rcond(T))
182     fprintf('Stopped due to singularity in matrix\n')
183     fprintf('RCOND velocity: %e \nRCOND pressure: %e\n',...
184         rcond(U), rcond(T))
185     fprintf('Problem occured after %d iterations\n', it-1)
186     return
187 end %if
188
189 c1 = 1/u_in*sqrt((U*u_star-bu')*(U*u_star-bu')); % coefficient summed
190 c3 = abs(sum(beta)); % continuity
191 c4 = 1/u_in*max(abs(u_circ - u_star')); % change from last iteration
192
193 c1_lim = 10^-6;
194 c3_lim = 10^-6;
195 c4_lim = 10^-6;
196
197 c1_diff = c1-c1_lim;
198 c3_diff = c3-c3_lim;
199 c4_diff = c4-c4_lim;
200
201 if (c1 < c1_lim) && (c3 < c3_lim) && (c4 < c4_lim) || (it == maxits)
202     conv = 1; % While loop is stopped
203     if (it == maxits)
204         fprintf('Stopped at max iterations (%d)\n',it);
205     else
206         fprintf('Solution converged after %d iterations\n',it);
207     end %if
208
209     fprintf('c1\tMomentum residual\t\t%.2e\tLimit: %.2e\n',c1,c1_lim);
210     fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',c3,c3_lim);
211     fprintf('c4\tDiff. last iteration\t\t%.2e\tLimit: %.2e\n',c4,c4_lim);
212
213     if max([c1_diff c3_diff c4_diff])== c1_diff
214         fprintf('Limiting criteria is c1\tMomentum residual\n')
215     elseif max([c1_diff c3_diff c4_diff])== c3_diff
216         fprintf('Limiting criteria is c3\tPressure correction\n')
217     elseif max([c1_diff c3_diff c4_diff])== c4_diff
218         fprintf('Limiting criteria is c4\tDiff. last iteration\n')
219     end %if
220
221
222 else

```

```

223     u_circ = u_new; % Not converged, updated variables.
224     p_circ = p_new;
225     it = it + 1;
226
227
228     end %if
229     if runiterationwise == 1 || conv == 1
230         % For iterationwise plotting and for when the model is stopped
231         % Plot after each iteration and close before proceeding to the next
232
233         u_new_plot = [u_in u_new];
234
235         % Discretied x-node points
236         p_plot = [p_new p_out];
237         p_corr_plot = [p_corr' 0];
238
239         fu = figure;
240         plot(xu_plot, u_new_plot)
241         s = sprintf('Plot of $u^{new}$ after %d iterations', it-1 );
242         %     f = title(s);
243         %     set(f, 'interpreter', 'latex', 'fontsize', 16)
244         set(gca, 'TickLabelInterpreter', 'latex')
245         xlabel('$x$-direction [m]', 'interpreter', 'latex')
246         xlim([0,3])
247         ylabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
248         %     set(fu, 'Position', [5,217,414.6667,420]);
249         % [left bottom width height]
250         saveas(gcf, 'unew1D.png')
251
252
253         fp = figure;
254         plot(xp_plot, p_plot)
255         s = sprintf('Plot of $p^{new}$ after %d iterations', it-1 );
256         %     f = title(s);
257         %     set(f, 'interpreter', 'latex', 'fontsize', 16)
258         set(gca, 'TickLabelInterpreter', 'latex')
259         xlabel('$x$-direction [m]', 'interpreter', 'latex')
260         xlim([0,3])
261         ylabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
262         %     set(fp, 'Position', [419.6667,217,434.6667,420]);
263         % [left bottom width height]
264         saveas(gcf, 'pnew1D.png')
265
266         fpcorr = figure;
267         plot(xp_plot, p_corr_plot)
268         s = sprintf('Plot of $p^{corr}$ after %d iterations', it-1 );
269         %     f = title(s);
270         %     set(f, 'interpreter', 'latex', 'fontsize', 16)
271         set(gca, 'TickLabelInterpreter', 'latex')
272         xlabel('$x$-direction [m]', 'interpreter', 'latex')
273         xlim([0,3])
274         ylabel('Pressure correction $p$, [Pa]', 'interpreter', 'latex')
275         %     set(fpcorr, 'Position', [855,217.6667,424,422.6667]);
276         % [left bottom width height]
277         saveas(gcf, 'pcorr1D.png')
278
279         if conv ~= 1 % if not converged
280             pause
281             close all
282         end %if
283     end % if
284
285 end %while
286
287 toc

```

E.4 Two Dimensional Straight Channel

The code `channel_2D.m` solves the two dimensional flow problem. The code `plot_2D.m` plots the solution to the two dimensional flow problem.

E.4.1 Codes

E.4.1.1 `channel_2D.m`

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %   Two dimensional fluid flow in a straight channel, dimensionless   %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  clear
5  clc
6  close all
7  tic
8
9  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10 %% Solver specifications
11 maxits = 100000; % Maximum number of iterations, stop if iterations exceed
12 N = 88;          % Number of scalar nodal points in x-direction
13 M = 18;          % Number of scalar nodal points in y-direction
14 runiterationwise = 0; % Plots the profiles after each iteration
15 plotinitialprofiles = 0; % Plot the initial guesses
16 solvvel = true; % Solve for v-velocity
17 contplots = false; % Show plots of continuity + cont_x and cont_y
18 v_out_zero = false; % Use v_out = zero as boundary condition
19
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21 %% System specifications
22 m = M - 1; % Number of y-velocity nodes in y-direction
23 L = 22; % Channel length
24 h = 1; % Channel height
25 D_hyd = 4*h*1/(1+1+h+h); % Hydraulic diameter for Reynolds number
26
27 x_0 = 0; % Defining the domain using x and y
28 x_N = L;
29 y_0 = 0;
30 y_M = h;
31
32 mu_true = 8.90 * 10^-4; % Viscosity of water
33
34 del_z_true = 1; % System depth
35 del_x_true = x_N/N; % Control volume width
36 del_y_true = y_M/M; % Control volume height
37 A_x_true = del_y_true*del_z_true; % Cross-sectional area in x-direction
38 A_y_true = del_x_true*del_z_true; % Cross-sectional area in y-direction
39
40 rho_true = 997; % Density of water
41 u_in_true = 0.0005; % Inlet u-velocity
42 g_x = 0; % No gravitation
43 g_y = 0; % No gravitation
44
45
46 Re = rho_true*D_hyd*u_in_true/mu_true; % Reynolds number
47
48 p_atm = 101325; % Atmospheric pressure at outlet
49 p_out_tilde = 0; % Adjusted pressure
50 p_out = ones(1,M)*p_out_tilde; % Outlet pressure profile
51
52 alpha_u = 0.01; % Under-relaxation factor for u
53 alpha_v = 0.01; % Under-relaxation factor for v
54 alpha_p = 0.02; % Under-relaxation factor for p
55
56 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
57 %% Dimensionless parameters
58 mu = 1; % Dimensionless viscosity
59 rho = 1; % Dimensionless density
60 del_x = del_x_true/D_hyd; % Dimensionless control volume width
61 del_y = del_y_true/D_hyd; % Dimensionless control volume height
62 A_x = A_x_true/D_hyd^2; % Dimensionless cross-sectional area in x-direction
63 A_y = A_y_true/D_hyd^2; % Dimensionless cross-sectional area in y-direction

```



```

64 D_x = 1/Re*mu/del_x; % Dimensionless diffusion conductance in x-direction
65 D_y = 1/Re*mu/del_y; % Dimensionless diffusion conductance in y-direction
66 u_in = 1; % Inlet u-velocity
67 v_in = 0; % Inlet v-velocity
68 u_guess = 1.0; % Initial guess for u-velocity
69 v_guess = 0.0; % Initial guess for v-velocity
70 u_circ = ones(1,M*N)*u_guess; % Initial guess vector for u-velocity
71 v_circ = ones(1,m*N)*v_guess; % Initial guess vector for v-velocity
72 p_guess = 0/(rho_true*u_in_true^2); % Initial guess for pressure
73 p_circ_vector = linspace(p_guess,p_out_tilde,N)'; % Linear profile from
74 % guess to known outlet pressure
75 p_circ = zeros(M*N,1);
76 for j = 1:M % Filling in initial pressure vector with the linear profile
77     p_circ((j-1)*N+1:j*N) = p_circ_vector;
78 end %for
79
80 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
81 %% Initialization of solution vectors
82 p_corr = zeros(1, M*N); % Pressure correction
83 p_new = zeros(1, M*N); % New pressure
84
85 u_star = zeros(1, M*N); % u-velocity after matrix inversion
86 u_corr = zeros(1, M*N); % u-velocity correction
87 u_new = zeros(1, M*N); % New u-velocity
88 U = zeros(M*N, M*N); % u-velocity coefficient matrix
89 bu = zeros(1, M*N); % u-velocity source term vector
90
91 v_star = zeros(1, m*N); % v-velocity after matrix inversion
92 v_corr = zeros(1, m*N); % v-velocity correction
93 v_new = zeros(1, m*N); % New v-velocity
94 V = zeros(m*N, m*N); % v-velocity coefficient matrix
95 bv = zeros(1, m*N); % v-velocity source term vector
96
97 F_xe = zeros(1, M*N); % Convective mass flux per unit area
98 F_xw = zeros(1, M*N);
99 F_xn = zeros(1, M*N);
100 F_xs = zeros(1, M*N);
101 F_ye = zeros(1, m*N);
102 F_yw = zeros(1, m*N);
103 F_yn = zeros(1, m*N);
104 F_ys = zeros(1, m*N);
105
106 T = zeros(M*N, M*N); % Pressure correction coefficient matrix
107 % for pressure
108 beta = zeros(1, M*N); % Pressure correction source term vector
109 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
110 %% Plots of initial guesses
111 if plotinitialprofiles == true
112     f11 = figure;
113     surf(linspace(x_0+del_x/2, x_N+del_x/2, N+1),...
114          linspace(y_0+del_y/2, y_M-del_y/2, M),...
115          [global2matrix(p_circ,N,M) p_out]); % surf(x,y,z)
116     s = sprintf('Initial guess $p_{circ}$');
117     f = title(s);
118     set(f, 'interpreter', 'latex', 'fontsize', 16)
119     set(gca, 'TickLabelInterpreter', 'latex')
120     xlabel('$x$-direction [m]', 'interpreter', 'latex')
121     ylabel('$y$-direction [m]', 'interpreter', 'latex')
122     zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
123
124     u_circ_carthesian = ones(M,N+1)*u_guess;
125     u_circ_carthesian(:,1) = u_in;
126
127     f122 = figure;
128     surf(linspace(x_0, x_N, N+1),... % surf(x,y,z)
129          linspace(y_0+del_y/2, y_M-del_y/2, M),u_circ_carthesian);
130     s = sprintf('Initial guess $u_{circ}$');
131     f = title(s);
132     set(f, 'interpreter', 'latex', 'fontsize', 16)
133     set(gca, 'TickLabelInterpreter', 'latex')
134     xlabel('$x$-direction [m]', 'interpreter', 'latex')
135     ylabel('$y$-direction [m]', 'interpreter', 'latex')
136     zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
137
138     v_circ_carthesian = ones(M+1,N+1)*v_guess;

```

```

139     v_circ_carthesian(1,:) = 0;
140     v_circ_carthesian(M+1,:) = 0;
141     v_circ_carthesian(:,1) = 0;
142
143     f133 = figure;
144     surf(linspace(-x_0-del_x/2, x_N+del_x/2, N+1),...
145         linspace(y_0, y_M, M+1),v_circ_carthesian);           % surf(x,y,z)
146     % set(f,'edgecolor','none')
147     s = sprintf('Initial guess $v_{circ}$');
148     f = title(s);
149     set(f, 'interpreter', 'latex', 'fontsize', 16)
150     set(gca,'TickLabelInterpreter','latex')
151     xlabel('$x$-direction [m]', 'interpreter', 'latex')
152     ylabel('$y$-direction [m]', 'interpreter', 'latex')
153     zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
154     pause
155     close all
156     end % if
157
158     % Defining x and y points for the staggered grid for plotting
159     xu_plot = linspace(x_0, x_N, N+1);
160     yu_plot = [0,linspace(y_0+del_y/2, y_M-del_y/2, M),h];
161     xv_plot = [0,linspace(x_0+del_x/2, x_N-del_x/2, N)];
162     yv_plot = linspace(y_0, y_M, M+1);
163     xp_plot = linspace(x_0+del_x/2, x_N+del_x/2, N+1) + del_x_true/2;
164     yp_plot = linspace(y_0+del_y/2, y_M-del_y/2, M) + del_y_true/2;
165
166     %% Specifications before iteration
167     if solvvel == false % Not solve for v-velocity
168         v_out_zero = false; % Turn off outlet boundary condition for v
169     end %if
170
171     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
172     %% While loop
173     conv = 0; % 0 is not converged, 1 when converged
174     it = 1; % The current iteration
175
176     % Coefficients in matrix, example:
177     % sP_coeff is part of the a_P-coefficient at the diagonal position in the
178     % matrix, while S_coeff is the coefficient in the matrix for the south node
179     while conv == 0 % it<=maxits %
180         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
181         %% Generation of F
182         %Generation of F_x:
183         for i = 1:M*N
184
185             etest = mod(i, N) == 0;
186             wtest = mod(i-1, N) == 0;
187             ntest = M*N - (N - 1) <= i && i <= N*M ;
188             stest = 1 <= i && i <= N ;
189
190             % Northeastern corner
191             if etest == true && ntest == true
192                 F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i)); % F_xe = F_xw
193                 F_xn(i) = 0; % v_NorthWall = 0;
194
195                 F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
196                 F_xs(i) = rho/2*v_circ(i-N);
197
198             % Southeastern corner
199             elseif etest == true && stest == true
200                 F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i)); % F_xe = F_xw
201                 F_xs(i) = 0; % v_SouthWall = 0;
202
203                 F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
204                 F_xn(i) = rho/2*v_circ(i);
205
206             % Northwestern corner
207             elseif wtest == true && ntest == true
208                 F_xw(i) = rho/2*(u_in+u_circ(i)); % Inlet
209                 F_xn(i) = 0; % v_NorthWall = 0;
210
211                 F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
212                 F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));
213
214             % Southwestern corner

```

```

215     elseif wtest == true && stest == true
216         F_xw(i) = rho/2*(u_in+u_circ(i));           % Inlet
217         F_xs(i) = 0;                               % v_SouthWall = 0;
218
219         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
220         F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
221
222     % At eastern boundary (x = L)
223     elseif etest == true && ntest == false && stest == false
224         F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));   % F_xe = F_xw
225
226         F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
227         F_xn(i) = rho/2*v_circ(i);
228         F_xs(i) = rho/2*v_circ(i-N);
229
230     % At western boundary (x = 0)
231     elseif wtest == true && ntest == false && stest == false
232         F_xw(i) = rho/2*(u_in+u_circ(i));           % Inlet
233
234         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
235         F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
236         F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));
237
238     % At northern boundary (y = h)
239     elseif ntest == true && etest == false && wtest == false
240         F_xn(i) = 0;                               % v_NorthWall = 0;
241
242         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
243         F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
244         F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));
245
246     % At southern boundary (y = 0)
247     elseif stest == true && etest == false && wtest == false
248         F_xs(i) = 0;                               % v_SouthWall = 0;
249
250         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
251         F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
252         F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
253
254     %Not at any boundary
255     else
256         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
257         F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
258         F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
259         F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));
260
261     end % if
262
263     etest = false;
264     wtest = false;
265     ntest = false;
266     stest = false;
267
268 end %for
269
270 %Generation of F_y:
271 for i = 1:m*N % Global indexing system
272
273     % Eastern boundary requires no special treatment (x = L)
274     wtest = mod(i-1, N) == 0;
275     ntest = m*N - (N - 1) <= i && i <= m*N ;
276     stest = 1 <= i && i <= N ;
277
278     % Northwestern corner
279     if wtest == true && ntest == true
280         F_yw(i) = rho*u_in;                         % inlet
281         F_yn(i) = rho/2*v_circ(i);                 % v_NorthWall = 0;
282
283         F_ye(i) = rho/2*(u_circ(i) + u_circ(i+N));
284         F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));
285
286     % Southwestern corner
287     elseif wtest == true && stest == true
288         F_yw(i) = rho*u_in;                         % inlet
289         F_ys(i) = rho/2*v_circ(i);                 % v_SouthWall = 0;
290

```

```

291         F_yc(i) = rho/2*(u_circ(i) + u_circ(i+N));
292         F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));
293
294     % At western boundary (x = 0)
295     elseif wtest == true && ntest == false && stest == false
296         F_yw(i) = rho*u_in; % inlet
297
298         F_yc(i) = rho/2*(u_circ(i) + u_circ(i+N));
299         F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));
300         F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));
301
302     % At northern boundary (y = h)
303     elseif ntest == true && wtest == false
304         F_yn(i) = rho/2*v_circ(i); % v_NorthWall = 0;
305
306         F_yc(i) = rho/2*(u_circ(i) + u_circ(i+N));
307         F_yw(i) = rho/2*(u_circ(i-1) + u_circ(i-1+N));
308         F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));
309
310     % At southern boundary (y = 0)
311     elseif stest == true && wtest == false
312         F_ys(i) = rho/2*v_circ(i); % v_SouthWall = 0;
313
314         F_yc(i) = rho/2*(u_circ(i) + u_circ(i+N));
315         F_yw(i) = rho/2*(u_circ(i-1) + u_circ(i-1+N));
316         F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));
317
318     %Not at any boundary
319     else
320         F_yc(i) = rho/2*(u_circ(i) + u_circ(i+N));
321         F_yw(i) = rho/2*(u_circ(i-1) + u_circ(i-1+N));
322         F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));
323         F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));
324
325     end % if
326
327     etest = false;
328     wtest = false;
329     ntest = false;
330     stest = false;
331
332 end % for
333
334 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
335 %% u-velocity
336 for i = 1:M*N % Global indexing system
337
338     etest = mod(i, N) == 0;
339     wtest = mod(i-1, N) == 0;
340     ntest = M*N - (N - 1) <= i && i <= N*M ;
341     stest = 1 <= i && i <= N ;
342
343     % Northeastern corner
344     if etest == true && ntest == true
345         % At eastern boundary (x = L)
346         E_coeff = -max(0, -F_xe(i)*A_x) - D_x*A_x;
347         eP_coeff = F_xe(i)*A_x;
348
349         bu(i) = -(p_out(end)-p_circ(i))*A_x;
350
351         % At northern boundary
352         nP_coeff = F_xn(i)*A_y + max(0, -F_xn(i)*A_y) + 2*D_y*A_y;
353         %wall shear stress
354
355         W_coeff = -max(F_xw(i)*A_x, 0) - D_x*A_x;
356         wP_coeff = -W_coeff - F_xw(i)*A_x;
357         U(i, i-1) = W_coeff;
358
359         S_coeff = -max(F_xs(i)*A_y, 0) - D_y*A_y;
360         sP_coeff = -S_coeff - F_xs(i)*A_y;
361         U(i, i-N) = S_coeff;
362
363     % Southeastern corner
364     elseif etest == true && stest == true
365         % At eastern boundary (x = L)
366         E_coeff = -max(0, -F_xe(i)*A_x) - D_x*A_x;

```

```

367         eP_coeff = F_xe(i)*A_x;
368
369         bu(i) = -(p_out(1)-p_circ(i))*A_x;
370
371         % At southern boundary (y = 0)
372         sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
373         %wall shear stress
374
375         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
376         wP_coeff = -W_coeff - F_xw(i)*A_x;
377         U(i, i-1) = W_coeff;
378
379         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
380         nP_coeff = -N_coeff + F_xn(i)*A_y;
381         U(i, i+N) = N_coeff;
382
383     % Northwestern corner
384     elseif wtest == true && ntest == true
385         % At western boundary (x = 0)
386         wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
387
388         % At northern boundary
389         nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;
390         %wall shear stress
391
392         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
393             +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;
394
395         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
396         eP_coeff = -E_coeff + F_xe(i)*A_x;
397         U(i, i+1) = E_coeff;
398
399         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
400         sP_coeff = -S_coeff - F_xs(i)*A_y;
401         U(i, i-N) = S_coeff;
402
403     % Southwestern corner
404     elseif wtest == true && stest == true
405         % At western boundary (x = 0)
406         wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
407
408         % At southern boundary (y = 0)
409         sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
410         %wall shear stress
411
412         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
413             +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;
414
415         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
416         eP_coeff = -E_coeff + F_xe(i)*A_x;
417         U(i, i+1) = E_coeff;
418
419         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
420         nP_coeff = -N_coeff + F_xn(i)*A_y;
421         U(i, i+N) = N_coeff;
422
423     % At eastern boundary (x = L)
424     elseif etest == true && ntest == false && stest == false
425         % At eastern boundary (x = L)
426         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
427         eP_coeff = F_xe(i)*A_x;
428
429         bu(i) = -(p_out(floor((i-1)/N)+1)-p_circ(i))*A_x;
430
431         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
432         wP_coeff = -W_coeff - F_xw(i)*A_x;
433         U(i, i-1) = W_coeff;
434
435         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
436         nP_coeff = -N_coeff + F_xn(i)*A_y;
437         U(i, i+N) = N_coeff;
438
439         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
440         sP_coeff = -S_coeff - F_xs(i)*A_y;
441         U(i, i-N) = S_coeff;
442

```

```

443 % At western boundary (x = 0)
444 elseif wtest == true && ntest == false && stest == false
445 % At western boundary (x = 0)
446 wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
447
448 bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
449 + (max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;
450
451 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
452 eP_coeff = -E_coeff + F_xe(i)*A_x;
453 U(i, i+1) = E_coeff;
454
455 N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
456 nP_coeff = -N_coeff + F_xn(i)*A_y;
457 U(i, i+N) = N_coeff;
458
459 S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
460 sP_coeff = -S_coeff - F_xs(i)*A_y;
461 U(i, i-N) = S_coeff;
462
463 % At northern boundary (y = h)
464 elseif ntest == true && etest == false && wtest == false
465 % At northern boundary
466 nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y) + 2*D_y*A_y;
467 %wall shear stress
468
469 bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
470 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
471 eP_coeff = -E_coeff + F_xe(i)*A_x;
472 U(i, i+1) = E_coeff;
473
474 W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
475 wP_coeff = -W_coeff - F_xw(i)*A_x;
476 U(i, i-1) = W_coeff;
477
478 S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
479 sP_coeff = -S_coeff - F_xs(i)*A_y;
480 U(i, i-N) = S_coeff;
481
482 % At southern boundary (y = 0)
483 elseif stest == true && etest == false && wtest == false
484 % At southern boundary (y = 0)
485 sP_coeff = -F_xs(i)*A_y + max(F_xs(i)*A_y,0) + 2*D_y*A_y;
486 %wall shear stress
487
488 bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
489
490 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
491 eP_coeff = -E_coeff + F_xe(i)*A_x;
492 U(i, i+1) = E_coeff;
493
494 W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
495 wP_coeff = -W_coeff - F_xw(i)*A_x;
496 U(i, i-1) = W_coeff;
497
498 N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
499 nP_coeff = -N_coeff + F_xn(i)*A_y;
500 U(i, i+N) = N_coeff;
501
502 %Not at any boundary
503 else
504 bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
505 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
506 eP_coeff = -E_coeff + F_xe(i)*A_x;
507 U(i, i+1) = E_coeff;
508
509 W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
510 wP_coeff = -W_coeff - F_xw(i)*A_x;
511 U(i, i-1) = W_coeff;
512
513 N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
514 nP_coeff = -N_coeff + F_xn(i)*A_y;
515 U(i, i+N) = N_coeff;
516
517 S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
518 sP_coeff = -S_coeff - F_xs(i)*A_y;

```

```

519         U(i, i-N) = S_coeff;
520
521     end % if
522
523     % Filling in the rest of the matrix, adding all point coefficients
524     U(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
525
526     etest = false;
527     wtest = false;
528     ntest = false;
529     stest = false;
530
531 end %for
532
533 u_star = U\bu';
534
535 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
536 % v-velocity
537 for i = 1:m*N % Global indexing system
538     bv(i) = -(p_circ(i+N)-p_circ(i))*A_y + rho*g_y*del_y*A_y;
539
540     etest = mod(i, N) == 0;
541     wtest = mod(i-1, N) == 0;
542     ntest = m*N - (N - 1) <= i && i <= m*N ;
543     stest = 1 <= i && i <= N ;
544
545     % Northeastern corner
546     if etest == true && ntest == true
547
548         % At eastern boundary (x = L)
549         E_coeff = -max(0,-F_yc(i)*A_x) - D_x*A_x;
550         eP_coeff = F_yc(i)*A_x;
551
552         if v_out_zero == true
553             eP_coeff = eP_coeff + 1e+30;
554         end %if
555
556         % At northern boundary
557         nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y;
558
559         W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
560         wP_coeff = -W_coeff - F_yw(i)*A_x;
561         V(i, i-1) = W_coeff;
562
563         S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
564         sP_coeff = -S_coeff - F_ys(i)*A_y;
565         V(i, i-N) = S_coeff;
566
567     % Southeastern corner
568     elseif etest == true && stest == true
569
570         % At eastern boundary (x = L)
571         E_coeff = -max(0,-F_yc(i)*A_x) - D_x*A_x;
572         eP_coeff = F_yc(i)*A_x;
573         if v_out_zero == true
574             eP_coeff = eP_coeff + 1e+30;
575         end %if
576         % At southern boundary (y = 0),
577         sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
578
579         W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
580         wP_coeff = -W_coeff - F_yw(i)*A_x;
581         V(i, i-1) = W_coeff;
582
583         N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
584         nP_coeff = -N_coeff + F_yn(i)*A_y;
585         V(i, i+N) = N_coeff;
586
587     % Northwestern corner
588     elseif wtest == true && ntest == true
589
590         % At western boundary (x = 0)
591         wP_coeff = -F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
592
593         % At northern boundary
594         nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;

```

```

595     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
596     eP_coeff = -E_coeff + F_ye(i)*A_x;
597     V(i, i+1) = E_coeff;
598
599
600     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
601     sP_coeff = -S_coeff - F_ys(i)*A_y;
602     V(i, i-N) = S_coeff;
603
604 % Southwestern corner
605 elseif wtest == true && stest == true
606
607     % At western boundary (x = 0)
608     wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
609
610     % At southern boundary (y = 0),
611     sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
612
613     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
614     eP_coeff = -E_coeff + F_ye(i)*A_x;
615     V(i, i+1) = E_coeff;
616
617     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
618     nP_coeff = -N_coeff + F_yn(i)*A_y;
619     V(i, i+N) = N_coeff;
620
621 % At eastern boundary (x = L)
622 elseif etest == true && ntest == false && stest == false
623
624     % At eastern boundary (x = L)
625     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
626     eP_coeff = F_ye(i)*A_x;
627     if v_out_zero == true
628         eP_coeff = eP_coeff + 1e+30;
629     end %if
630
631     W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
632     wP_coeff = -W_coeff - F_yw(i)*A_x;
633     V(i, i-1) = W_coeff;
634
635     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
636     nP_coeff = -N_coeff + F_yn(i)*A_y;
637     V(i, i+N) = N_coeff;
638
639     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
640     sP_coeff = -S_coeff - F_ys(i)*A_y;
641     V(i, i-N) = S_coeff;
642
643 % At western boundary (x = 0)
644 elseif wtest == true && ntest == false && stest == false
645
646     % At western boundary (x = 0)
647     wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
648
649     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
650     eP_coeff = -E_coeff + F_ye(i)*A_x;
651     V(i, i+1) = E_coeff;
652
653     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
654     nP_coeff = -N_coeff + F_yn(i)*A_y;
655     V(i, i+N) = N_coeff;
656
657     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
658     sP_coeff = -S_coeff - F_ys(i)*A_y;
659     V(i, i-N) = S_coeff;
660
661 % At northern boundary (y = h)
662 elseif ntest == true && etest == false && wtest == false
663
664     % At northern boundary
665     nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;
666
667     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
668     eP_coeff = -E_coeff + F_ye(i)*A_x;
669     V(i, i+1) = E_coeff;
670

```



```

671         W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
672         wP_coeff = -W_coeff - F_yw(i)*A_x;
673         V(i, i-1) = W_coeff;
674
675         S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
676         sP_coeff = -S_coeff - F_ys(i)*A_y;
677         V(i, i-N) = S_coeff;
678
679     % At southern boundary (y = 0)
680     elseif stest == true && etest == false && wtest == false
681
682         % At southern boundary (y = 0),
683         sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
684
685         E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
686         eP_coeff = -E_coeff + F_ye(i)*A_x;
687         V(i, i+1) = E_coeff;
688
689         W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
690         wP_coeff = -W_coeff - F_yw(i)*A_x;
691         V(i, i-1) = W_coeff;
692
693         N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
694         nP_coeff = -N_coeff + F_yn(i)*A_y;
695         V(i, i+N) = N_coeff;
696
697     %Not at any boundary
698     else
699
700         E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
701         eP_coeff = -E_coeff + F_ye(i)*A_x;
702         V(i, i+1) = E_coeff;
703
704         W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
705         wP_coeff = -W_coeff - F_yw(i)*A_x;
706         V(i, i-1) = W_coeff;
707
708         N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
709         nP_coeff = -N_coeff + F_yn(i)*A_y;
710         V(i, i+N) = N_coeff;
711
712         S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
713         sP_coeff = -S_coeff - F_ys(i)*A_y;
714         V(i, i-N) = S_coeff;
715
716     end % if
717
718     % Filling in the rest of the matrix, adding all point coefficients
719     V(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
720
721     etest = false;
722     wtest = false;
723     ntest = false;
724     stest = false;
725
726 end % for
727 v_star = V\bv';
728 if solvvel == false
729     v_star = zeros(length(v_star),1);
730 end %if
731
732 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
733 %% Pressure correction
734 au = diag(U); % a^center-coefficients for u-velocity
735 av = diag(V); % a^center-coefficients for v-velocity
736
737 for i = 1:M*N % Global indexing system
738
739     etest = mod(i, N) == 0;
740     wtest = mod(i-1, N) == 0;
741     ntest = M*N - (N - 1) <= i && i <= N*M ;
742     stest = 1 <= i && i <= N ;
743
744     % Northeastern corner
745     if etest == true && ntest == true
746

```

```

747     beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
748             + A_y*v_star(i-N));
749
750     % At eastern boundary (x = L)
751     eP_coeff = rho*A_x^2/au(i);
752
753     % At northern boundary
754     nP_coeff = 0 ;
755
756     W_coeff = -rho*A_x^2/au(i-1);
757     wP_coeff = -W_coeff;
758     T(i, i-1) = W_coeff;
759
760     S_coeff = -rho*A_y^2/av(i-N);
761     sP_coeff = -S_coeff;
762     T(i, i-N) = S_coeff;
763
764     % Southeastern corner
765     elseif etest == true && stest == true
766
767         beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
768                 -A_y*v_star(i));
769
770         % At eastern boundary (x = L)
771         eP_coeff = rho*A_x^2/au(i);
772
773         % At southern boundary (y = 0)
774         sP_coeff = 0;
775
776         W_coeff = -rho*A_x^2/au(i-1);
777         wP_coeff = -W_coeff;
778         T(i, i-1) = W_coeff;
779
780         N_coeff = -rho*A_y^2/av(i);
781         nP_coeff = -N_coeff;
782         T(i, i+N) = N_coeff;
783
784     % Northwestern corner
785     elseif wtest == true && ntest == true
786
787         beta(i) = rho*(-A_x*u_star(i) +A_x*u_in ...
788                 + A_y*v_star(i-N));
789
790         % At western boundary (x = 0)
791         wP_coeff = 0;
792
793         % At northern boundary
794         nP_coeff = 0 ;
795
796         E_coeff = -rho*A_x^2/au(i);
797         eP_coeff = -E_coeff ;
798         T(i, i+1) = E_coeff;
799
800         S_coeff = -rho*A_y^2/av(i-N);
801         sP_coeff = -S_coeff;
802         T(i, i-N) = S_coeff;
803
804     % Southwestern corner
805     elseif wtest == true && stest == true
806         beta(i) = rho*(-A_x*u_star(i) +A_x*u_in ...
807                 -A_y*v_star(i));
808
809         % At western boundary (x = 0)
810         wP_coeff = 0;
811
812         % At southern boundary (y = 0)
813         sP_coeff = 0;
814
815         E_coeff = -rho*A_x^2/au(i);
816         eP_coeff = -E_coeff ;
817         T(i, i+1) = E_coeff;
818
819         N_coeff = -rho*A_y^2/av(i);
820         nP_coeff = -N_coeff;
821         T(i, i+N) = N_coeff;
822

```

```

823     % At eastern boundary (x = L)
824     elseif etest == true && ntest == false && stest == false
825
826         beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)...
827             -A_y*v_star(i) + A_y*v_star(i-N));
828
829     % At eastern boundary (x = L)
830     eP_coeff = rho*A_x^2/au(i);
831
832     W_coeff = -rho*A_x^2/au(i-1);
833     wP_coeff = -W_coeff;
834     T(i, i-1) = W_coeff;
835
836     N_coeff = -rho*A_y^2/av(i);
837     nP_coeff = -N_coeff;
838     T(i, i+N) = N_coeff;
839
840     S_coeff = -rho*A_y^2/av(i-N);
841     sP_coeff = -S_coeff;
842     T(i, i-N) = S_coeff;
843
844     % At western boundary (x = 0)
845     elseif wtest == true && ntest == false && stest == false
846
847         beta(i) = rho*(-A_x*u_star(i) + A_x*u_in ...
848             -A_y*v_star(i) + A_y*v_star(i-N));
849
850     % At western boundary (x = 0)
851     wP_coeff = 0;
852
853     E_coeff = -rho*A_x^2/au(i);
854     eP_coeff = -E_coeff ;
855     T(i, i+1) = E_coeff;
856
857     N_coeff = -rho*A_y^2/av(i);
858     nP_coeff = -N_coeff;
859     T(i, i+N) = N_coeff;
860
861     S_coeff = -rho*A_y^2/av(i-N);
862     sP_coeff = -S_coeff;
863     T(i, i-N) = S_coeff;
864
865     % At northern boundary (y = h)
866     elseif ntest == true && etest == false && wtest == false
867
868         beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)...
869             + A_y*v_star(i-N));
870
871     % At northern boundary
872     nP_coeff = 0 ;
873
874     E_coeff = -rho*A_x^2/au(i);
875     eP_coeff = -E_coeff ;
876     T(i, i+1) = E_coeff;
877
878     W_coeff = -rho*A_x^2/au(i-1);
879     wP_coeff = -W_coeff;
880     T(i, i-1) = W_coeff;
881
882     S_coeff = -rho*A_y^2/av(i-N);
883     sP_coeff = -S_coeff;
884     T(i, i-N) = S_coeff;
885
886     % At southern boundary (y = 0)
887     elseif stest == true && etest == false && wtest == false
888
889         beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)...
890             -A_y*v_star(i));
891
892     % At southern boundary (y = 0)
893     sP_coeff = 0;
894
895     E_coeff = -rho*A_x^2/au(i);
896     eP_coeff = -E_coeff ;
897     T(i, i+1) = E_coeff;
898

```

```

899     W_coeff = -rho*A_x^2/au(i-1);
900     wP_coeff = -W_coeff;
901     T(i, i-1) = W_coeff;
902
903     N_coeff = -rho*A_y^2/av(i);
904     nP_coeff = -N_coeff;
905     T(i, i+N) = N_coeff;
906
907     %Not at any boundary
908     else
909
910         beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1) ...
911             -A_y*v_star(i) + A_y*v_star(i-N));
912
913         E_coeff = -rho*A_x^2/au(i);
914         eP_coeff = -E_coeff ;
915         T(i, i+1) = E_coeff;
916
917         W_coeff = -rho*A_x^2/au(i-1);
918         wP_coeff = -W_coeff;
919         T(i, i-1) = W_coeff;
920
921         N_coeff = -rho*A_y^2/av(i);
922         nP_coeff = -N_coeff;
923         T(i, i+N) = N_coeff;
924
925         S_coeff = -rho*A_y^2/av(i-N);
926         sP_coeff = -S_coeff;
927         T(i, i-N) = S_coeff;
928
929     end % if
930
931     % Filling in the rest of the matrix, adding all point coefficients
932     T(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
933
934     etest = false;
935     wtest = false;
936     ntest = false;
937     stest = false;
938 end % for
939 p_corr = T\beta';
940
941 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
942 %% Velocity correction
943
944 for j = 1:length(p_corr)
945     if mod(j, N) == 0 % eastern boundary
946         u_corr(j) = - A_x/au(j)*(-p_corr(j));
947         % pressure correction is zero for known outlet pressure
948     else
949         u_corr(j) = - A_x/au(j)*(p_corr(j+1)-p_corr(j));
950     end % if
951 end %for
952
953 for k = 1:length(p_corr)-N
954     v_corr(k) = - A_y/av(k)*(p_corr(k+N)-p_corr(k));
955 end %for
956
957 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
958 %% Under-relaxation
959
960 p_new = p_circ + alpha_p* p_corr;
961 u_new = alpha_u*(u_star' + u_corr) + (1-alpha_u)*u_circ;
962 v_new = alpha_v*(v_star' + v_corr) + (1-alpha_v)*v_circ;
963
964 if solvvel == false
965     v_new = zeros(1,length(v_new));
966     annoSolvvel = '$v$-velocity: Not solved';
967 else
968     annoSolvvel = '$v$-velocity: Solved';
969 end %if
970
971 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
972 %% Check convergence
973 % Make sure there are no mistakes in the matrix operations above
974 if ~isvector(u_new) || ~isvector(p_new) || ~isvector(v_new)

```

```

975     fprintf('u_new - %dx%d\n', size(u_new,1), size(u_new,2))
976     fprintf('v_new - %dx%d\n', size(v_new,1), size(v_new,2))
977     fprintf('p_new - %dx%d\n', size(p_new,1), size(p_new,2))
978     error('Matrix addition gone wrong')
979 end
980
981 if isnan(rcond(U)) || isnan(rcond(V)) || isnan(rcond(T))
982 %     clc % Remove if warnings are desired
983     fprintf('Stopped due to singularity in matrix\n')
984     fprintf('RCOND u-velocity: %e \nRCOND v-velocity: %e \n',...
985           rcond(U), rcond(V))
986     fprintf('RCOND pressure: %e\n', rcond(T))
987     fprintf('Problem occured after %d iterations\n', it)
988     return
989 end %if
990
991 c1 = 1/u_in*sqrt((U*u_star-bu')*(U*u_star-bu')); % residuals
992 c2 = 1/u_in*sqrt((V*v_star-bv')*(V*v_star-bv')); % residuals
993 c3 = abs(sum(beta)); % continuity fulfilled
994 c4 = 1/u_in*max(abs(u_circ - u_star')); % change from last iteration
995 c5 = 1/u_in*max(abs(v_circ - v_star')); % change from last iteration
996
997
998 c1_lim = 10^-8; % Limits
999 c2_lim = 10^-8;
1000 c3_lim = 10^-10;
1001 c4_lim = 10^-8;
1002 c5_lim = 10^-8;
1003
1004
1005 if solvvel == false % Overwrite if v-velocity is not solved for
1006     c2 = 0;
1007     c5 = 0;
1008 end %if
1009
1010 c1_diff = c1-c1_lim; % How far away from convergence
1011 c2_diff = c2-c2_lim;
1012 c3_diff = c3-c3_lim;
1013 c4_diff = c4-c4_lim;
1014 c5_diff = c5-c5_lim;
1015
1016 if (c1 < c1_lim) && (c2 < c2_lim) && (c3 < c3_lim) && (c4 < c4_lim)...
1017     && (c5 < c5_lim) || (it == maxits)
1018     conv = 1; % Converged
1019     if (it == maxits)
1020         fprintf('Stopped at max iterations (%d)\n',it);
1021     else
1022         fprintf('Solution converged after %d iterations\n',it);
1023     end %if
1024
1025     fprintf('c1\tMomentum residual u\t\t%.2e\tLimit: %.2e\n',...
1026           c1,c1_lim);
1027     fprintf('c2\tMomentum residual v\t\t%.2e\tLimit: %.2e\n',...
1028           c2,c2_lim);
1029     fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',...
1030           c3,c3_lim);
1031     fprintf('c4\tDiff. last iteration u\t%.2e\tLimit: %.2e\n',...
1032           c4,c4_lim);
1033     fprintf('c5\tDiff. last iteration v\t%.2e\tLimit: %.2e\n',...
1034           c5,c5_lim);
1035
1036     if max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c1_diff
1037         fprintf('Limiting criteria is c1\tMomentum residual u\n')
1038     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c2_diff
1039         fprintf('Limiting criteria is c2\tMomentum residual v\n')
1040     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c3_diff
1041         fprintf('Limiting criteria is c3\tPressure correction\n')
1042     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c4_diff
1043         fprintf('Limiting criteria is c4\tDiff. last iteration u\n')
1044     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c5_diff
1045         fprintf('Limiting criteria is c5\tDiff. last iteration v\n')
1046     end %if
1047
1048 else
1049
1050     u_circ = u_new ; % Not converged, updated variables

```

```

1051         v_circ = v_new ;                               % Not converged, updated variables
1052         p_circ = p_new ;                               % Not converged, updated variables
1053
1054     end % if
1055
1056     if runiterationwise == 1 || conv == 1
1057         plot_2D
1058         if conv == 0 % if not converged
1059             pause
1060             close all
1061         end %if
1062     end %if
1063
1064     it = it + 1;                                       % Update number of iterations
1065
1066     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1067 end %while
1068 toc

```

E.4.1.2 plot_2D.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                               Plotting of the two dimensional fluid flow                               %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  %% Velocities and pressure back to matrices
6
7  u_new_plot = zeros(M+2,N+1);
8
9  u_new_plot(:,1) = u_in_true;
10 u_star_plot(:,1) = u_in_true;
11
12 u_new_plot(1,1) = Inf;                               % The walls at the inlet are blocked out
13 u_new_plot(end,1) = Inf;
14
15 for j = 1:M
16     for i = 1:N
17         u_new_plot(j+1,i+1) = u_new((j-1)*N + i)*u_in_true;
18     end % for
19 end % for
20
21
22 v_new_plot = zeros(M+1,N+1);
23
24 for j = 1:m                                         % The rest of the points are zero
25     for i = 1:N
26         v_new_plot(j+1,i+1) = v_new((j-1)*N + i)*u_in_true;
27     end % for
28 end % for
29
30
31 p_plot = zeros(M,N+1);
32 p_corrplot = zeros(M,N+1);
33
34 p_plot(:,N+1) = p_atm;
35
36 for j = 1:M                                         % The rest of the points are zero
37     for i = 1:N
38         p_plot(j,i) = p_new((j-1)*N + i)*rho_true*u_in_true + p_atm;
39     %
40         p_corrplot(j,i) = p_corr((j-1)*N + i)*rho_true*u_in_true;
41     end % for
42 end % for
43
44 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
45 %% Plot
46 az_outlet = 45;                                     % Azimuth angle for setting viewpoint in figures
47 el_outlet = 30;                                     % Elevation height for setting viewpoint in figures
48
49 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50 f1 = figure;
51 f = surf(xu_plot,yu_plot,u_new_plot);               % surf(x,y,z)
52 s = sprintf('Plot of $u_{new}$ after %d iterations', it );
53 % f = title(s);
54 % set(f, 'interpreter', 'latex', 'fontsize', 16)
55 set(gca, 'TickLabelInterpreter', 'latex')

```

```

56 xlabel('$x$-direction [m]', 'interpreter', 'latex')
57 ylabel('$y$-direction [m]', 'interpreter', 'latex')
58 zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
59 ztickformat('%.2f')
60 saveas(gcf, 'unew2D.png')
61
62 f1_outlet = figure;
63 f = surf(xu_plot, yu_plot, u_new_plot);           % surf(x,y,z)
64 view(az_outlet, el_outlet)
65 s = sprintf('Plot of $u_{new}$ after %d iterations', it );
66 % f = title(s);
67 % set(f, 'interpreter', 'latex', 'fontsize', 16)
68 set(gca, 'TickLabelInterpreter', 'latex')
69 xlabel('$x$-direction [m]', 'interpreter', 'latex')
70 ylabel('$y$-direction [m]', 'interpreter', 'latex')
71 zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
72 ztickformat('%.2f')
73 saveas(gcf, 'unewoutlet2D.png')
74
75 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
76 f2 = figure;
77 f = surf(xv_plot, yv_plot, v_new_plot);           % surf(x,y,z)
78 s = sprintf('Plot of $v_{new}$ after %d iterations', it );
79 % f = title(s);
80 % set(f, 'interpreter', 'latex', 'fontsize', 16)
81 set(gca, 'TickLabelInterpreter', 'latex')
82 xlabel('$x$-direction [m]', 'interpreter', 'latex')
83 ylabel('$y$-direction [m]', 'interpreter', 'latex')
84 zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
85 ztickformat('%.2f')
86 saveas(gcf, 'vnew2D.png')
87
88 f2_outlet = figure;
89 f = surf(xv_plot, yv_plot, v_new_plot);           % surf(x,y,z)
90 view(az_outlet, el_outlet)
91 s = sprintf('Plot of $v_{new}$ after %d iterations', it );
92 % f = title(s);
93 % set(f, 'interpreter', 'latex', 'fontsize', 16)
94 set(gca, 'TickLabelInterpreter', 'latex')
95 xlabel('$x$-direction [m]', 'interpreter', 'latex')
96 ylabel('$y$-direction [m]', 'interpreter', 'latex')
97 zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
98 ztickformat('%.2f')
99 saveas(gcf, 'vnewoutlet2D.png')
100
101 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
102 f3 = figure;
103 surf(xp_plot, yp_plot, p_corrplot);               % surf(x,y,z)
104 s = sprintf('Plot of $p^{corr}$ after %d iterations', it );
105 % f = title(s);
106 % set(f, 'interpreter', 'latex', 'fontsize', 16)
107 set(gca, 'TickLabelInterpreter', 'latex')
108 xlabel('$x$-direction [m]', 'interpreter', 'latex')
109 ylabel('$y$-direction [m]', 'interpreter', 'latex')
110 zlabel('Pressure correction $p^{corr}$, [Pa]', 'interpreter', 'latex')
111 ztickformat('%.2f')
112 % zlim([-0.1 0.1])
113 saveas(gcf, 'pcorr2D.png')
114
115 f3_outlet = figure;
116 surf(xp_plot, yp_plot, p_corrplot);               % surf(x,y,z)
117 view(az_outlet, el_outlet)
118 s = sprintf('Plot of $p^{corr}$ after %d iterations', it );
119 % f = title(s);
120 % set(f, 'interpreter', 'latex', 'fontsize', 16)
121 set(gca, 'TickLabelInterpreter', 'latex')
122 xlabel('$x$-direction [m]', 'interpreter', 'latex')
123 ylabel('$y$-direction [m]', 'interpreter', 'latex')
124 zlabel('Pressure correction $p^{corr}$, [Pa]', 'interpreter', 'latex')
125 % zlim([-0.1 0.1])
126 ztickformat('%.2f')
127 saveas(gcf, 'pcorroutlet2D.png')
128
129 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
130 f4 = figure;
131 f = surf(xp_plot, yp_plot, p_plot);               % surf(x,y,z)

```

```

132 s = sprintf('Plot of $p_{new}$ after %d iterations', it );
133 % f = title(s);
134 % set(f, 'interpreter', 'latex', 'fontsize', 16)
135 set(gca, 'TickLabelInterpreter', 'latex')
136 xlabel('$x$-direction [m]', 'interpreter', 'latex')
137 ylabel('$y$-direction [m]', 'interpreter', 'latex')
138 zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
139 ztickformat('%.7f')
140 saveas(gcf, 'pnew2D.png')
141
142 f4_outlet = figure;
143 f = surf(xp_plot, yp_plot, p_plot); % surf(x,y,z)
144 view(az_outlet, el_outlet)
145 s = sprintf('Plot of $p_{new}$ after %d iterations', it );
146 % f = title(s);
147 % set(f, 'interpreter', 'latex', 'fontsize', 16)
148 set(gca, 'TickLabelInterpreter', 'latex')
149 xlabel('$x$-direction [m]', 'interpreter', 'latex')
150 ylabel('$y$-direction [m]', 'interpreter', 'latex')
151 zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
152 ztickformat('%.7f')
153 saveas(gcf, 'pnewoutlet2D.png')
154
155 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
156 %% Plotting the continuity
157
158 for i = 1:M*N % Global indexing system
159
160     etest = mod(i, N) == 0;
161     wtest = mod(i-1, N) == 0;
162     ntest = M*N - (N - 1) <= i && i <= N*M ;
163     stest = 1 <= i && i <= N ;
164
165     % Northeastern corner
166     if etest == true && ntest == true
167         beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1) ...
168             + A_y*v_star(i-N));
169         cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)) ;
170         cont_y(i) = rho*( A_y*v_star(i-N)) ;
171
172     % Southeastern corner
173     elseif etest == true && stest == true
174         beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1) ...
175             - A_y*v_star(i));
176         cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)) ;
177         cont_y(i) = rho*(-A_y*v_star(i)) ;
178
179     % Northwestern corner
180     elseif wtest == true && ntest == true
181         beta(i) = rho*(-A_x*u_star(i) + A_x*u_in ...
182             + A_y*v_star(i-N));
183         cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_in);
184         cont_y(i) = rho*(A_y*v_star(i-N));
185
186     % Southwestern corner
187     elseif wtest == true && stest == true
188         beta(i) = rho*(-A_x*u_star(i) + A_x*u_in ...
189             - A_y*v_star(i));
190         cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_in) ;
191         cont_y(i) = rho*(-A_y*v_star(i)) ;
192
193     % At eastern boundary (x = L)
194     elseif etest == true && ntest == false && stest == false
195         beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1) ...
196             - A_y*v_star(i) + A_y*v_star(i-N));
197         cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)) ;
198         cont_y(i) = rho*(-A_y*v_star(i) + A_y*v_star(i-N)) ;
199
200     % At western boundary (x = 0)
201     elseif wtest == true && ntest == false && stest == false
202         beta(i) = rho*(-A_x*u_star(i) + A_x*u_in ...
203             - A_y*v_star(i) + A_y*v_star(i-N));
204         cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_in) ;
205         cont_y(i) = rho*(-A_y*v_star(i) + A_y*v_star(i-N)) ;
206
207     % At northern boundary (y = h)

```



```

208         elseif ntest == true && etest == false && wtest == false
209             beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)...
210                 + A_y*v_star(i-N));
211             cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)) ;
212             cont_y(i) = rho*(A_y*v_star(i-N)) ;
213
214             % At southern boundary (y = 0)
215             elseif stest == true && etest == false && wtest == false
216                 beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)...
217                     -A_y*v_star(i));
218                 cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1));
219                 cont_y(i) = rho*(-A_y*v_star(i));
220
221             %Not at any boundary
222             else
223                 beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1) ...
224                     -A_y*v_star(i) + A_y*v_star(i-N));
225                 cont_x(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1)) ;
226                 cont_y(i) = rho*(-A_y*v_star(i) + A_y*v_star(i-N)) ;
227
228
229             end % if
230         end %for
231
232         beta_plot = zeros(M,N);
233         cont_x_plot = zeros(M,N);
234         cont_y_plot = zeros(M,N);
235
236         for j = 1:M % the rest of the points are zero
237             for i = 1:N
238                 cont_x_plot(j,i) = cont_x((j-1)*N + i);
239                 cont_y_plot(j,i) = cont_y((j-1)*N + i);
240                 beta_plot(j,i) = beta((j-1)*N + i);
241             end % for
242         end % for
243
244
245         if contplots
246             f5 = figure;
247             f = surf(xp_plot(1:end-1),yp_plot, cont_x_plot ); % surf(x,y,z)
248             s = sprintf(...
249                 'Plot of $x$-component of continuity after %d iterations', it );
250             f = title(s);
251             set(f, 'interpreter', 'latex', 'fontsize', 16)
252             set(gca, 'TickLabelInterpreter', 'latex')
253             xlabel('$x$-direction [m]', 'interpreter', 'latex')
254             ylabel('$y$-direction [m]', 'interpreter', 'latex')
255             zlabel('Mass flow rate [kg/s]', 'interpreter', 'latex')
256             ztickformat('% .2f')
257             saveas(gcf, 'cont_x.png')
258
259
260             f6 = figure;
261             f = surf(xp_plot(1:end-1),yp_plot, cont_y_plot ); % surf(x,y,z)
262             s = sprintf(...
263                 'Plot of $y$-component of continuity after %d iterations', it );
264             f = title(s);
265             set(f, 'interpreter', 'latex', 'fontsize', 16)
266             set(gca, 'TickLabelInterpreter', 'latex')
267             xlabel('$x$-direction [m]', 'interpreter', 'latex')
268             ylabel('$y$-direction [m]', 'interpreter', 'latex')
269             zlabel('Mass flow rate [kg/s]', 'interpreter', 'latex')
270             ztickformat('% .2f')
271             saveas(gcf, 'cont_y.png')
272
273             f7 = figure;
274             f = surf(xp_plot(1:end-1),yp_plot, beta_plot ); % surf(x,y,z)
275             s = sprintf(...
276                 'Plot of $\\beta$ (continuity) after %d iterations', it );
277             f = title(s);
278             set(f, 'interpreter', 'latex', 'fontsize', 16)
279             set(gca, 'TickLabelInterpreter', 'latex')
280             xlabel('$x$-direction [m]', 'interpreter', 'latex')
281             ylabel('$y$-direction [m]', 'interpreter', 'latex')
282             zlabel('Mass flow rate [kg/s]', 'interpreter', 'latex')
283             ztickformat('% .2f')

```

```

284     saveas(gcf, 'beta.png')
285 end

```

E.5 Backwards Facing Step Model

See figure in section 4.9 for a map of the working principle of the backwards facing step codes.

E.5.1 Constant Inlet Velocity

The code `channel_BFS.m` solves the two dimensional backwards facing step problem. The code `BFS_u_velocity.m` contains the calculations of the Momentum equation for the u -velocity component, `BFS_v_velocity.m` contains the calculations of the Momentum equation for the v -velocity component and `BFS_pressurecorrection.m` contains the calculations of the Momentum equation for the u -velocity component.

The code `plot.BFS.m` plots the surface plots for the velocities, pressure and pressure correction. The code `plotVelocityQuiver.m` plots the velocity quiver plots. The code `plotColoredQuiver.m` plots the velocity quiver plots with the contour plot for background colour. The code `plotVelocityCorrection.m` is used to plot the velocity corrections. The code `plotIntermediates.m` is used to plot the intermediate velocities u^* and v^* corrections. The code `plot_BFS_iterations.m` is used to plot the velocities, pressure and pressure correction for every specified iteration and saves them to a `.gif` file. The code `plotVelocityCorrection.m` is used to plot the initial, intermediate, corrected and new velocities and saving them to a `.gif` file.

The code `isWide.m` is used to check if a node point is in the narrow or wide section in the backwards facing step simulations. The code `getRowNumber.m` is used for the globally indexed vectors to obtain the row number in the corresponding matrix given the dimensions of the matrix. The code `getRowUnder.m` is used for the globally indexed vectors to obtain the row number directly below the node in the corresponding matrix given the dimensions of the matrix. The code `getRowOver.m` is used for the globally indexed vectors to obtain the row number directly above the node in the corresponding matrix given the dimensions of the matrix. The code `global2matrix.m` is used to convert the globally indexed vectors into their corresponding matrices given the dimensions of the matrix.

E.5.1.1 `channel_BFS.m`

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Two dimensional fluid flow over a backwards facing step, dimensionless %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  close all
5  clear
6  clc
7  tic
8  warning on
9
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 %% Solver specs
12 maxits = 25000; % Maximum number of iterations, stop if iterations exceed
13
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15 %% Options
16 plotiterationwise = false; % Plots the profiles after each iteration
17 solvvel = true; % Solve for v-velocity
18 plotCircVels = false; % Plot u_circ and v_circ
19 plotCorrVels = false; % Plot u_corr and v_corr

```



```

96 %% Dimensionless parameters
97 mu = 1; % Dimensionless viscosity
98 rho = 1; % Dimensionless density
99 del_x = del_x_true/D_hyd; % Dimensionless control volume width
100 del_y = del_y_true/D_hyd; % Dimensionless control volume height
101 A_x = A_x_true/D_hyd^2; % Dimensionless cross-sectional area in x-direction
102 A_y = A_y_true/D_hyd^2; % Dimensionless cross-sectional area in y-direction
103 D_x = 1/Re*mu/del_x; % Dimensionless diffusion conductance in x-direction
104 D_y = 1/Re*mu/del_y; % Dimensionless diffusion conductance in y-direction
105 u_in = 1; % Inlet u-velocity
106 v_in = 0; % Inlet v-velocity
107 u_guess = 1.0; % Initial guess for u-velocity
108 v_guess = 0.0; % Initial guess for v-velocity
109 p_guess = 0/(rho_true*u_in_true^2); % Initial guess for pressure
110
111 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
112 %% Initialisation of p
113 % Filling in initial pressure vector with the linear profile.
114 % This section is set up for if gravity is added, but could be more compact
115 % if the option to add gravity was not there.
116
117 p_circ_y_wide = linspace(p_guess, p_guess+rho*g_y*H_total,M_total);
118 p_circ_carthesian_wide = zeros(M_total,N_wide);
119 for j = 1:M_total
120     for i = 1:N_wide
121         p_circ_carthesian_wide(j,i) = p_circ_y_wide(j);
122     end %for
123 end %for
124
125 p_circ_y_narrow = p_circ_y_wide(M_wide+1:end);
126 p_circ_carthesian_narrow = zeros(M_narrow,N_narrow);
127 for j = 1:M_narrow
128     for i = 1:N_narrow
129         p_circ_carthesian_narrow(j,i) = p_circ_y_narrow(j);
130     end %for
131 end %for
132
133 filler = zeros(M_wide, N_narrow);
134 p_circ_carthesian = [[filler; p_circ_carthesian_narrow] ...
135     p_circ_carthesian_wide ];
136 p_circ_carthesian = flip(p_circ_carthesian,1);
137
138 p_circ = p_circ_carthesian(1,:); % Take the first vector
139
140 for i = 2:M_total
141     row = p_circ_carthesian(i);
142     if i <= M_narrow % Take whole row
143         p_circ = [p_circ, p_circ_carthesian(i,:)];
144     else % Take part of the row
145         p_circ = [p_circ, p_circ_carthesian(i,N_narrow+1:N_total)];
146     end %if
147 end %for
148
149 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
150 %% Initialisation of u and v
151
152 u_circ = ones(totalpoints,1)*u_guess; % Fill in guess in the initial vector
153 if ~onlyChannel % Only for the normal mode with the BFS enabled
154     for i = 1:totalpoints
155         if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
156             u_circ(i) = u_guess*(M_narrow/M_total);
157         end %if
158     end %for
159 end %if
160
161
162 v_circ = ones(totalpoints_v,1)*v_guess; % Fill in guess in the initial vec.
163 if ~onlyChannel % Only for the normal mode with the BFS enabled
164     for i = 1:totalpoints_v
165         if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
166             v_circ(i) = v_guess*(m_narrow/m_total);
167         end %if
168     end %for
169 end %if
170
171 if plotInitialProfiles == true % Plot the initial profiles if desired

```



```

248 % Make sure there are no mistakes in the matrix operations above
249 if ~isvector(u_new) || ~isvector(p_new) || ~isvector(p_new)
250     fprintf('u_new - %dx%d\n',size(u_new,1),size(u_new,2))
251     fprintf('v_new - %dx%d\n',size(v_new,1),size(v_new,2))
252     fprintf('p_new - %dx%d\n',size(p_new,1),size(p_new,2))
253     error('Matrix addition gone wrong')
254 end
255
256 if isnan(rcond(U)) || isnan(rcond(V)) || isnan(rcond(T))
257 %     clc % Remove if warnings are desired
258     fprintf('Stopped due to singularity in matrix\n')
259     fprintf('RCOND u-velocity: %e \nRCOND v-velocity: %e \n',...
260         rcond(U), rcond(V))
261     fprintf('RCOND pressure correction: %e\n',rcond(T))
262     fprintf('Problem occured after %d iterations\n', it)
263     toc
264     return
265 end %if
266
267 c1 = 1/u_in*sqrt((U*u_star-bu')*(U*u_star-bu')); % residuals
268 c2 = 1/u_in*sqrt((V*v_star-bv')*(V*v_star-bv')); % residuals
269 c3 = abs(sum(beta)); % continuity fulfilled
270 c4 = 1/u_in*max(abs(u_circ - u_star)) ; % change from last iteration
271 c5 = 1/u_in*max(abs(v_circ - v_star)) ; % change from last iteration
272
273 c1_lim = 10^-8; % Limits
274 c2_lim = 10^-8;
275 c3_lim = 10^-10;
276 c4_lim = 10^-8;
277 c5_lim = 10^-8;
278
279 if solvvel == false % Overwrite if v-velocity is not solved for
280     c2 = 0;
281     c5 = 0;
282 end %if
283
284 c1_diff = c1-c1_lim; % How far away from convergence
285 c2_diff = c2-c2_lim;
286 c3_diff = c3-c3_lim;
287 c4_diff = c4-c4_lim;
288 c5_diff = c5-c5_lim;
289
290
291 if (c1 < c1_lim) && (c2 < c2_lim) && (c3 < c3_lim) && (c4 < c4_lim) ...
292     && (c5 < c5_lim) || (it == maxits)
293     conv = 1; % Converged
294     if (it == maxits)
295         fprintf('Stopped at max iterations (%d)\n',it);
296     else
297         fprintf('Solution converged after %d iterations\n',it);
298     end %if
299
300     fprintf('c1\tMomentum residual u\t\t%.2e\tLimit: %.2e\n',...
301         c1,c1_lim);
302     fprintf('c2\tMomentum residual v\t\t%.2e\tLimit: %.2e\n',...
303         c2,c2_lim);
304     fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',...
305         c3,c3_lim);
306     fprintf('c4\tDiff. last iteration u\t%.2e\tLimit: %.2e\n',...
307         c4,c4_lim);
308     fprintf('c5\tDiff. last iteration v\t%.2e\tLimit: %.2e\n',...
309         c5,c5_lim);
310
311     if max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c1_diff
312         fprintf('Limiting criteria is c1\tMomentum residual u\n')
313     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c2_diff
314         fprintf('Limiting criteria is c2\tMomentum residual v\n')
315     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c3_diff
316         fprintf('Limiting criteria is c3\tPressure correction\n')
317     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c4_diff
318         fprintf('Limiting criteria is c4\tDiff. last iteration u\n')
319     elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c5_diff
320         fprintf('Limiting criteria is c5\tDiff. last iteration u\n')
321     end %if
322
323     showStep = false;

```

```

324     if plotCircVels == true
325         plotIntermediates
326     end
327     if plotCorrVels == true
328         plotVelocityCorrection
329     end
330     plot_BFS
331     if showVelociyQuiver == true
332         plotVelocityQuiver
333         plotColoredQuiver
334     end %if
335
336 else
337     if plotiterationwise == true
338         showStep = false;
339         if plotCircVels == true
340             plotIntermediates
341         end
342         if plotCorrVels == true
343             plotVelocityCorrection
344         end
345         plot_BFS
346         if showVelociyQuiver == true
347             plotVelocityQuiver
348             plotColoredQuiver
349         end %if
350         pause
351         close all
352     end %if
353     if printSetPlotIt && ismember(it,itSaves)
354         plot_BFS_iterations
355         if gifIntermediates == true
356             plotVelInts_BFS_iterations;
357         end %if
358     end
359
360     u_circ = u_new;           % Not converged, updated variables
361     v_circ = v_new;           % Not converged, updated variables
362     p_circ = p_new;           % Not converged, updated variables
363
364     it = it + 1; % Update number of iterations
365 end % if
366 end %while
367 toc

```

E.5.1.2 BFS_u_velocity.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %           u-velocity script for the BFS model           %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  U = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
6  bu = zeros(1, totalpoints);         % Initialisation of source term vector
7
8  F_xe = zeros(1, totalpoints);       % Initialisation of convective mass fluxes
9  F_xw = zeros(1, totalpoints);
10 F_xn = zeros(1, totalpoints);
11 F_xs = zeros(1, totalpoints);
12
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 % Generation of F_x, Convective mass fluxes
15
16
17 for i = 1:totalpoints
18
19     etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )... % below step
20     || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0 );
21     wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
22     ntest = totalpoints - N_total < i && i <= totalpoints ;
23     if ~onlyChannel % Normal mode
24         wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
25         stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
26         || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
27         scorner = i == N_wide*M_wide + N_narrow; % Only the corner value
28     else % No step mode
29         wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;

```

```

30     stest = i <= N_wide*M_wide + N_total; % Excluding the corner value
31     scorner = false; % Only the corner value
32 end %if
33
34
35 % Northeastern corner
36 if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
37     F_xe(i) = rho/2*(u_circ(i)+ u_circ(i-1));
38     F_xn(i) = 0;
39
40     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
41     F_xs(i) = rho/2*v_circ(i-N_total);
42
43 % Southeastern corner
44 elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
45     F_xe(i) = rho/2*(u_circ(i)+u_circ(i-1));
46     F_xs(i) = 0;
47
48     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
49     F_xn(i) = rho/2*v_circ(i);
50
51 % Northwestern corner
52 elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner
53     F_xw(i) = rho/2*(u_in+u_circ(i));
54     F_xn(i) = 0;
55
56     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
57     F_xs(i) = rho/2*(v_circ(i-N_total) + v_circ(i-N_total+1));
58
59 % Southwestern corner at inlet
60 elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner
61     F_xw(i) = rho/2*(u_in+u_circ(i));
62     F_xs(i) = 0;
63
64     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
65     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
66
67 % Southwestern corner at step
68 elseif ~etest && ~wtest && ~ntest && stest && wwall && ~scorner
69     F_xw(i) = rho/2*(0 + u_circ(i));
70     F_xs(i) = 0;
71
72     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
73     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
74
75 % At corner
76 elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner
77     F_xs(i) = rho/2*(0 + ...
78         v_circ(getRowUnder(i, N_wide, M_wide, N_total)+1));
79     F_xs(i)= 0;
80
81     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
82     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
83     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
84
85 % At eastern boundary (x = L)
86 elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner
87     F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));
88
89     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
90     F_xn(i) = rho/2*v_circ(i);
91     F_xs(i) = rho/2*v_circ(getRowUnder(i, N_wide, M_wide, N_total));
92
93 % At western boundary (x = 0)
94 elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner
95     F_xw(i) = rho/2*(u_in+u_circ(i));
96
97     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
98     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
99     F_xs(i) = rho/2*(...
100         v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
101         v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
102
103 % At western wall at step
104 elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner

```



```

105     F_xw(i) = rho/2*(0+u_circ(i));
106
107     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
108     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
109     F_xs(i) = rho/2*(...
110         v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
111         v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
112
113
114
115 % At northern boundary (y = h)
116 elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
117     F_xn(i) = 0;
118
119     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
120     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
121     F_xs(i) = rho/2*(...
122         v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
123         v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
124
125
126 % At southern boundary (y = 0)
127 elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
128     F_xs(i) = 0;
129
130     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
131     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
132     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
133
134 % Not at any boundary
135 else
136     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
137     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
138     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
139     F_xs(i) = rho/2*(...
140         v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
141         v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
142
143
144 end % if
145 etest = false;
146 wtest = false;
147 wwall = false;
148 ntest = false;
149 stest = false;
150 scorner = false;
151 end %for
152
153
154
155 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
156 %% u-velocity
157
158
159 for i = 1:totalpoints % Global indexing system
160
161     etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )... % below step
162         || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
163     wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
164     ntest = totalpoints - N_total < i && i <= totalpoints ;
165     if ~onlyChannel % Normal mode
166         wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
167         stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
168             || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
169         scorner = i == N_wide*M_wide + N_narrow; % Only the corner value
170     else % No step mode
171         wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
172         stest = i <= N_wide*M_wide + N_total; % Excluding the corner value
173         scorner = false; % Only the corner value
174     end %if
175
176
177 % Northeastern corner
178 if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
179
180     bu(i) = -(p_out(end)-p_circ(i))*A_x;

```

```

181
182 % At eastern boundary (x = L)
183 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
184 eP_coeff = F_xe(i)*A_x;
185
186 % At northern boundary
187 nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y) + 2*D_y*A_y;
188
189 W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
190 wP_coeff = -W_coeff - F_xw(i)*A_x;
191 U(i, i-1) = W_coeff;
192
193 S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
194 sP_coeff = -S_coeff - F_xs(i)*A_y;
195 U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
196
197
198 % Southeastern corner
199 elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
200
201 bu(i) = -(p_out(1)-p_circ(i))*A_x;
202
203 % At eastern boundary (x = L)
204 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
205 eP_coeff = F_xe(i)*A_x;
206
207 % At southern boundary (y = 0)
208 sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
209
210 W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
211 wP_coeff = -W_coeff - F_xw(i)*A_x;
212 U(i, i-1) = W_coeff;
213
214 N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
215 nP_coeff = -N_coeff + F_xn(i)*A_y;
216 U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
217
218
219 % Northwestern corner
220 elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner
221
222 bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
223         +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;
224
225 % At western boundary (x = 0)
226 wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
227
228 % At northern boundary
229 nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;
230
231 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
232 eP_coeff = -E_coeff + F_xe(i)*A_x;
233 U(i, i+1) = E_coeff;
234
235 S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
236 sP_coeff = -S_coeff - F_xs(i)*A_y;
237 U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
238
239
240 % Southwestern corner at inlet
241 elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner
242
243 bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
244         +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;
245
246 % At western boundary (x = 0)
247 wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
248
249 % At southern boundary (y = 0)
250 sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
251
252 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
253 eP_coeff = -E_coeff + F_xe(i)*A_x;
254 U(i, i+1) = E_coeff;
255
256 N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;

```

```

257     nP_coeff = -N_coeff + F_xn(i)*A_y;
258     U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
259
260
261     % Southwestern corner at step
262     elseif ~etest && ~wtest && ~ntest && stest && wwall && ~scorner
263
264         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
265             +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;
266
267         % At western boundary (x = 0)
268         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
269         wP_coeff = -W_coeff - F_xw(i)*A_x;
270
271         % At southern boundary (y = 0)
272         S_coeff = -max(F_xs(i)*A_y,0) - 2*D_y*A_y;
273         sP_coeff = -S_coeff - F_xs(i)*A_y;
274
275         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
276         eP_coeff = -E_coeff + F_xe(i)*A_x;
277         U(i, i+1) = E_coeff;
278
279         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
280         nP_coeff = -N_coeff + F_xn(i)*A_y;
281         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
282
283
284     % At corner
285     elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner
286
287         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
288
289         % At southern boundary (y = 0)
290         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
291         sP_coeff = -S_coeff - F_xs(i)*A_y;
292
293
294         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
295         eP_coeff = -E_coeff + F_xe(i)*A_x;
296         U(i, i+1) = E_coeff;
297
298         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
299         wP_coeff = -W_coeff - F_xw(i)*A_x;
300         U(i, i-1) = W_coeff;
301
302         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
303         nP_coeff = -N_coeff + F_xn(i)*A_y;
304         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
305
306
307     % At eastern boundary (x = L)
308     elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner
309
310         bu(i) = -(p_out(1)-p_circ(i))*A_x;
311
312         % At eastern boundary (x = L)
313         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
314         eP_coeff = F_xe(i)*A_x;
315
316         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
317         wP_coeff = -W_coeff - F_xw(i)*A_x;
318         U(i, i-1) = W_coeff;
319
320         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
321         nP_coeff = -N_coeff + F_xn(i)*A_y;
322         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
323
324         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
325         sP_coeff = -S_coeff - F_xs(i)*A_y;
326         U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
327
328
329     % At western boundary (x = 0)
330     elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner
331
332         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...

```

```

333         +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;
334
335     % At western boundary (x = 0)
336     wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
337
338     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
339     eP_coeff = -E_coeff + F_xe(i)*A_x;
340     U(i, i+1) = E_coeff;
341
342     N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
343     nP_coeff = -N_coeff + F_xn(i)*A_y;
344     U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
345
346     S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
347     sP_coeff = -S_coeff - F_xs(i)*A_y;
348     U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
349
350
351     % At western wall
352     elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner
353
354         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
355             +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;
356
357     % At western boundary (x = 0)
358     W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
359     wP_coeff = -W_coeff - F_xw(i)*A_x;
360
361     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
362     eP_coeff = -E_coeff + F_xe(i)*A_x;
363     U(i, i+1) = E_coeff;
364
365     N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
366     nP_coeff = -N_coeff + F_xn(i)*A_y;
367     U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
368
369     S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
370     sP_coeff = -S_coeff - F_xs(i)*A_y;
371     U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
372
373
374     % At northern boundary (y = h)
375     elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
376
377         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
378
379     % At northern boundary
380     nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;
381
382     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
383     eP_coeff = -E_coeff + F_xe(i)*A_x;
384     U(i, i+1) = E_coeff;
385
386     W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
387     wP_coeff = -W_coeff - F_xw(i)*A_x;
388     U(i, i-1) = W_coeff;
389
390     S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
391     sP_coeff = -S_coeff - F_xs(i)*A_y;
392     U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
393
394
395     % At southern boundary (y = 0)
396     elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
397
398         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
399
400     % At southern boundary (y = 0)
401     sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
402
403     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
404     eP_coeff = -E_coeff + F_xe(i)*A_x;
405     U(i, i+1) = E_coeff;
406
407     W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
408     wP_coeff = -W_coeff - F_xw(i)*A_x;

```

```

409     U(i, i-1) = W_coeff;
410
411     N_coeff = -max(0, -F_xn(i)*A_y) - D_y*A_y;
412     nP_coeff = -N_coeff + F_xn(i)*A_y;
413     U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
414
415
416     %Not at any boundary
417     else
418
419         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
420         E_coeff = -max(0, -F_xe(i)*A_x) - D_x*A_x;
421         eP_coeff = -E_coeff + F_xe(i)*A_x;
422         U(i, i+1) = E_coeff;
423
424         W_coeff = -max(F_xw(i)*A_x, 0) - D_x*A_x;
425         wP_coeff = -W_coeff - F_xw(i)*A_x;
426         U(i, i-1) = W_coeff;
427
428         N_coeff = -max(0, -F_xn(i)*A_y) - D_y*A_y;
429         nP_coeff = -N_coeff + F_xn(i)*A_y;
430         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
431
432         S_coeff = -max(F_xs(i)*A_y, 0) - D_y*A_y;
433         sP_coeff = -S_coeff - F_xs(i)*A_y;
434         U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
435
436     end % if
437
438     % Filling in the rest of the matrix, adding all point coefficients
439     U(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
440
441     % If the step is disabled the points below the step are blocked out
442     if onlyChannel && i <= N_wide*M_wide
443         U(i,i) = U(i,i) + 10e+30;
444     end %if
445
446     etest = false;
447     wtest = false;
448     ntest = false;
449     stest = false;
450     wwall = false;
451
452 end %for
453 u_star = U\bu'; % Matrix inversion

```

E.5.1.3 BFS_v_velocity.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                               v-velocity script for the BFS model          %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  V = zeros(totalpoints_v, totalpoints_v); % Initialisation of coeff. matrix
6  bv = zeros(1, totalpoints_v); % Initialisation of source term vector
7
8  F_ye = zeros(1, totalpoints_v); % Initialisation of convective mass fluxes
9  F_yw = zeros(1, totalpoints_v);
10 F_yn = zeros(1, totalpoints_v);
11 F_ys = zeros(1, totalpoints_v);
12
13
14
15
16 %% Generation of F_y, Convective mass fluxes
17
18 for i = 1:totalpoints_v % Global indexing system
19
20     % Eastern boundary requires no special treatment (x = L)
21     etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ... % below step
22         || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0 );
23     wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
24     ntest = totalpoints_v - N_total < i && i <= totalpoints_v ;
25     if ~onlyChannel % Normal mode
26         wwall = i <= N_wide*m_wide && mod(i-1, N_wide) == 0; %
27         stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
28             || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow);

```

```

29     wcorner = i == N_wide*(m_wide-1) + 1;           % Only the corner value
30     else                                           % No step mode
31         wwall = i <= N_wide*m_wide && mod(i-1, N_wide) == 0; %
32         stest = i <= N_wide*m_wide + N_total; % Excluding the corner value
33         wcorner = false;                          % Only the corner value
34     end %if
35
36
37
38     % Northwestern corner
39     if wtest && ntest && ~stest && ~wwall && ~wcorner
40         F_yw(i) = rho/2*(u_in+u_in);
41         F_yn(i) = rho/2*v_circ(i);
42
43         F_ye(i) = rho/2*(u_circ(i) + ...
44             u_circ(getRowOver(i, N_wide, M_wide, N_total)));
45         F_ys(i) = rho/2*(v_circ(i) + ...
46             v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
47
48     % Southwestern corner at inlet
49     elseif wtest && ~ntest && stest && ~wwall && ~wcorner
50         F_yw(i) = rho*u_in;
51         F_ys(i) = rho/2*v_circ(i);
52
53         F_ye(i) = rho/2*(u_circ(i) + ...
54             u_circ(getRowOver(i, N_wide, M_wide, N_total)));
55         F_yn(i) = rho/2*(v_circ(i) + ...
56             v_circ(getRowOver(i, N_wide, M_wide, N_total)));
57
58     % Southwestern corner at step
59     elseif ~wtest && ~ntest && stest && wwall && ~wcorner
60         F_yw(i) = rho*0;
61         F_ys(i) = rho/2*v_circ(i);
62
63         F_ye(i) = rho/2*(u_circ(i) + ...
64             u_circ(getRowOver(i, N_wide, M_wide, N_total)));
65         F_yn(i) = rho/2*(v_circ(i) + ...
66             v_circ(getRowOver(i, N_wide, M_wide, N_total)));
67
68     % At western boundary (x = 0)
69     elseif wtest && ~ntest && ~stest && ~wwall && ~wcorner
70         F_yw(i) = rho*u_in;
71
72         F_ye(i) = rho/2*(u_circ(i) + ...
73             u_circ(getRowOver(i, N_wide, M_wide, N_total)));
74         F_yn(i) = rho/2*(v_circ(i) + ...
75             v_circ(getRowOver(i, N_wide, M_wide, N_total)));
76         F_ys(i) = rho/2*(v_circ(i) + ...
77             v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
78
79     % At western wall
80     elseif ~wtest && ~ntest && ~stest && wwall && ~wcorner
81         F_yw(i) = rho*0;
82
83         F_ye(i) = rho/2*(u_circ(i) + ...
84             u_circ(getRowOver(i, N_wide, M_wide, N_total)));
85         F_yn(i) = rho/2*(v_circ(i) + ...
86             v_circ(getRowOver(i, N_wide, M_wide, N_total)));
87         F_ys(i) = rho/2*(v_circ(i) + ...
88             v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
89
90     % At corner, right point from the corner
91     elseif ~wtest && ~ntest && ~stest && wwall && wcorner
92         F_yw(i) = 0;
93
94         F_ye(i) = rho/2*(u_circ(i) + ...
95             u_circ(getRowOver(i, N_wide, M_wide, N_total)));
96         F_yn(i) = rho/2*(v_circ(i) + ...
97             v_circ(getRowOver(i, N_wide, M_wide, N_total)));
98         F_ys(i) = rho/2*(v_circ(i) + ...
99             v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
100
101
102
103
104

```

```

105 % At northern boundary (y = h)
106 elseif ~wtest && ntest && ~stest && ~wwall && ~wcorner
107     F_yn(i) = rho/2*v_circ(i);
108
109     F_ye(i) = rho/2*(u_circ(i) + ...
110         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
111     F_yw(i) = rho/2*(u_circ(i-1) + ...
112         u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
113     F_ys(i) = rho/2*(v_circ(i) + ...
114         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
115
116
117 % At southern boundary (y = 0)
118 elseif ~wtest && ~ntest && stest && ~wwall && ~wcorner
119     F_ys(i) = rho/2*v_circ(i);
120
121     F_ye(i) = rho/2*(u_circ(i) + ...
122         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
123     F_yw(i) = rho/2*(u_circ(i-1) + ...
124         u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
125     F_yn(i) = rho/2*(v_circ(i) + ...
126         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
127
128
129 %Not at any boundary, including eastern boundary
130 else
131     F_ye(i) = rho/2*(u_circ(i) + ...
132         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
133     F_yw(i) = rho/2*(u_circ(i-1) + ...
134         u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
135
136     F_yn(i) = rho/2*(v_circ(i) + ...
137         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
138     F_ys(i) = rho/2*(v_circ(i) + ...
139         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
140
141 end % if
142 etest = false;
143 wtest = false;
144 ntest = false;
145 stest = false;
146 wwall = false;
147 wcorner = false;
148
149 end % for
150
151 %% v-velocity
152
153
154 for i = 1:totalpoints_v % Global indexing system
155
156     etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ... % below step
157         || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0);
158     wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
159     ntest = totalpoints_v - N_total < i && i <= totalpoints_v ;
160     if ~onlyChannel % Normal mode
161         wwall = i <= N_wide*m_wide && mod(i-1, N_wide) == 0; %
162         stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
163             || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow) ;
164         wcorner = i == N_wide*(m_wide-1) + 1; % Only the corner value
165     else % No step mode
166         wwall = i <= N_wide*m_wide && mod(i-1, N_wide) == 0; %
167         stest = i <= N_wide*m_wide + N_total; % Excluding the corner value
168         wcorner = false; % Only the corner value
169     end %if
170
171
172
173 % Northeastern corner
174 if etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner
175
176     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
177         -p_circ(i))*A_y + rho*g_y*del_y*A_y;
178
179     % At eastern boundary (x = L)
180     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;

```

```

181     eP_coeff = F_yc(i)*A_x;
182
183     % At northern boundary
184     nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y;
185
186     W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
187     wP_coeff = -W_coeff - F_yw(i)*A_x;
188     V(i, i-1) = W_coeff;
189
190     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
191     sP_coeff = -S_coeff - F_ys(i)*A_y;
192     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
193
194     % Southeastern corner
195     elseif etest && ~wttest && ~nttest && stest && ~wwall && ~wcorner
196         bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
197             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
198
199         % At eastern boundary (x = L)
200         E_coeff = -max(0,-F_yc(i)*A_x) - D_x*A_x;
201         eP_coeff = F_yc(i)*A_x;
202
203         % At southern boundary (y = 0),
204         sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
205
206         W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
207         wP_coeff = -W_coeff - F_yw(i)*A_x;
208         V(i, i-1) = W_coeff;
209
210         N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
211         nP_coeff = -N_coeff + F_yn(i)*A_y;
212         V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
213
214
215     % Northwestern corner
216     elseif ~etest && wttest && nttest && ~stest && ~wwall && ~wcorner
217         bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
218             p_circ(i))*A_y + rho*g_y*del_y*A_y;
219
220         % At western boundary (x = 0)
221         wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
222
223         % At northern boundary
224         nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;
225
226         E_coeff = -max(0,-F_yc(i)*A_x) - D_x*A_x;
227         eP_coeff = -E_coeff + F_yc(i)*A_x;
228         V(i, i+1) = E_coeff;
229
230         S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
231         sP_coeff = -S_coeff - F_ys(i)*A_y;
232         V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
233
234     % Southwestern corner at inlet
235     elseif ~etest && wttest && ~nttest && stest && ~wwall && ~wcorner
236
237         bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
238             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
239
240         % At western boundary (x = 0)
241         wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
242
243         % At southern boundary (y = 0),
244         sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
245
246         E_coeff = -max(0,-F_yc(i)*A_x) - D_x*A_x;
247         eP_coeff = -E_coeff + F_yc(i)*A_x;
248         V(i, i+1) = E_coeff;
249
250         N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
251         nP_coeff = -N_coeff + F_yn(i)*A_y;
252         V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
253
254     % Southwestern corner at step
255     elseif ~etest && ~wttest && ~nttest && stest && wwall && ~wcorner

```



```

257     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
258             -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
259             0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);
260
261     % At western boundary (x = 0)
262     W_coeff = -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
263     wP_coeff = -W_coeff - F_yw(i)*A_x;
264
265     % At southern boundary (y = 0),
266     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
267     sP_coeff = -S_coeff - F_ys(i)*A_y;
268
269     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
270     eP_coeff = -E_coeff + F_ye(i)*A_x;
271     V(i, i+1) = E_coeff;
272
273     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
274     nP_coeff = -N_coeff + F_yn(i)*A_y;
275     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
276
277     % At eastern boundary (x = L)
278     elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~wcorner
279
280     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
281             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
282
283     % At eastern boundary (x = L)
284     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
285     eP_coeff = F_ye(i)*A_x;
286
287     W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
288     wP_coeff = -W_coeff - F_yw(i)*A_x;
289     V(i, i-1) = W_coeff;
290
291     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
292     nP_coeff = -N_coeff + F_yn(i)*A_y;
293     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
294
295     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
296     sP_coeff = -S_coeff - F_ys(i)*A_y;
297     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
298
299     % At western boundary (x = 0)
300     elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~wcorner
301
302     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
303             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
304
305     % At western boundary (x = 0)
306     wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
307
308     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
309     eP_coeff = -E_coeff + F_ye(i)*A_x;
310     V(i, i+1) = E_coeff;
311
312     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
313     nP_coeff = -N_coeff + F_yn(i)*A_y;
314     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
315
316     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
317     sP_coeff = -S_coeff - F_ys(i)*A_y;
318     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
319
320     % At west wall (x = 0) [EXCLUDED CORNER]
321     elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~wcorner
322
323     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
324             -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
325             0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);
326
327     % At western boundary (x = 0)
328     W_coeff = -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
329     wP_coeff = -W_coeff - F_yw(i)*A_x;
330
331     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
332     eP_coeff = -E_coeff + F_ye(i)*A_x;

```

```

333     V(i, i+1) = E_coeff;
334
335     N_coeff = -max(0, -F_yn(i)*A_y) - D_y*A_y;
336     nP_coeff = -N_coeff + F_yn(i)*A_y;
337     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
338
339     S_coeff = -max(F_ys(i)*A_y, 0) - D_y*A_y;
340     sP_coeff = -S_coeff - F_ys(i)*A_y;
341     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
342
343     % At corner
344     elseif ~etest && ~wtest && ~ntest && ~stest && wwall && wcorner
345
346         bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
347             -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
348             0*(-max(F_yw(i)*A_x, 0) - D_x*A_x);
349
350         % At western boundary (x = 0)
351         W_coeff = -max(F_yw(i)*A_x, 0) - D_x*A_x;
352         wP_coeff = -W_coeff - F_yw(i)*A_x;
353
354         E_coeff = -max(0, -F_ye(i)*A_x) - D_x*A_x;
355         eP_coeff = -E_coeff + F_ye(i)*A_x;
356         V(i, i+1) = E_coeff;
357
358         N_coeff = -max(0, -F_yn(i)*A_y) - D_y*A_y;
359         nP_coeff = -N_coeff + F_yn(i)*A_y;
360         V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
361
362         S_coeff = -max(F_ys(i)*A_y, 0) - D_y*A_y;
363         sP_coeff = -S_coeff - F_ys(i)*A_y;
364         V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
365
366
367     % At northern boundary (y = h)
368     elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner
369
370         bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
371             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
372
373         % At northern boundary
374         nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;
375
376         E_coeff = -max(0, -F_ye(i)*A_x) - D_x*A_x;
377         eP_coeff = -E_coeff + F_ye(i)*A_x;
378         V(i, i+1) = E_coeff;
379
380         W_coeff = -max(F_yw(i)*A_x, 0) - D_x*A_x;
381         wP_coeff = -W_coeff - F_yw(i)*A_x;
382         V(i, i-1) = W_coeff;
383
384         S_coeff = -max(F_ys(i)*A_y, 0) - D_y*A_y;
385         sP_coeff = -S_coeff - F_ys(i)*A_y;
386         V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
387
388
389     % At southern boundary (y = 0)
390     elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~wcorner
391
392         bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
393             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
394
395         % At southern boundary (y = 0),
396         sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y, 0) + D_y*A_y;
397
398         E_coeff = -max(0, -F_ye(i)*A_x) - D_x*A_x;
399         eP_coeff = -E_coeff + F_ye(i)*A_x;
400         V(i, i+1) = E_coeff;
401
402         W_coeff = -max(F_yw(i)*A_x, 0) - D_x*A_x;
403         wP_coeff = -W_coeff - F_yw(i)*A_x;
404         V(i, i-1) = W_coeff;
405
406         N_coeff = -max(0, -F_yn(i)*A_y) - D_y*A_y;
407         nP_coeff = -N_coeff + F_yn(i)*A_y;
408         V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

```

```

409
410     %Not at any boundary
411     else
412
413         bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
414             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
415
416         E_coeff = -max(0,-F_yc(i)*A_x) - D_x*A_x;
417         eP_coeff = -E_coeff + F_yc(i)*A_x;
418         V(i, i+1) = E_coeff;
419
420         W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
421         wP_coeff = -W_coeff - F_yw(i)*A_x;
422         V(i, i-1) = W_coeff;
423
424         N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
425         nP_coeff = -N_coeff + F_yn(i)*A_y;
426         V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
427
428         S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
429         sP_coeff = -S_coeff - F_ys(i)*A_y;
430         V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
431
432     end % if
433
434     % Filling in the rest of the matrix, adding all point coefficients
435     V(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
436
437     % If the step is disabled the points below the step are blocked out
438     if onlyChannel && i <= N_wide*m_wide
439         V(i,i) = V(i,i) + 10e+30;
440     end %if
441
442     etest = false;
443     wtest = false;
444     ntest = false;
445     stest = false;
446     wwall = false;
447
448 end % for
449 v_star = V\bv'; % Matrix inversion

```

E.5.1.4 BFS_pressurecorrection.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %           Pressure correction script for the BFS model           %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  T = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
6  beta = zeros(1, totalpoints); % Initialisation of source term vector
7
8  au = diag(U); % a^center-coefficients from the momentum equations
9  av = diag(V);
10
11
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 %% Calculation
14 for i = 1:totalpoints % Global indexing system
15
16     etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 ) ... % below step
17         || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
18     ntest = totalpoints - N_total < i && i <= totalpoints ;
19     wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
20
21     if ~onlyChannel % Normal Mode
22         wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
23         stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
24             || (N_wide*M_wide < i && i <= N_wide*M_wide + N_narrow) ;
25     else % No step mode
26         wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
27         stest = i <= N_wide*M_wide + N_total; % Excluding the corner value
28     end
29
30
31 % Northeastern corner
32 if etest && ~wtest && ntest && ~stest && ~wwall

```

```

33
34     beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
35         + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
36
37     % At eastern boundary (x = L)
38     eP_coeff = rho*A_x^2/au(i);
39
40     % At northern boundary (y = h) (y = H)
41     nP_coeff = 0 ;
42
43     W_coeff = -rho*A_x^2/au(i-1);
44     wP_coeff = -W_coeff;
45     T(i, i-1) = W_coeff;
46
47     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
48     sP_coeff = -S_coeff;
49     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
50
51
52     % Southeastern corner
53     elseif etest && ~wtest && ~ntest && stest && ~wwall
54
55         beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
56             -A_y*v_star(i));
57
58         % At eastern boundary (x = L)
59         eP_coeff = rho*A_x^2/au(i);
60
61         % At southern boundary (y = 0)
62         sP_coeff = 0;
63
64         W_coeff = -rho*A_x^2/au(i-1);
65         wP_coeff = -W_coeff;
66         T(i, i-1) = W_coeff;
67
68         N_coeff = -rho*A_y^2/av(i);
69         nP_coeff = -N_coeff;
70         T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
71
72
73     % Northwestern corner
74     elseif ~etest && wtest && ntest && ~stest && ~wwall
75
76         beta(i) = rho*(-A_x*u_star(i) +A_x*u_in ...
77             + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
78
79         % At western boundary (x = 0)
80         wP_coeff = 0;
81
82         % At northern boundary (y = h) (y = H)
83         nP_coeff = 0 ;
84
85         E_coeff = -rho*A_x^2/au(i);
86         eP_coeff = -E_coeff ;
87         T(i, i+1) = E_coeff;
88
89         S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
90         sP_coeff = -S_coeff;
91         T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
92
93
94     % Southwestern corner at inlet
95     elseif ~etest && wtest && ~ntest && stest && ~wwall
96
97         beta(i) = rho*(-A_x*u_star(i) +A_x*u_in ...
98             -A_y*v_star(i));
99
100        % At western boundary (x = 0)
101        wP_coeff = 0;
102
103        % At southern boundary (y = 0)
104        sP_coeff = 0;
105
106        E_coeff = -rho*A_x^2/au(i);
107        eP_coeff = -E_coeff ;
108        T(i, i+1) = E_coeff;

```

```

109
110     N_coeff = -rho*A_y^2/av(i);
111     nP_coeff = -N_coeff;
112     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
113
114
115     % Southwestern corner at step
116     elseif ~etest && ~wtest && ~ntest && stest && wwall
117
118         beta(i) = rho*(-A_x*u_circ(i)...
119                 +A_x*0 -A_y*v_circ(i)); % wall/"inlet" velocity is zero
120
121     % At western boundary (x = 0)
122     wP_coeff = 0;
123
124     % At southern boundary (y = 0)
125     sP_coeff = 0;
126
127     E_coeff = -rho*A_x^2/au(i);
128     eP_coeff = -E_coeff ;
129     T(i, i+1) = E_coeff;
130
131     N_coeff = -rho*A_y^2/av(i);
132     nP_coeff = -N_coeff;
133     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
134
135
136     % At eastern boundary (x = L)
137     elseif etest && ~wtest && ~ntest && ~stest && ~wwall
138
139     beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
140                 -A_y*v_star(i) + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
141
142     % At eastern boundary (x = L)
143     eP_coeff = rho*A_x^2/au(i);
144
145
146     W_coeff = -rho*A_x^2/au(i-1);
147     wP_coeff = -W_coeff;
148     T(i, i-1) = W_coeff;
149
150     N_coeff = -rho*A_y^2/av(i);
151     nP_coeff = -N_coeff;
152     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
153
154     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
155     sP_coeff = -S_coeff;
156     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
157
158
159     % At western boundary at inlet (x = 0)
160     elseif ~etest && wtest && ~ntest && ~stest && ~wwall
161
162     beta(i) = rho*(-A_x*u_star(i) +A_x*u_in -A_y*v_star(i) ...
163                 + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
164
165     % At western boundary (x = 0)
166     wP_coeff = 0;
167
168     E_coeff = -rho*A_x^2/au(i);
169     eP_coeff = -E_coeff ;
170     T(i, i+1) = E_coeff;
171
172     N_coeff = -rho*A_y^2/av(i);
173     nP_coeff = -N_coeff;
174     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
175
176     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
177     sP_coeff = -S_coeff;
178     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
179
180
181     % At western wall
182     elseif ~etest && ~wtest && ~ntest && ~stest && wwall
183
184     beta(i) = rho*(-A_x*u_circ(i)... % West wall / inlet velocity is zero

```

```

185         +A_x*0 -A_y*v_circ(i) +...
186         A_y*v_circ(getRowUnder(i, N_wide, M_wide, N_total));
187
188     % At western boundary (x = 0)
189     wP_coeff = 0;
190
191     E_coeff = -rho*A_x^2/au(i);
192     eP_coeff = -E_coeff ;
193     T(i, i+1) = E_coeff;
194
195     N_coeff = -rho*A_y^2/av(i);
196     nP_coeff = -N_coeff;
197     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
198
199     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
200     sP_coeff = -S_coeff;
201     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
202
203
204     % At northern boundary (y = h)
205     elseif ~etest && ~wtest && ntest && ~stest && ~wwall
206
207         beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
208             + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
209
210     % At northern boundary (y = h)
211     nP_coeff = 0 ;
212
213     E_coeff = -rho*A_x^2/au(i);
214     eP_coeff = -E_coeff ;
215     T(i, i+1) = E_coeff;
216
217     W_coeff = -rho*A_x^2/au(i-1);
218     wP_coeff = -W_coeff;
219     T(i, i-1) = W_coeff;
220
221     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
222     sP_coeff = -S_coeff;
223     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
224
225
226     % At southern boundary (y = 0)
227     elseif ~etest && ~wtest && ~ntest && stest && ~wwall
228
229         beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
230             -A_y*v_star(i));
231
232     % At southern boundary (y = 0)
233     sP_coeff = 0;
234
235     E_coeff = -rho*A_x^2/au(i);
236     eP_coeff = -E_coeff ;
237     T(i, i+1) = E_coeff;
238
239     W_coeff = -rho*A_x^2/au(i-1);
240     wP_coeff = -W_coeff;
241     T(i, i-1) = W_coeff;
242
243     N_coeff = -rho*A_y^2/av(i);
244     nP_coeff = -N_coeff;
245     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
246
247
248     %Not at any boundary
249     else
250
251         beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) -A_y*v_star(i) + ...
252             A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
253
254     E_coeff = -rho*A_x^2/au(i);
255     eP_coeff = -E_coeff ;
256     T(i, i+1) = E_coeff;
257
258     W_coeff = -rho*A_x^2/au(i-1);
259     wP_coeff = -W_coeff;
260     T(i, i-1) = W_coeff;

```

```

261
262     N_coeff = -rho*A_y^2/av(i);
263     nP_coeff = -N_coeff;
264     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
265
266     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
267     sP_coeff = -S_coeff;
268     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
269
270     end % if
271
272     % Filling in the rest of the matrix, adding all point coefficients
273     T(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
274
275     % If the step is disabled the points below the step are blocked out
276     if onlyChannel && i <= N_wide*M_wide
277         T(i,i) = T(i,i) + 10e+30;
278     end %if
279
280     etest = false;
281     wtest = false;
282     ntest = false;
283     stest = false;
284     wwall = false;
285
286     end %for
287     p_corr = T\beta'; % Matrix inversion

```

E.5.1.5 plot_BFS.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %     Surface plots for velocities, pressure and pressure correction %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  showOutletView = true;% Save profiles seen from outlet in addition to inlet
5  az = 37.5; % Viewpoints when seen from outlet
6  el = 30;
7  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8  %% Settings
9  % filler is a value that is filled in where the step is. showStep can be
10 % adjusted if it is desirable to plot the profiles with zero at the step.
11 if ~exist('showStep','var')
12     filler = Inf;
13 else
14     if showStep == true
15         filler = 0;
16     else
17         filler = Inf;
18     end %if
19 end %if
20
21 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
22 %% Velocities to matrices
23 % u-velocity
24
25 u_fullplot = zeros(M_total+2, N_total+1);
26 u_fullplot(M_wide+2:end-1,1) = u_in;
27 u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
28     global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
29 u_fullplot(M_wide+2:end-1,2:end) = ...
30     global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
31 u_fullplot(1:M_wide, 1:N_narrow) = filler;
32
33 % Transformation from dimensionless to regular
34 u_fullplot = u_fullplot*u_in_true;
35
36 % Create a mesh for the plotting
37 [xu_plot,yu_plot] = meshgrid(x_0:del_x_true:x_N,...
38     [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);
39
40 % y-points are adjusted at the inlet because the southern wall of the
41 % narrow channel does not align with the u-velocity nodes in the wide sec.
42 jj = [linspace(0, H_total-h, M_total-M_narrow+1),...
43     linspace(H_total-h+del_y_true/2, y_M-del_y_true/2, M_narrow), H_total];
44 for i = 1:N_narrow
45     for j = 1:M_total
46         % alter the points of the plot for the wall of the narrow section

```

```

47     yu_plot(j,i) = jj(j);
48     end %for
49 end %for
50
51 % v-velocity
52 v_fullplot = zeros(m_total+2, N_total+1);           % Inlet is zero
53 v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
54     global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
55 v_fullplot(m_wide+2:end-1,2:end) = ...
56     global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
57 v_fullplot(1:m_wide, 1:N_narrow) = filler;
58
59 % Transformation from dimensionless to regular
60 v_fullplot = v_fullplot*u_in_true;
61
62 % Create a mesh for the plotting
63 [xv_plot,yv_plot] = meshgrid(x_0:del_x_true:x_N, y_0:del_y_true:y_M);
64
65
66 f1 = figure;
67 f = surf(xu_plot,yu_plot,u_fullplot);
68 % set(f,'edgecolor','none')
69 s = sprintf('Plot of $u_{new}$ after %d iterations', it );
70 % f = title(s);
71 % set(f, 'interpreter', 'latex', 'fontsize', 16)
72 set(gca,'TickLabelInterpreter','latex')
73 xlabel('$x$-direction [m]', 'interpreter', 'latex')
74 ylabel('$y$-direction [m]', 'interpreter', 'latex')
75 zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
76 ztickformat('%0.2f')
77 set(f1, 'Position', [3.6667 40.3333 555.3333 284.6667]);
78 %[left bottom width height]
79 saveas(gcf,'vnewBFS.png')
80 if showOutletView
81     view(az,el)
82     saveas(gcf,'vnewoutletBFS.png')
83     view(37.5,30) % back to normal
84 end % if
85
86
87 f2 = figure;
88 f = surf(xv_plot,yv_plot,v_fullplot);           % surf(x,y,z)
89 % set(f,'edgecolor','none')
90 s = sprintf('Plot of $v_{new}$ after %d iterations', it );
91 % f = title(s);
92 % set(f, 'interpreter', 'latex', 'fontsize', 16)
93 set(gca,'TickLabelInterpreter','latex')
94 xlabel('$x$-direction [m]', 'interpreter', 'latex')
95 ylabel('$y$-direction [m]', 'interpreter', 'latex')
96 zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
97 ztickformat('%0.2f')
98 set(f2, 'Position', [3.6667 327.0000 554.0000 314.0000]);
99 %[left bottom width height]
100 saveas(gcf,'vnewBFS.png')
101 if showOutletView
102     view(az,el)
103     saveas(gcf,'vnewoutletBFS.png')
104     view(37.5,30) % back to normal
105 end % if
106
107 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
108 %% Pressure to matrix
109
110 p_fullplot = zeros(M_total, N_total+1);
111 p_fullplot(1:M_wide,N_narrow+1:end-1) = ...
112     global2matrix(p_new(1:N_wide*M_wide), N_wide, M_wide);
113 p_fullplot(M_wide+1:end,1:end-1) = ...
114     global2matrix(p_new(N_wide*M_wide+1:end), N_total, M_narrow);
115 p_fullplot(1:M_wide, 1:N_narrow) = filler;
116 p_fullplot(:, end) = p_out;
117
118 % Transformation from dimensionless to regular
119 p_fullplot = p_fullplot*rho_true*u_in_true + p_atm;
120
121 % Create a mesh for the plotting
122 [xp_plot,yp_plot] = meshgrid(...

```



```

123     x_0:del_x_true:x_N, ...
124     y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);
125 xp_plot = xp_plot + del_x_true/2;
126 yp_plot = yp_plot + del_y_true/2;
127
128 f3 = figure;
129 f = surf(xp_plot,yp_plot,p_fullplot);
130 % set(f,'edgecolor','none')
131 s = sprintf('Plot of $p_{new}$ after %d iterations', it );
132 % f = title(s);
133 % set(f, 'interpreter', 'latex', 'fontsize', 16)
134 set(gca,'TickLabelInterpreter','latex')
135 xlabel('$x$-direction [m]', 'interpreter', 'latex')
136 ylabel('$y$-direction [m]', 'interpreter', 'latex')
137 zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
138 ztickformat('%.7f')
139 set(f3, 'Position', [ 721.6667 40.3333 560.0000 287.3333]);
140 %[left bottom width height]
141 saveas(gcf,'pnewBFS.png')
142 if showOutletView
143     view(az,el)
144     saveas(gcf,'pnewoutletBFS.png')
145     view(37.5,30) % back to normal
146 end % if
147
148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
149 %% Pressure correction to matrix
150 if it > 0 % Don't plot in case of initial profiles (plotinitialprofiles)
151
152     p_corrplot = zeros(M_total, N_total+1);
153     p_corrplot(1:M_wide,N_narrow+1:end-1) = ...
154         global2matrix(p_corr(1:N_wide*M_wide), N_wide, M_wide);
155     p_corrplot(M_wide+1:end,1:end-1) = ...
156         global2matrix(p_corr(N_wide*M_wide+1:end), N_total, M_narrow);
157     p_corrplot(1:M_wide, 1:N_narrow) = filler;
158
159     % Transformation from dimensionless to regular
160     p_corrplot = p_corrplot*rho_true*u_in_true;
161
162     % Create a mesh for the plotting
163     [xp_plot,yp_plot] = meshgrid(...
164         x_0:del_x_true:x_N, ...
165         y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);
166
167     f4 = figure;
168     f = surf(xp_plot,yp_plot,p_corrplot); % surf(x,y,z)
169     % set(f,'edgecolor','none')
170     s = sprintf('Plot of $p_{corr}$ after %d iterations', it );
171     % f = title(s);
172     % set(f, 'interpreter', 'latex', 'fontsize', 16)
173     set(gca,'TickLabelInterpreter','latex')
174     xlabel('$x$-direction [m]', 'interpreter', 'latex')
175     ylabel('$y$-direction [m]', 'interpreter', 'latex')
176     ztickformat('%.2f')
177     zlabel('Pressure correction $p_{corr}$, [Pa]', 'interpreter', 'latex')
178     set(f4, 'Position', [719.6667 329.6667 560.0000 311.3333]);
179     saveas(gcf,'pcorrBFS.png')
180
181
182     if showOutletView
183         view(az,el)
184         saveas(gcf,'pcorroutletBFS.png')
185         view(37.5,30) % back to normal
186     end % if
187 end %if

```

E.5.1.6 plotVelocityQuiver.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Velocity quiver plots %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 filler = 0; % For the quiver plots, the velocities at the step are set to
5 % zero and not Inf, rectangles are therefore used to block
6 % out the step from the plots afterwards.
7
8 % u-velocity

```

```

 9 u_fullplot = zeros(M_total+2, N_total+1);
10 u_fullplot(M_wide+2:end-1,1) = u_in;
11 u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
12     global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
13 u_fullplot(M_wide+2:end-1,2:end) = ...
14     global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
15 u_fullplot(1:M_wide, 1:N_narrow) = 0;
16
17 % Transformation from dimensionless to regular
18 u_fullplot = u_fullplot*u_in_true;
19
20
21 % v-velocity
22 v_fullplot = zeros(m_total+2, N_total+1);
23 v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
24     global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
25 v_fullplot(m_wide+2:end-1,2:end) = ...
26     global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
27 v_fullplot(1:m_wide, 1:N_narrow) = filler;
28
29 % Transformation from dimensionless to regular
30 v_fullplot = v_fullplot*u_in_true;
31
32
33 uSN = zeros(M_total, N_total);
34 vSN = zeros(M_total, N_total);
35 for i = 2:N_total+1
36     for j = 1:M_total
37         uSN(j,i-1) = 1/2*(u_fullplot(j+1,i-1) + u_fullplot(j+1,i));
38     end %for
39 end %for
40 for j = 2:M_total+1
41     for i = 1:N_total
42         vSN(j-1,i) = 1/2*(v_fullplot(j-1,i) + v_fullplot(j,i));
43     end %for
44 end %for
45
46 % Need to make a combined velocity vector
47 combvel = sqrt(uSN.^2 + vSN.^2);
48
49 % Create a mesh for the plotting
50 [xSN,ySN] = meshgrid(...
51     x_0:del_x:x_N-del_x, ...
52     y_0+del_y/2:del_y:y_M-del_y/2);
53
54
55 fql = figure;
56 qn = quiver( xSN, ySN , uSN , vSN,'LineWidth',0.5,'Color','k');
57
58 %Block out the step
59 r = rectangle('Position',[0.03 -0.05 3 0.55]);
60 r.FaceColor = [1 1 1];
61 r.EdgeColor = 'none';%'k';
62 r.LineWidth = .0000010;
63
64 s = rectangle('Position',[0.03 -0.05 22 0.05]);
65 s.FaceColor = [1 1 1];
66 s.EdgeColor = 'none';%'k';
67 s.LineWidth = .0000010;
68
69 t = rectangle('Position',[0.03 1.5 22 0.05]);
70 t.FaceColor = [1 1 1];
71 t.EdgeColor = 'none';%'k';
72 t.LineWidth = .0000010;
73
74 hold on
75 set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
76     'Marker','o','MarkerSize',1,'ShowArrowHead','on')
77 s = sprintf('Plot of velocities as vectors after %d iterations', it );
78 % f = title(s);
79 ax = gca;
80 % set(f, 'interpreter', 'latex', 'fontSize', 16)
81 set(gca,'TickLabelInterpreter','latex')
82 ax.FontSize = 12;
83 xlabel('$x$-direction [m]', 'interpreter', 'latex')
84 xlim([0,22])

```

```

85 ylabel('$y$-direction [m]', 'interpreter', 'latex')
86 ylim([-0.05,1.55])
87 ytickformat('%0.1f')
88 set(fq1,'Position', [3 250 717 420]);
89 saveas(gcf,'velocityquiver.png')
90 ax.Layer = 'top';
91
92
93 fq2 = figure;
94 qn = quiver(...
95     xSN, ySN , uSN , vSN,...%u_fullplot(1:end-1,:))
96     'LineWidth',0.5,'Color','k');
97
98     r = rectangle('Position',[0.03 -0.05 3 0.55]);
99 r.FaceColor = [1 1 1];
100 r.EdgeColor = 'none';%k';
101 r.LineWidth = .0000010;
102
103 s = rectangle('Position',[0.03 -0.05 22 0.05]);
104 s.FaceColor = [1 1 1];
105 s.EdgeColor = 'none';%k';
106 s.LineWidth = .0000010;
107
108 hold on
109 set(qn,'AutoScale','on', 'AutoScaleFactor', 1.5,'Marker','o',...
110     'MarkerSize',1,'MaxHeadSize',0.01);%ShowArrowHead','off')
111 % qw = quiver(...
112 %     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
113 %         'LineWidth',0.5,'Color','k');
114 s = sprintf(...
115     'Plot of velocities as vectors after %d iterations scales x 1.5', it );
116 % f = title(s);
117 ax = gca;
118 % set(f, 'interpreter', 'latex', 'fontsize', 16)
119 set(gca,'TickLabelInterpreter','latex')
120 ax.FontSize = 12;
121 xlabel('$x$-direction [m]', 'interpreter', 'latex')
122 xlim([1-1/4,1*3])
123 ylabel('$y$-direction [m]', 'interpreter', 'latex')
124 ylim([0,H+H/4])
125 ytickformat('%0.1f')
126 set(fq2,'Position', [724 250 560 420]);
127 saveas(gcf,'velocityquiver_zoomed.png')
128 ax.Layer = 'top';
129
130
131 fq3 = figure;
132 qn = quiver(...
133     xSN(1:M_wide,N_narrow+1:N_narrow*2), ...
134     ySN(1:M_wide,N_narrow+1:N_narrow*2), ...
135     uSN(1:M_wide,N_narrow+1:N_narrow*2), ...
136     vSN(1:M_wide,N_narrow+1:N_narrow*2),...%u_fullplot(1:end-1,:))
137     'LineWidth',0.5,'Color','k');
138
139 r = rectangle('Position',[0.03 -0.05 3 0.55]);
140 r.FaceColor = [1 1 1];
141 r.EdgeColor = 'none';%k';
142 r.LineWidth = .0000010;
143
144 s = rectangle('Position',[0.03 -0.05 22 0.05]);
145 s.FaceColor = [1 1 1];
146 s.EdgeColor = 'none';%k';
147 s.LineWidth = .0000010;
148
149 hold on
150 set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
151     'Marker','o','MarkerSize',1,'ShowArrowHead','on')
152 % qw = quiver(...
153 %     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
154 %         'LineWidth',0.5,'Color','k');
155 s = sprintf(...
156     'Plot of velocities as vectors after %d iterations, scaled * 2', it );
157 % f = title(s);
158 ax = gca;
159 % set(f, 'interpreter', 'latex', 'fontsize', 16)
160 set(gca,'TickLabelInterpreter','latex')

```

```

161 ax.FontSize = 12;
162 xlabel('$x$-direction [m]', 'interpreter', 'latex')
163 xlim([1,2*1])
164 ylabel('$y$-direction [m]', 'interpreter', 'latex')
165 ylim([0,H])
166 ytickformat('%.1f')
167 set(fq3,'Position',[724 250 560 420]);
168 saveas(gcf,'velocityquiver_zoomed.png')
169 ax.Layer = 'top';

```

E.5.1.7 plotColoredQuiver.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %           Colored velocity quiver plots           %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  filler = 0;    % For the quiver plots, the velocities at the step are set to
5                 % zero and not Inf, rectangles are therefore used to block
6                 % out the step from the plots afterwards.
7  levels = 50;    % Number of different colors for the representation
8  showvals = false;    % Show the value of each color
9  lines = 'none';    % Show lines in between each color
10
11 % u-velocity
12 u_fullplot = zeros(M_total+2, N_total+1);
13 u_fullplot(M_wide+2:end-1,1) = u_in;
14 u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
15     global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
16 u_fullplot(M_wide+2:end-1,2:end) = ...
17     global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
18 u_fullplot(1:M_wide, 1:N_narrow) = 0;
19
20 % Transformation from dimensionless to regular
21 u_fullplot = u_fullplot*u_in_true;
22
23
24 % v-velocity
25 v_fullplot = zeros(m_total+2, N_total+1);
26 v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
27     global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
28 v_fullplot(m_wide+2:end-1,2:end) = ...
29     global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
30 v_fullplot(1:m_wide, 1:N_narrow) = filler;
31
32 % Transformation from dimensionless to regular
33 v_fullplot = v_fullplot*u_in_true;
34
35
36 uSN = zeros(M_total, N_total);
37 vSN = zeros(M_total, N_total);
38 for i = 2:N_total+1
39     for j = 1:M_total
40         uSN(j,i-1) = 1/2*(u_fullplot(j+1,i-1) + u_fullplot(j+1,i));
41     end %for
42 end %for
43 for j = 2:M_total+1
44     for i = 1:N_total
45         vSN(j-1,i) = 1/2*(v_fullplot(j-1,i) + v_fullplot(j,i));
46     end %for
47 end %for
48
49
50 % Need to make a combined velocity vector
51 combvel = sqrt(uSN.^2 + vSN.^2);
52
53
54
55 % Create a mesh for the plotting
56 [xSN,ySN] = meshgrid(...
57     x_0+ del_x_true/2:del_x_true:x_N-del_x_true/2, ...
58     y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);
59
60 combvelwall = [zeros(1,N_total); combvel ; zeros(1,N_total)];
61
62 fq1 = figure;
63 % Contour plot
64 [M,c] = contourf([xSN(1,:) ; xSN ; xSN(end,:)],...

```

```

65     [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
66     combvelwall,levels);
67 c.LineColor = lines;
68 hold on
69 qn = quiver( xSN, ySN , uSN , vSN,'LineWidth',0.5,'Color','k');
70
71 %Block out the step
72 r = rectangle('Position',[0.03 -0.05 3 0.55]);
73 r.FaceColor = [1 1 1];
74 r.EdgeColor = 'none';%'k';
75 r.LineWidth = .0000010;
76
77
78 hold on
79 set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
80     'Marker','o','MarkerSize',1,'ShowArrowHead','on')
81 s = sprintf('Plot of velocities as vectors after %d iterations', it );
82 % f = title(s);
83 ax = gca;
84 % set(f, 'interpreter', 'latex', 'fontsize', 16)
85 set(gca,'TickLabelInterpreter','latex')
86 ax.FontSize = 12;
87 xlabel('$x$-direction [m]', 'interpreter', 'latex')
88 xlim([0,22])
89 ylabel('$y$-direction [m]', 'interpreter', 'latex')
90 ylim([-0.05,1.55])
91 ytickformat('%.1f')
92 set(fq1,'Position', [3 250 717 420]);
93 saveas(gcf,'velocityquiver.png')
94 ax.Layer = 'top';
95
96
97 fq2 = figure;
98 [M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
99     [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
100     combvelwall,levels);
101 c.LineColor = lines;
102 hold on
103 qn = quiver(...
104     xSN, ySN , uSN , vSN,...%u_fullplot(1:end-1,:)
105     'LineWidth',0.5,'Color','k');
106
107 r = rectangle('Position',[0.03 -0.05 3 0.55]);
108 r.FaceColor = [1 1 1];
109 r.EdgeColor = 'none';%'k';
110 r.LineWidth = .0000010;
111
112
113 hold on
114 set(qn,'AutoScale','on', 'AutoScaleFactor', 2.1,'Marker','o',...
115     'MarkerSize',1,'MaxHeadSize',0.01);%'ShowArrowHead','off')
116 % qw = quiver(...
117 %     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
118 %     'LineWidth',0.5,'Color','k');
119 s = sprintf(...
120     'Plot of velocities as vectors after %d iterations scales x 1.5', it );
121 % f = title(s);
122 ax = gca;
123 % set(f, 'interpreter', 'latex', 'fontsize', 16)
124 set(gca,'TickLabelInterpreter','latex')
125 ax.FontSize = 12;
126 xlabel('$x$-direction [m]', 'interpreter', 'latex')
127 xlim([1-1/4,1*3])
128 ylabel('$y$-direction [m]', 'interpreter', 'latex')
129 ylim([0,H+H/4])
130 ytickformat('%.1f')
131 set(fq2,'Position', [724 250 560 420]);
132 saveas(gcf,'velocityquiver1zoomed.png')
133 ax.Layer = 'top';
134
135
136 fq3 = figure;
137 [M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
138     [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
139     combvelwall,levels);
140 c.LineColor = lines;

```

```

141 hold on
142 qn = quiver(...
143     xSN(1:M_wide,N_narrow+1:N_narrow*2), ...
144     ySN(1:M_wide,N_narrow+1:N_narrow*2) ,...
145     uSN(1:M_wide,N_narrow+1:N_narrow*2) , ...
146     vSN(1:M_wide,N_narrow+1:N_narrow*2),...%u_fullplot(1:end-1,:)
147     'LineWidth',0.5,'Color','k');
148
149 r = rectangle('Position',[0.03 -0.05 3 0.55]);
150 r.FaceColor = [1 1 1];
151 r.EdgeColor = 'none';%'k';
152 r.LineWidth = .0000010;
153
154 hold on
155 set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 2.1,...
156     'Marker','o','MarkerSize',1,'ShowArrowHead','on')
157 % qw = quiver(...
158 %     xv_plot, yv_plot, uplot(1:end-1,:), vplot,...
159 %     'LineWidth',0.5,'Color','k');
160 s = sprintf(...
161     'Plot of velocities as vectors after %d iterations, scaled * 2', it );
162 % f = title(s);
163 ax = gca;
164 % set(f, 'interpreter', 'latex', 'fontSize', 16)
165 set(gca,'TickLabelInterpreter','latex')
166 ax.FontSize = 12;
167 xlabel('$x$-direction [m]', 'interpreter', 'latex')
168 xlim([1,2*1])
169 ylabel('$y$-direction [m]', 'interpreter', 'latex')
170 ylim([0,H])
171 ytickformat('%1f')
172 set(fq3,'Position',[724 250 560 420]);
173 saveas(gcf,'velocityquiver2zoomed.png')
174 ax.Layer = 'top';

```

E.5.1.8 plotVelocityCorrection.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                               Surface plots for velocity corrections %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %% Settings
5  % filler is a value that is filled in where the step is. showStep can be
6  % adjusted if it is desirable to plot the profiles with zero at the step.
7  if ~exist('showStep','var')
8     filler = Inf;
9  else
10     if showStep == true
11         filler = 0;
12     else
13         filler = Inf;
14     end %if
15 end %if
16
17 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
18 %% Velocities to matrices
19 % u-velocity
20
21 u_fullplot = zeros(M_total+2, N_total+1);
22 u_fullplot(M_wide+2:end-1,1) = 0;
23 u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
24     global2matrix(u_corr(1:N_wide*M_wide), N_wide, M_wide);
25 u_fullplot(M_wide+2:end-1,2:end) = ...
26     global2matrix(u_corr(N_wide*M_wide+1:end), N_total, M_narrow);
27 u_fullplot(1:M_wide, 1:N_narrow) = filler;
28
29 % Transformation from dimensionless to regular
30 u_fullplot = u_fullplot*u_in_true;
31
32 % Create a mesh for the plotting
33 [xu_plot,yu_plot] = meshgrid(...
34     x_0:del_x_true:x_N, ...
35     [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);
36
37 % y-points are adjusted at the inlet because the southern wall of the
38 % narrow channel does not align with the u-velocity nodes in the wide sec.
39 for i = 1:N_narrow

```

```

40     % alter the points of the plot for the wall of the narrow section
41     yu_plot(:,i) = [linspace(0, H_total-h, M_total-M_narrow+1), ...
42                   linspace(H_total-h+del_y_true/2, y_M-del_y_true/2, M_narrow), ...
43                   H_total];
44 end %for
45
46
47 % v-velocity
48 v_fullplot = zeros(m_total+2, N_total+1);           % Inlet is zero
49 v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
50     global2matrix(v_corr(1:N_wide*m_wide), N_wide, m_wide);
51 v_fullplot(m_wide+2:end-1,2:end) = ...
52     global2matrix(v_corr(N_wide*m_wide+1:end), N_total, m_narrow);
53 v_fullplot(1:m_wide, 1:N_narrow) = filler;
54
55 % Transformation from dimensionless to regular
56 v_fullplot = v_fullplot*u_in_true;
57
58 % Create a mesh for the plotting
59 [xv_plot,yv_plot] = meshgrid(...
60     x_0:del_x_true:x_N, ...
61     y_0:del_y_true:y_M);
62
63 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
64 %% Plot
65
66 f1 = figure;
67 f = surf(xu_plot,yu_plot,u_fullplot);           % surf(x,y,z)
68 % set(f,'edgecolor','none')
69 s = sprintf('Plot of $u_{corr}$ after %d iterations', it);
70 % f = title(s);
71 % set(f,'interpreter','latex','fontsize',16)
72 set(gca,'TickLabelInterpreter','latex')
73 xlabel('$x$-direction [m]', 'interpreter','latex')
74 ylabel('$y$-direction [m]', 'interpreter','latex')
75 zlabel('Velocity $u$, [m/s]', 'interpreter','latex')
76 ztickformat('%.2f')
77 set(f1, 'Position', [3.6667 40.3333 555.3333 284.6667]); %[left bottom width
78     height]
79 saveas(gcf,'ucorrBFS.png')
80
81 f2 = figure;
82 f = surf(xv_plot,yv_plot,v_fullplot);           % surf(x,y,z)
83 % set(f,'edgecolor','none')
84 s = sprintf('Plot of $v_{corr}$ after %d iterations', it);
85 % f = title(s);
86 % set(f,'interpreter','latex','fontsize',16)
87 set(gca,'TickLabelInterpreter','latex')
88 xlabel('$x$-direction [m]', 'interpreter','latex')
89 ylabel('$y$-direction [m]', 'interpreter','latex')
90 zlabel('Velocity $v$, [m/s]', 'interpreter','latex')
91 ztickformat('%.2f')
92 set(f2, 'Position', [3.6667 327.0000 554.0000 314.0000]); %[left bottom width
93     height]
94 saveas(gcf,'vcorrBFS.png')

```

E.5.1.9 plotIntermediates.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %           Surface plots for the intermediate velocities           %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 showOutletView = true;% Save profiles seen from outlet in addition to inlet
5 az = 37.5; % Viewpoints when seen from outlet
6 el = 30;
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8 %% Settings
9 % filler is a value that is filled in where the step is. showStep can be
10 % adjusted if it is desirable to plot the profiles with zero at the step.
11 if ~exist('showStep','var')
12     filler = Inf;
13 else
14     if showStep == true
15         filler = 0;
16     else
17         filler = Inf;
18     end %if

```

```

19 end %if
20
21 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
22 %% Velocities to matrices
23 % u-velocity
24
25 u_fullplot = zeros(M_total+2, N_total+1);
26 u_fullplot(M_wide+2:end-1,1) = u_in;
27 u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
28     global2matrix(u_star(1:N_wide*M_wide), N_wide, M_wide);
29 u_fullplot(M_wide+2:end-1,2:end) = ...
30     global2matrix(u_star(N_wide*M_wide+1:end), N_total, M_narrow);
31 u_fullplot(1:M_wide, 1:N_narrow) = filler;
32
33 % Transformation from dimensionless to regular
34 u_fullplot = u_fullplot*u_in_true;
35
36 % Create a mesh for the plotting
37 [xu_plot,yu_plot] = meshgrid(x_0:del_x_true:x_N,...
38     [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);
39
40 % y-points are adjusted at the inlet because the southern wall of the
41 % narrow channel does not align with the u-velocity nodes in the wide sec.
42 jj = [linspace(0, H_total-h, M_total-M_narrow+1), linspace(H_total-h+del_y_true/2, y_M
43     -del_y_true/2, M_narrow), H_total];
44 for i = 1:N_narrow
45     for j = 1:M_total
46         % alter the points of the plot for the wall of the narrow section
47         yu_plot(j,i) = jj(j);
48     end %for
49 end %for
50
51 % v-velocity
52 v_fullplot = zeros(m_total+2, N_total+1); % Inlet is zero
53 v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
54     global2matrix(v_star(1:N_wide*m_wide), N_wide, m_wide);
55 v_fullplot(m_wide+2:end-1,2:end) = ...
56     global2matrix(v_star(N_wide*m_wide+1:end), N_total, m_narrow);
57 v_fullplot(1:m_wide, 1:N_narrow) = filler;
58
59 % Transformation from dimensionless to regular
60 v_fullplot = v_fullplot*u_in_true;
61
62 % Create a mesh for the plotting
63 [xv_plot,yv_plot] = meshgrid(x_0:del_x_true:x_N, y_0:del_y_true:y_M);
64
65 f1 = figure;
66 f = surf(xu_plot,yu_plot,u_fullplot); % surf(x,y,z)
67 % set(f,'edgecolor','none')
68 s = sprintf('Plot of $u_{star}$ after %d iterations', it );
69 f = title(s);
70 set(f, 'interpreter', 'latex', 'fontsize', 16)
71 set(gca, 'TickLabelInterpreter','latex')
72 xlabel('$x$-direction [m]', 'interpreter', 'latex')
73 ylabel('$y$-direction [m]', 'interpreter', 'latex')
74 zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
75 ztickformat('%0.2f')
76 set(f1, 'Position', [3.6667 40.3333 555.3333 284.6667]); %[left bottom width
77     height]
78 saveas(gcf,'ustarBFS.png')
79 if showOutletView
80     view(az,el)
81     saveas(gcf,'ustaroutletBFS.png')
82     view(37.5,30) % back to normal
83 end % if
84
85 f2 = figure;
86 f = surf(xv_plot,yv_plot,v_fullplot); % surf(x,y,z)
87 % set(f,'edgecolor','none')
88 s = sprintf('Plot of $v_{star}$ after %d iterations', it );
89 f = title(s);
90 set(f, 'interpreter', 'latex', 'fontsize', 16)
91 set(gca, 'TickLabelInterpreter','latex')
92 xlabel('$x$-direction [m]', 'interpreter', 'latex')
93 ylabel('$y$-direction [m]', 'interpreter', 'latex')

```



```

93 xlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
94 ztickformat('%.2f')
95 set(f2, 'Position', [3.6667 327.0000 554.0000 314.0000]); %[left bottom width
    height]
96 saveas(gcf, 'vstarBFS.png')
97 if showOutletView
98     view(az,el)
99     saveas(gcf, 'vstaroutletBFS.png')
100     view(37.5,30) % back to normal
101 end % if

```

E.5.1.10 plot_BFS_iterations.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %           Plots for velocities, pressure and pressure correction      %
3  %           saved for each specified iteration                          %
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  %% Settings
6  % filler is a value that is filled in where the step is. showStep can be
7  % adjusted if it is desirable to plot the profiles with zero at the step.
8  if ~exist('showStep','var')
9     filler = Inf;
10 else
11     if showStep == true
12         filler = 0;
13     else
14         filler = Inf;
15     end %if
16 end %if
17
18 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
19 %% Velocities to matrices
20 % u-velocity
21
22 u_fullplot = zeros(M_total+2, N_total+1);
23 u_fullplot(M_wide+2:end-1,1) = u_in;
24 u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
25     global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
26 u_fullplot(M_wide+2:end-1,2:end) = ...
27     global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
28 u_fullplot(1:M_wide, 1:N_narrow) = filler;
29
30 % Transformation from dimensionless to regular
31 u_fullplot = u_fullplot*u_in_true;
32
33 % Create a mesh for the plotting
34 [xu_plot,yu_plot] = meshgrid(x_0:del_x_true:x_N,...
35     [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);
36
37 % y-points are adjusted at the inlet because the southern wall of the
38 % narrow channel does not align with the u-velocity nodes in the wide sec.
39 jj = [linspace(0, H_total-h, M_total-M_narrow+1), ...
40     linspace(H_total-h+del_y_true/2, y_M-del_y_true/2, M_narrow), H_total];
41 for i = 1:N_narrow
42     for j = 1:M_total
43         % alter the points of the plot for the wall of the narrow section
44         yu_plot(j,i) = jj(j);
45     end %for
46 end %for
47
48 % v-velocity
49 v_fullplot = zeros(m_total+2, N_total+1); % Inlet is zero
50 v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
51     global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
52 v_fullplot(m_wide+2:end-1,2:end) = ...
53     global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
54 v_fullplot(1:m_wide, 1:N_narrow) = filler;
55
56 % Transformation from dimensionless to regular
57 v_fullplot = v_fullplot*u_in_true;
58
59 % Create a mesh for the plotting
60 [xv_plot,yv_plot] = meshgrid(x_0:del_x_true:x_N, y_0:del_y_true:y_M);
61
62
63 %% Plot velocities

```

```

64 f1 = figure('units','normalized','outerposition',[0 0 1 1]);
65
66
67 subplot(2,2,1);
68 f = surf(xu_plot,yu_plot,u_fullplot);           % surf(x,y,z)
69 s = sprintf('Plot of $u_{new}$ after %d iterations', it );
70 f = title(s);
71 set(f, 'interpreter', 'latex', 'fontsize', 16)
72 set(gca,'TickLabelInterpreter','latex')
73 xlabel('$x$-direction [m]', 'interpreter', 'latex')
74 ylabel('$y$-direction [m]', 'interpreter', 'latex')
75 zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
76 ztickformat('%.2f')
77
78 subplot(2,2,3);
79 f = surf(xv_plot,yv_plot,v_fullplot);           % surf(x,y,z)
80 s = sprintf('Plot of $v_{new}$ after %d iterations', it );
81 f = title(s);
82 set(f, 'interpreter', 'latex', 'fontsize', 16)
83 set(gca,'TickLabelInterpreter','latex')
84 xlabel('$x$-direction [m]', 'interpreter', 'latex')
85 ylabel('$y$-direction [m]', 'interpreter', 'latex')
86 zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
87 ztickformat('%.2f')
88 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89 %% Pressure
90
91 p_fullplot = zeros(M_total, N_total+1);
92 p_fullplot(1:M_wide,N_narrow+1:end-1) = ...
93     global2matrix(p_new(1:N_wide*M_wide), N_wide, M_wide);
94 p_fullplot(M_wide+1:end,1:end-1) = ...
95     global2matrix(p_new(N_wide*M_wide+1:end), N_total, M_narrow);
96 p_fullplot(1:M_wide, 1:N_narrow) = filler;
97 p_fullplot(:, end) = p_out;
98
99 % Transformation from dimensionless to regular
100 p_fullplot = p_fullplot*rho_true*u_in_true + p_atm;
101
102 % Create a mesh for the plotting
103 [xp_plot,yp_plot] = meshgrid(...
104     x_0:del_x_true:x_N, ...
105     y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);
106
107 subplot(2,2,2);
108 f = surf(xp_plot,yp_plot,p_fullplot);           % surf(x,y,z)
109 s = sprintf('Plot of $p_{new}$ after %d iterations', it );
110 f = title(s);
111 set(f, 'interpreter', 'latex', 'fontsize', 16)
112 set(gca,'TickLabelInterpreter','latex')
113 xlabel('$x$-direction [m]', 'interpreter', 'latex')
114 ylabel('$y$-direction [m]', 'interpreter', 'latex')
115 zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
116 ztickformat('%.7f')
117
118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119 %% Pressure correction to matrix
120
121 p_corrplot = zeros(M_total, N_total+1);
122 p_corrplot(1:M_wide,N_narrow+1:end-1) = ...
123     global2matrix(p_corr(1:N_wide*M_wide), N_wide, M_wide);
124 p_corrplot(M_wide+1:end,1:end-1) = ...
125     global2matrix(p_corr(N_wide*M_wide+1:end), N_total, M_narrow);
126 p_corrplot(1:M_wide, 1:N_narrow) = filler;
127
128 % Transformation from dimensionless to regular
129 p_corrplot = p_corrplot*rho_true*u_in_true;
130
131 % Create a mesh for the plotting
132 [xp_plot,yp_plot] = meshgrid(...
133     x_0:del_x_true:x_N, ...
134     y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);
135
136
137 subplot(2,2,4);
138 f = surf(xp_plot,yp_plot,p_corrplot);           % surf(x,y,z)
139 s = sprintf('Plot of $p_{corr}$ after %d iterations', it );

```

```

140 f = title(s);
141 set(f, 'interpreter', 'latex', 'fontsize', 16)
142 set(gca, 'TickLabelInterpreter', 'latex')
143 xlabel('$x$-direction [m]', 'interpreter', 'latex')
144 ylabel('$y$-direction [m]', 'interpreter', 'latex')
145 zlabel('Pressure correction $p$' '$', [Pa], 'interpreter', 'latex')
146 ztickformat('%.2f')
147
148 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
149 %% Make .gif
150
151 axis tight manual % this ensures that getframe() returns a consistent size
152 filename = 'itdev_allfour.gif';
153 % Capture the plot as an image
154 frame = getframe(f1);
155 im = frame2im(frame);
156 [imind,cm] = rgb2ind(im,256);
157 % Write to the GIF File
158 if (it == 1 && plotInitialProfiles == false) || ...
159     (it == 0 && plotInitialProfiles == true)
160     imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
161 else
162     imwrite(imind,cm,filename,'gif','WriteMode','append');
163 end
164
165 close all

```

E.5.1.11 plotVelInts_BFS_iterations.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %           Plots for velocity intermediates and           %
3 %           saved for each specified iteration             %
4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5 %% Settings
6 % filler is a value that is filled in where the step is. showStep can be
7 % adjusted if it is desirable to plot the profiles with zero at the step.
8 if ~exist('showStep','var')
9     filler = Inf;
10 else
11     if showStep == true
12         filler = 0;
13     else
14         filler = Inf;
15     end %if
16 end %if
17
18 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
19 %% u-velocity
20 u_circplot = zeros(M_total+2, N_total+1);
21 u_circplot(M_wide+2:end-1,1) = u_in;
22 u_circplot(2:M_wide+1,N_narrow+2:end) = ...
23     global2matrix(u_circ(1:N_wide*M_wide), N_wide, M_wide);
24 u_circplot(M_wide+2:end-1,2:end) = ...
25     global2matrix(u_circ(N_wide*M_wide+1:end), N_total, M_narrow);
26 u_circplot(1:M_wide, 1:N_narrow) = filler;
27
28 % Transformation from dimensionless to regular
29 u_circplot = u_circplot*u_in_true;
30
31
32 u_starplot = zeros(M_total+2, N_total+1);
33 u_starplot(M_wide+2:end-1,1) = u_in;
34 u_starplot(2:M_wide+1,N_narrow+2:end) = ...
35     global2matrix(u_star(1:N_wide*M_wide), N_wide, M_wide);
36 u_starplot(M_wide+2:end-1,2:end) = ...
37     global2matrix(u_star(N_wide*M_wide+1:end), N_total, M_narrow);
38 u_starplot(1:M_wide, 1:N_narrow) = filler;
39
40 % Transformation from dimensionless to regular
41 u_starplot = u_starplot*u_in_true;
42
43
44 u_corrplot = zeros(M_total+2, N_total+1);
45 u_corrplot(M_wide+2:end-1,1) = 0; % no correction at known
46 u_corrplot(2:M_wide+1,N_narrow+2:end) = ...
47     global2matrix(u_corr(1:N_wide*M_wide), N_wide, M_wide);

```

```

48 u_corrplot(M_wide+2:end-1,2:end) = ...
49     global2matrix(u_corr(N_wide*M_wide+1:end), N_total, M_narrow);
50 u_corrplot(1:M_wide, 1:N_narrow) = filler;
51
52 % Transformation from dimensionless to regular
53 u_corrplot = u_corrplot*u_in_true;
54
55
56 u_newplot = zeros(M_total+2, N_total+1);
57 u_newplot(M_wide+2:end-1,1) = u_in;
58 u_newplot(2:M_wide+1,N_narrow+2:end) = ...
59     global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
60 u_newplot(M_wide+2:end-1,2:end) = ...
61     global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
62 u_newplot(1:M_wide, 1:N_narrow) = filler;
63
64 % Transformation from dimensionless to regular
65 u_newplot = u_newplot*u_in_true;
66
67
68 % Create a mesh for the plotting
69 [xu_plot,yu_plot] = meshgrid(...
70     x_0:del_x_true:x_N, ...
71     [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);
72
73 for i = 1:N_narrow % alter the points of the plot for the wall of the narrow section
74     yu_plot(:,i) = [linspace(0, H_total-h, M_total-M_narrow+1), linspace(H_total-h+
75         del_y_true/2, y_M-del_y_true/2, M_narrow), H_total];
76     % yu_plot(i,:) = [linspace(0, H_total-h, M-M+1), linspace(H_total-h+del_y_true/2,
77         y_M-del_y_true/2, M), H_total];
78 end %for
79
80
81
82 f0 = figure('units','normalized','outerposition',[0 0 1 1]);
83
84
85 subplot(2,2,1);
86 f = surf(xu_plot,yu_plot,u_circplot); % surf(x,y,z)
87
88 s = sprintf('Plot of $u_{circ}$ after %d iterations', it );
89 f = title(s);
90 set(f, 'interpreter', 'latex', 'fontsize', 16)
91 set(gca,'TickLabelInterpreter','latex')
92 xlabel('$x$-direction [m]', 'interpreter', 'latex')
93 ylabel('$y$-direction [m]', 'interpreter', 'latex')
94 zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
95 ztickformat('%0.2f')
96
97 subplot(2,2,3);
98 f = surf(xu_plot,yu_plot,u_starplot); % surf(x,y,z)
99 s = sprintf('Plot of $u_{star}$ after %d iterations', it );
100 f = title(s);
101 set(f, 'interpreter', 'latex', 'fontsize', 16)
102 set(gca,'TickLabelInterpreter','latex')
103 xlabel('$x$-direction [m]', 'interpreter', 'latex')
104 ylabel('$y$-direction [m]', 'interpreter', 'latex')
105 zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
106 ztickformat('%0.2f')
107
108 subplot(2,2,2);
109 f = surf(xu_plot,yu_plot,u_corrplot); % surf(x,y,z)
110 s = sprintf('Plot of $u_{corr}$ after %d iterations', it );
111 f = title(s);
112 set(f, 'interpreter', 'latex', 'fontsize', 16)
113 set(gca,'TickLabelInterpreter','latex')
114 xlabel('$x$-direction [m]', 'interpreter', 'latex')
115 ylabel('$y$-direction [m]', 'interpreter', 'latex')
116 zlabel('Velocity, [m/s]', 'interpreter', 'latex')
117 ztickformat('%0.2f')
118
119 subplot(2,2,4);
120 f = surf(xu_plot,yu_plot,u_newplot); % surf(x,y,z)
121 s = sprintf('Plot of $u_{new}$ after %d iterations', it );

```

```

122 f = title(s);
123 set(f, 'interpreter', 'latex', 'fontsize', 16)
124 set(gca, 'TickLabelInterpreter', 'latex')
125 xlabel('$x$-direction [m]', 'interpreter', 'latex')
126 ylabel('$y$-direction [m]', 'interpreter', 'latex')
127 zlabel('Velocity, [m/s]', 'interpreter', 'latex')
128 ztickformat('%.2f')
129
130
131
132 axis tight manual % this ensures that getframe() returns a consistent size
133 filename = 'itdev_uintermediates.gif';
134 % Capture the plot as an image
135 frame = getframe(f0);
136 im = frame2im(frame);
137 [imind,cm] = rgb2ind(im,256);
138 % Write to the GIF File
139 if it == 1
140     imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
141 else
142     imwrite(imind,cm,filename,'gif','WriteMode','append');
143 end
144
145 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
146 %% v-velocity
147 v_circplot = zeros(m_total+2, N_total+1);
148 v_circplot(2:m_wide+1,N_narrow+2:end) = ...
149     global2matrix(v_circ(1:N_wide*m_wide), N_wide, m_wide);
150 v_circplot(m_wide+2:end-1,2:end) = ...
151     global2matrix(v_circ(N_wide*m_wide+1:end), N_total, m_narrow);
152 v_circplot(1:m_wide, 1:N_narrow) = filler;
153
154 % Transformation from dimensionless to regular
155 v_circplot = v_circplot*u_in_true;
156
157
158 v_starplot = zeros(m_total+2, N_total+1);
159 v_starplot(2:m_wide+1,N_narrow+2:end) = ...
160     global2matrix(v_star(1:N_wide*m_wide), N_wide, m_wide);
161 v_starplot(m_wide+2:end-1,2:end) = ...
162     global2matrix(v_star(N_wide*m_wide+1:end), N_total, m_narrow);
163 v_starplot(1:m_wide, 1:N_narrow) = filler;
164
165 % Transformation from dimensionless to regular
166 v_starplot = v_starplot*u_in_true;
167
168
169 v_corrplot = zeros(m_total+2, N_total+1);
170 v_corrplot(2:m_wide+1,N_narrow+2:end) = ...
171     global2matrix(v_corr(1:N_wide*m_wide), N_wide, m_wide);
172 v_corrplot(m_wide+2:end-1,2:end) = ...
173     global2matrix(v_corr(N_wide*m_wide+1:end), N_total, m_narrow);
174 v_corrplot(1:m_wide, 1:N_narrow) = filler;
175
176 % Transformation from dimensionless to regular
177 v_corrplot = v_corrplot*u_in_true;
178
179
180 v_newplot = zeros(m_total+2, N_total+1);
181 v_newplot(2:m_wide+1,N_narrow+2:end) = ...
182     global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
183 v_newplot(m_wide+2:end-1,2:end) = ...
184     global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
185 v_newplot(1:m_wide, 1:N_narrow) = filler;
186
187 % Transformation from dimensionless to regular
188 v_newplot = v_newplot*u_in_true;
189
190
191 % Create a mesh for the plotting
192 [xv_plot,yv_plot] = meshgrid(...
193     x_0:del_x_true:x_N, ...
194     y_0:del_y_true:y_M);
195
196
197 f2 = figure('units','normalized','outerposition',[0 0 1 1]);

```

```

198
199
200 subplot(2,2,1);
201 f = surf(xv_plot,yv_plot,v_circplot);           % surf(x,y,z)
202 s = sprintf('Plot of  $v_{\text{circ}}$  after %d iterations', it );
203 f = title(s);
204 set(f, 'interpreter', 'latex', 'fontsize', 16)
205 set(gca,'TickLabelInterpreter','latex')
206 xlabel('$x$-direction [m]', 'interpreter', 'latex')
207 ylabel('$y$-direction [m]', 'interpreter', 'latex')
208 zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
209 ztickformat('%0.2f')
210
211
212 subplot(2,2,3);
213 f = surf(xv_plot,yv_plot,v_starplot);           % surf(x,y,z)
214 s = sprintf('Plot of  $v_{\text{star}}$  after %d iterations', it );
215 f = title(s);
216 set(f, 'interpreter', 'latex', 'fontsize', 16)
217 set(gca,'TickLabelInterpreter','latex')
218 xlabel('$x$-direction [m]', 'interpreter', 'latex')
219 ylabel('$y$-direction [m]', 'interpreter', 'latex')
220 zlabel('Velocity, [m/s]', 'interpreter', 'latex')
221 ztickformat('%0.2f')
222
223
224 subplot(2,2,2);
225 f = surf(xv_plot,yv_plot,v_corrplot);           % surf(x,y,z)
226 s = sprintf('Plot of  $v_{\text{corr}}$  after %d iterations', it );
227 f = title(s);
228 set(f, 'interpreter', 'latex', 'fontsize', 16)
229 set(gca,'TickLabelInterpreter','latex')
230 xlabel('$x$-direction [m]', 'interpreter', 'latex')
231 ylabel('$y$-direction [m]', 'interpreter', 'latex')
232 zlabel('Velocity, [m/s]', 'interpreter', 'latex')
233 ztickformat('%0.2f')
234
235
236 subplot(2,2,4);
237 f = surf(xv_plot,yv_plot,v_newplot);
238 s = sprintf('Plot of  $v_{\text{new}}$  after %d iterations', it );
239 f = title(s);
240 set(f, 'interpreter', 'latex', 'fontsize', 16)
241 set(gca,'TickLabelInterpreter','latex')
242 xlabel('$x$-direction [m]', 'interpreter', 'latex')
243 ylabel('$y$-direction [m]', 'interpreter', 'latex')
244 zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
245 ztickformat('%0.2f')
246
247
248 axis tight manual % this ensures that getframe() returns a consistent size
249 filename = 'itdev_vintermediates.gif';
250 % Capture the plot as an image
251 frame = getframe(f2);
252 im = frame2im(frame);
253 [imind,cm] = rgb2ind(im,256);
254 % Write to the GIF File
255 if it == 1
256     imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
257 else
258     imwrite(imind,cm,filename,'gif','WriteMode','append');
259 end
260 close all

```

E.5.1.12 isWide.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %           Function checkin if a node is in the wide section or not           %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function res = isWide(a, N_narrow, N_wide, M_wide)
5     res = ones(1,length(a))*false;
6     for j = 1:length(a)
7         i = a(j);
8         rownumber = getRowNumber(i, N_wide, M_wide, N_narrow + N_wide);
9         if rownumber <= M_wide || i - M_wide*N_wide - ...
10            (rownumber-M_wide-1)*(N_narrow + N_wide) > N_narrow

```

```

11         res(j) = true;
12     end %if
13 end %for
14 end %function

```

E.5.1.13 getRowNumber.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %           Function giving the row number of a node           %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 % getRowNumber.m returns the row number of an arbitrary computational point
5 % in the domain defined by N and M in the main BFC_globaldomain_spring.m
6 function rownumber = getRowNumber(a, N_wide, M_wide, N_total)
7 rownumber = zeros(length(a),1);
8     for j = 1:length(a)
9         i = a(j);
10        if i <= N_wide*M_wide
11            rownumber(j) = floor((N_wide+i-1)/N_wide);
12        elseif i > N_wide*M_wide
13            rownumber(j) = M_wide + floor((i-N_wide*M_wide-1)/N_total)+1;
14        end %if
15    end %for
16 end %function

```

E.5.1.14 getRowUnder.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %           Function giving the row number of a node below itself %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 % getRowUnder.m returns the index of the point directly below itself.
5 function index = getRowUnder(i, N_wide, M_wide, N_total)
6     index = zeros(1, length(i));
7     for j = 1:length(i)
8         if i(j) <= N_wide*M_wide
9             index(j) = i(j)-N_wide;
10        elseif i(j) > N_wide*M_wide
11            index(j) = i(j)-N_total;
12        end %if
13    end %for
14 end %function

```

E.5.1.15 getRowOver.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %           Function giving the row number of a node above itself %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 % getRowOver.m returns the index of the point directly above itself.
5 function index = getRowOver(i, N_wide, M_wide, N_total)
6     index = zeros(1, length(i));
7     for j = 1:length(i)
8         if i(j) <= N_wide*(M_wide-1)
9             index(j) = i(j)+N_wide;
10        elseif i(j) > N_wide*(M_wide-1)
11            index(j) = i(j)+N_total;
12        end %if
13    end %for
14 end %function

```

E.5.1.16 global2matrix.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %           Function transforming a globally indexed vector into a matrix %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function [matrix] = global2matrix(glob, N, M)
5     for j = 1:M % "down" % the rest of the points are zero
6         for i = 1:N % "left"
7             matrix(j,i) = glob((j-1)*N + i);
8         end % for
9     end % for
10 end %function

```

E.5.2 Parabolic Inlet Velocity Profile

The code `channel_BFS_parabolic.m` solves the two dimensional backwards facing step problem. The code `BFS_u_velocity_parabolic.m` contains the calculations of the Momentum equation for the u -velocity component. The code `BFS_v_velocity_parabolic.m` contains the calculations of the Momentum equation for the v -velocity component. The code `BFS_pressurecorrection_parabolic.m` contains the calculations of the Momentum equation for the u -velocity component. The code `plotColoredQuiver_parabolic.m` plots the velocity quiver plots with the contour plot for background colour.

The same helper functions `isWide.m`, `getRowNumber.m`, `getRowUnder.m`, `getRowOver.m` and `global2matrix.m` as given in section E.5.1 are used.

E.5.2.1 `channel_BFS_parabolic.m`

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  % Two dimensional fluid flow over a backwards facing step, dimensionless %
3  %           Model adjusted to Reynolds number comparison                %
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  close all
6  clear
7  clc
8  tic
9  warning on
10
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12 %% Solver specs
13 maxits = 50000; % Maximum number of iterations, stop if iterations exceed
14 % Choose which inlet profile to use
15 run('inletprofileRe400.m')
16
17 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
18 %% System specifications
19 % Specify number of narrow points, leave the rest
20
21 N_narrow = 10; % Number of scalar nodal points in narrow section in x-dir.
22 M_narrow = 10; % Number of scalar nodal points in narrow section in x-dir.
23
24 l = 5; % Narrow channel length
25 h = 1; % Narrow channel height
26 L = 30; % Wide channel length
27 H = 1; % Wide channel height
28
29 L_total = l + L; % Total channel length
30 H_total = h + H; % Total channel height
31
32 x_0 = 0; % Defining the domain using x and y
33 x_N = L_total;
34 y_0 = 0;
35 y_M = H_total;
36
37 N_wide = N_narrow*L/l; % # scalar nodal points in wide section in x-dir.
38 M_wide = M_narrow*H/h; % # scalar nodal points in wide section in y-dir.
39
40 % For extension to the wide channel the number of nodes in the narrow
41 % section needs to meet these criteria:
42 if floor(N_narrow)~= N_narrow || floor(N_wide)~= N_wide || ...
43     floor(M_narrow)~= M_narrow || floor(M_wide)~= M_wide
44     msg = 'Points don''t match dimensions';
45     error(msg)
46 end %if
47 N_total = N_narrow + N_wide;% Total # of scalar nodal points in x-direction
48 M_total = M_narrow + M_wide;% Total # of scalar nodal points in y-direction
49
50 m_total = M_total - 1; % Total number of y-velocity nodes in y-direction
51 m_wide = M_wide;% Number of y-velocity nodes in y-direction in wide section
52 m_narrow = M_narrow - 1;% # of y-velocity nodes in y-dir. in narrow section
53
54 % Total number of computational points in the domain ...
55 totalpoints = N_narrow*M_narrow + N_wide*M_total; % ... for u and P
56 totalpoints_v = N_narrow*m_narrow + N_wide*m_total; % ... for v

```



```

57
58 D_hyd = 2*h; % Hydraulic diameter
59 mu_true = 8.90 * 10^-4; % Viscosity of water
60
61 del_z_true = 1; % System depth
62 del_x_true = L_total/N_total; % Control volume width
63 del_y_true = H_total/M_total; % Control volume height
64 A_x_true = del_y_true*del_z_true; % Cross-sectional area in x-direction
65 A_y_true = del_x_true*del_z_true; % Cross-sectional area in y-direction
66
67 rho_true = 997; % Density of water
68 u_in_true = u_bulk; % Inlet u-velocity
69
70 g_x = 0; % No gravitation
71 g_y = 0; % No gravitation
72
73 Re = rho_true*D_hyd*u_in_true/mu_true; % Reynolds number
74
75 p_atm = 101325; % Atmospheric pressure at outlet
76 p_out_tilde = 0; % Adjusted pressure
77 p_out = ones(1,M_total)*p_out_tilde; % Outlet pressure profile
78
79 alpha_u = 0.01; % Under-relaxation factor for u
80 alpha_v = 0.01; % Under-relaxation factor for v
81 alpha_p = 0.02; % Under-relaxation factor for p
82 alpha_u = 0.005; % Under-relaxation factor for u
83 alpha_v = 0.005; % Under-relaxation factor for v
84 alpha_p = 0.01; % Under-relaxation factor for p
85
86 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
87 %% Dimensionless parameters
88 mu = 1; % Dimensionless viscosity
89 rho = 1; % Dimensionless density
90 del_x = del_x_true/h; % Dimensionless control volume width
91 del_y = del_y_true/h; % Dimensionless control volume height
92 A_x = A_x_true/h^2; % Dimensionless cross-sectional area in x-direction
93 A_y = A_y_true/h^2; % Dimensionless cross-sectional area in y-direction
94 D_x = 2/Re*mu/del_x; % Dimensionless diffusion conductance in x-direction
95 D_y = 2/Re*mu/del_y; % Dimensionless diffusion conductance in y-direction
96 u_in = u_in/u_in_true; % Inlet u-velocity
97 u_bulk_dimless = u_bulk/u_in_true; % Bulk inlet velocity (which is 1)
98 v_in = 0; % Inlet v-velocity
99 u_guess = u_max; % Initial guess for u-velocity
100 v_guess = 0.0; % Initial guess for v-velocity
101 p_guess = 0/(rho_true*u_in_true^2); % Initial guess for pressure
102
103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104 %% Initialisation of p
105 % Filling in initial pressure vector with the linear profile.
106 % This section is set up for if gravity is added, but could be more compact
107 % if the option to add gravity was not there.
108
109 p_circ_y_wide = linspace(p_guess, p_guess+rho*g_y*H_total,M_total);
110 p_circ_carthesian_wide = zeros(M_total,N_wide);
111 for j = 1:M_total
112     for i = 1:N_wide
113         p_circ_carthesian_wide(j,i) = p_circ_y_wide(j);
114     end %for
115 end %for
116
117 p_circ_y_narrow = p_circ_y_wide(M_wide+1:end);
118 p_circ_carthesian_narrow = zeros(M_narrow,N_narrow);
119 for j = 1:M_narrow
120     for i = 1:N_narrow
121         p_circ_carthesian_narrow(j,i) = p_circ_y_narrow(j);
122     end %for
123 end %for
124
125 filler = zeros(M_wide, N_narrow);
126 p_circ_carthesian = [[filler; p_circ_carthesian_narrow] ...
127     p_circ_carthesian_wide ];
128 p_circ_carthesian = flip(p_circ_carthesian,1);
129
130 p_circ = p_circ_carthesian(1,:); % Take the first vector
131
132 for i = 2:M_total

```

```

133     row = p_circ_carthesian(i);
134     if i <= M_narrow                                     % Take whole row
135         p_circ = [p_circ, p_circ_carthesian(i,:)];
136     else                                               % Take part of the row
137         p_circ = [p_circ, p_circ_carthesian(i,N_narrow+1:N_total)];
138     end %if
139 end %for
140
141
142 %% Initialisation of u and v
143
144 u_circ = ones(totalpoints,1)*u_guess; % Fill in guess in the initial vector
145 for i = 1:totalpoints
146     if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
147         u_circ(i) = u_guess*(M_narrow/M_total);
148     end %if
149 end %for
150
151
152 v_circ = ones(totalpoints_v,1)*v_guess; % Fill in guess in the initial vec.
153 for i = 1:totalpoints_v
154     if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
155         v_circ(i) = v_guess*(m_narrow/m_total);
156     end %if
157 end %for
158
159
160 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
161 %% Initialisation of solution vectors
162 p_new = zeros(1, totalpoints);                               % New pressure
163
164 u_corr = zeros(1, totalpoints);                             % u-velocity correction
165 u_new = zeros(1, totalpoints);                               % New u-velocity
166
167 v_corr = zeros(1, totalpoints_v);                           % v-velocity correction
168 v_new = zeros(1, totalpoints_v);                             % New v-velocity
169
170 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
171 %% While loop
172 conv = 0;                                                    % 0 is not converged, 1 when converged
173 it = 1;                                                       % The current iteration
174
175 while conv == 0
176     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
177     %% Calculate velocities and pressure correction
178     % Run the scripts:
179     % Velocities
180     BFS_u_velocity_parabolic
181     BFS_v_velocity_parabolic
182
183     % Pressure correction
184     BFS_pressurecorrection_parabolic
185
186     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
187     %% Velocity correction
188
189     startCorr = 1;
190     for j = startCorr:totalpoints
191         if ( i <= N_wide*M_wide && mod(i, N_wide) == 0 ) ... % Below step
192             || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0 )
193             % Eastern boundary : eastern pressure is known, no press. corr.
194             u_corr(j) = - A_x/au(j)*(-p_corr(j));
195         else
196             u_corr(j) = - A_x/au(j)*(p_corr(j+1)-p_corr(j));
197         end % if
198     end %for
199
200     for k = startCorr:totalpoints_v
201         v_corr(k) = - A_y/av(k)*...
202             (p_corr(getRowOver(k, N_wide, M_wide, N_total))-p_corr(k));
203     end %for
204
205     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
206     %% Under-relaxation
207
208     u_new = alpha_u*(u_star + u_corr') + (1-alpha_u)*u_circ;

```

```

209
210     v_new = alpha_v*(v_star + v_corr') + (1-alpha_v)*v_circ;
211
212     p_new = p_circ + alpha_p* p_corr';
213
214     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
215     %% Check convergence
216     % Make sure there are no mistakes in the matrix operations above
217     if ~isvector(u_new) || ~isvector(p_new) || ~isvector(p_new)
218         fprintf('u_new - %dx%d\n',size(u_new,1),size(u_new,2))
219         fprintf('v_new - %dx%d\n',size(v_new,1),size(v_new,2))
220         fprintf('p_new - %dx%d\n',size(p_new,1),size(p_new,2))
221         error('Matrix addition gone wrong')
222     end
223
224     if isnan(rcond(U)) || isnan(rcond(V)) || isnan(rcond(T))
225 %       clc % Remove if warnings are desired
226         fprintf('Stopped due to singularity in matrix\n')
227         fprintf('RCOND u-velocity: %e \nRCOND v-velocity: %e \n',...
228             rcond(U), rcond(V))
229         fprintf('RCOND pressure correction: %e\n',rcond(T))
230         fprintf('Problem occured after %d iterations\n', it)
231         toc
232         return
233     end %if
234
235     c1 = 1/u_bulk_dimless*sqrt((U*u_star-bu')*(U*u_star-bu')); % residuals
236     c2 = 1/u_bulk_dimless*sqrt((V*v_star-bv')*(V*v_star-bv')); % residuals
237     c3 = abs(sum(beta)); % continuity fulfilled
238     c4 = 1/u_bulk_dimless*max(abs(u_circ - u_star)) ; % change from last iteration
239     c5 = 1/u_bulk_dimless*max(abs(v_circ - v_star)) ; % change from last iteration
240
241     c1_lim = 10^-8; % Limits
242     c2_lim = 10^-8;
243     c3_lim = 10^-10;
244     c4_lim = 10^-8;
245     c5_lim = 10^-8;
246
247     c1_diff = c1-c1_lim; % How far away from convergence
248     c2_diff = c2-c2_lim;
249     c3_diff = c3-c3_lim;
250     c4_diff = c4-c4_lim;
251     c5_diff = c5-c5_lim;
252
253
254     if (c1 < c1_lim) && (c2 < c2_lim) && (c3 < c3_lim) && (c4 < c4_lim) ...
255         && (c5 < c5_lim) || (it == maxits)
256         conv = 1; % Converged
257         if (it == maxits)
258             fprintf('Stopped at max iterations (%d)\n',it);
259         else
260             fprintf('Solution converged after %d iterations\n',it);
261         end %if
262
263         fprintf('c1\tMomentum residual u\t\t%.2e\tLimit: %.2e\n',...
264             c1,c1_lim);
265         fprintf('c2\tMomentum residual v\t\t%.2e\tLimit: %.2e\n',...
266             c2,c2_lim);
267         fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',...
268             c3,c3_lim);
269         fprintf('c4\tDiff. last iteration u\t%.2e\tLimit: %.2e\n',...
270             c4,c4_lim);
271         fprintf('c5\tDiff. last iteration v\t%.2e\tLimit: %.2e\n',...
272             c5,c5_lim);
273
274         if max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c1_diff
275             fprintf('Limiting criteria is c1\tMomentum residual u\n')
276         elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c2_diff
277             fprintf('Limiting criteria is c2\tMomentum residual v\n')
278         elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c3_diff
279             fprintf('Limiting criteria is c3\tPressure correction\n')
280         elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c4_diff
281             fprintf('Limiting criteria is c4\tDiff. last iteration u\n')
282         elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c5_diff
283             fprintf('Limiting criteria is c5\tDiff. last iteration v\n')
284         end %if

```

```

285
286     plotColoredQuiver_parabolic
287
288     else
289
290         u_circ = u_new;           % Not converged, updated variables
291         v_circ = v_new;           % Not converged, updated variables
292         p_circ = p_new;           % Not converged, updated variables
293
294         it = it + 1; % Update number of iterations
295     end % if
296 end %while
297 toc

```

E.5.2.2 BFS_u_velocity_parabolic.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %           u-velocity script for the BFS model                               %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  U = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
6  bu = zeros(1, totalpoints);         % Initialisation of source term vector
7
8  F_xe = zeros(1, totalpoints);       % Initialisation of convective mass fluxes
9  F_xw = zeros(1, totalpoints);
10 F_xn = zeros(1, totalpoints);
11 F_xs = zeros(1, totalpoints);
12
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 %% Generation of F_x, Convective mass fluxes
15
16
17 for i = 1:totalpoints
18
19     etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )... % below step
20     || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
21     wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
22     ntest = totalpoints - N_total < i && i <= totalpoints ;
23     wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
24     stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
25     || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
26     scorner = i == N_wide*M_wide + N_narrow; % Only the corner value
27
28
29 % Northeastern corner
30 if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
31     F_xe(i) = rho/2*(u_circ(i)+ u_circ(i-1));
32     F_xn(i) = 0;
33
34     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
35     F_xs(i) = rho/2*v_circ(i-N_total);
36
37 % Southeastern corner
38 elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
39     F_xe(i) = rho/2*(u_circ(i)+u_circ(i-1));
40     F_xs(i) = 0;
41
42     F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
43     F_xn(i) = rho/2*v_circ(i);
44
45 % Northwestern corner
46 elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner
47     F_xw(i) = rho/2*(...
48     u_in(getRowNumber(i, N_wide, M_wide, N_total))+u_circ(i));
49     F_xn(i) = 0;
50
51     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
52     F_xs(i) = rho/2*(v_circ(i-N_total) + v_circ(i-N_total+1));
53
54 % Southwestern corner at inlet
55 elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner
56     F_xw(i) = rho/2*(...
57     u_in(getRowNumber(i, N_wide, M_wide, N_total))+u_circ(i));
58     F_xs(i) = 0;
59
60     F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));

```

```

61     F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
62
63     % Southwestern corner at step
64     elseif ~etest && ~wtest && ~ntest && stest && wwall && ~scorner
65         F_xw(i) = rho/2*(0 + u_circ(i));
66         F_xs(i) = 0;
67
68         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
69         F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
70
71     % At corner
72     elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner
73         F_xs(i) = rho/2*(0 + ...
74             v_circ(getRowUnder(i, N_wide, M_wide, N_total)+1));
75         F_xs(i)= 0;
76
77         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
78         F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
79         F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
80
81     % At eastern boundary (x = L)
82     elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner
83         F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));
84
85         F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
86         F_xn(i) = rho/2*v_circ(i);
87         F_xs(i) = rho/2*v_circ(getRowUnder(i, N_wide, M_wide, N_total));
88
89     % At western boundary (x = 0)
90     elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner
91         F_xw(i) = rho/2*(...
92             u_in(getRowNumber(i, N_wide, M_wide, N_total))+u_circ(i));
93
94         F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
95         F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
96         F_xs(i) = rho/2*(...
97             v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
98             v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
99
100    % At western wall at step
101    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner
102        F_xw(i) = rho/2*(0+u_circ(i));
103
104        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
105        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
106        F_xs(i) = rho/2*(...
107            v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
108            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
109
110
111
112    % At northern boundary (y = h)
113    elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
114        F_xn(i) = 0;
115
116        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
117        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
118        F_xs(i) = rho/2*(...
119            v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
120            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
121
122
123
124    % At southern boundary (y = 0)
125    elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
126        F_xs(i) = 0;
127
128        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
129        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
130        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
131
132    % Not at any boundary
133    else
134        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
135        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
136        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

```

```

136     F_xs(i) = rho/2*(...
137         v_circ( getRowUnder(i, N_wide, M_wide, N_total) ) +...
138         v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 ) );
139
140
141     end % if
142     etest = false;
143     wtest = false;
144     wwall = false;
145     ntest = false;
146     stest = false;
147     scorner = false;
148 end %for
149
150
151
152 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
153 %% u-velocity
154
155 for i = 1:totalpoints                % Global indexing system
156
157     etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )... % below step
158         || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
159     wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
160     ntest = totalpoints - N_total < i && i <= totalpoints ;
161     wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
162     stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
163         || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
164     scorner = i == N_wide*M_wide + N_narrow; % Only the corner value
165
166
167
168 % Northeastern corner
169 if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
170
171     bu(i) = -(p_out(end)-p_circ(i))*A_x;
172
173     % At eastern boundary (x = L)
174     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
175     eP_coeff = F_xe(i)*A_x;
176
177     % At northern boundary
178     nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y) + 2*D_y*A_y;
179
180     W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
181     wP_coeff = -W_coeff - F_xw(i)*A_x;
182     U(i, i-1) = W_coeff;
183
184     S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
185     sP_coeff = -S_coeff - F_xs(i)*A_y;
186     U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
187
188
189 % Southeastern corner
190 elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
191
192     bu(i) = -(p_out(1)-p_circ(i))*A_x;
193
194     % At eastern boundary (x = L)
195     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
196     eP_coeff = F_xe(i)*A_x;
197
198     % At southern boundary (y = 0)
199     sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0) + 2*D_y*A_y;
200
201     W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
202     wP_coeff = -W_coeff - F_xw(i)*A_x;
203     U(i, i-1) = W_coeff;
204
205     N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
206     nP_coeff = -N_coeff + F_xn(i)*A_y;
207     U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
208
209
210 % Northwestern corner
211 elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner

```

```

212
213     bu(i) = -(p_circ(i+1)-p_circ(i))*A_x +(max(F_xw(i)*A_x,0)...
214             + D_x*A_x)*u_in(getRowNumber(i, N_wide, M_wide, N_total));
215
216     % At western boundary (x = 0)
217     wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
218
219     % At northern boundary
220     nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;
221
222     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
223     eP_coeff = -E_coeff + F_xe(i)*A_x;
224     U(i, i+1) = E_coeff;
225
226     S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
227     sP_coeff = -S_coeff - F_xs(i)*A_y;
228     U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
229
230
231     % Southwestern corner at inlet
232     elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner
233
234         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x+(max(F_xw(i)*A_x,0) ...
235             + D_x*A_x)*u_in(getRowNumber(i, N_wide, M_wide, N_total));
236
237         % At western boundary (x = 0)
238         wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
239
240         % At southern boundary (y = 0)
241         sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
242
243         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
244         eP_coeff = -E_coeff + F_xe(i)*A_x;
245         U(i, i+1) = E_coeff;
246
247         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
248         nP_coeff = -N_coeff + F_xn(i)*A_y;
249         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
250
251
252     % Southwestern corner at step
253     elseif ~etest && ~wtest && ~ntest && stest && wwall && ~scorner
254
255         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
256             +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;
257
258         % At western boundary (x = 0)
259         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
260         wP_coeff = -W_coeff - F_xw(i)*A_x;
261
262         % At southern boundary (y = 0)
263         S_coeff = -max(F_xs(i)*A_y,0) - 2*D_y*A_y;
264         sP_coeff = -S_coeff -F_xs(i)*A_y;
265
266         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
267         eP_coeff = -E_coeff + F_xe(i)*A_x;
268         U(i, i+1) = E_coeff;
269
270         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
271         nP_coeff = -N_coeff + F_xn(i)*A_y;
272         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
273
274
275     % At corner
276     elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner
277
278         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
279
280         % At southern boundary (y = 0)
281         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
282         sP_coeff = -S_coeff - F_xs(i)*A_y;
283
284
285         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
286         eP_coeff = -E_coeff + F_xe(i)*A_x;
287         U(i, i+1) = E_coeff;

```

```

288
289     W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
290     wP_coeff = -W_coeff - F_xw(i)*A_x;
291     U(i, i-1) = W_coeff;
292
293     N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
294     nP_coeff = -N_coeff + F_xn(i)*A_y;
295     U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
296
297
298     % At eastern boundary (x = L)
299     elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner
300
301         bu(i) = -(p_out(1)-p_circ(i))*A_x;
302
303         % At eastern boundary (x = L)
304         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
305         eP_coeff = F_xe(i)*A_x;
306
307         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
308         wP_coeff = -W_coeff - F_xw(i)*A_x;
309         U(i, i-1) = W_coeff;
310
311         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
312         nP_coeff = -N_coeff + F_xn(i)*A_y;
313         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
314
315         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
316         sP_coeff = -S_coeff - F_xs(i)*A_y;
317         U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
318
319
320     % At western boundary (x = 0)
321     elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner
322
323         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x +(max(F_xw(i)*A_x,0) ...
324             + D_x*A_x)*u_in(getRowNumber(i, N_wide, M_wide, N_total));
325
326         % At western boundary (x = 0)
327         wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;
328
329         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
330         eP_coeff = -E_coeff + F_xe(i)*A_x;
331         U(i, i+1) = E_coeff;
332
333         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
334         nP_coeff = -N_coeff + F_xn(i)*A_y;
335         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
336
337         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
338         sP_coeff = -S_coeff - F_xs(i)*A_y;
339         U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
340
341
342     % At western wall
343     elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner
344
345         bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
346             +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;
347
348         % At western boundary (x = 0)
349         W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
350         wP_coeff = -W_coeff - F_xw(i)*A_x;
351
352         E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
353         eP_coeff = -E_coeff + F_xe(i)*A_x;
354         U(i, i+1) = E_coeff;
355
356         N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
357         nP_coeff = -N_coeff + F_xn(i)*A_y;
358         U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
359
360         S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
361         sP_coeff = -S_coeff - F_xs(i)*A_y;
362         U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
363

```



```

364
365 % At northern boundary (y = h)
366 elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
367
368     bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
369
370 % At northern boundary
371 nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;
372
373 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
374 eP_coeff = -E_coeff + F_xe(i)*A_x;
375 U(i, i+1) = E_coeff;
376
377 W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
378 wP_coeff = -W_coeff - F_xw(i)*A_x;
379 U(i, i-1) = W_coeff;
380
381 S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
382 sP_coeff = -S_coeff - F_xs(i)*A_y;
383 U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
384
385
386 % At southern boundary (y = 0)
387 elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
388
389     bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
390
391 % At southern boundary (y = 0)
392 sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
393
394 E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
395 eP_coeff = -E_coeff + F_xe(i)*A_x;
396 U(i, i+1) = E_coeff;
397
398 W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
399 wP_coeff = -W_coeff - F_xw(i)*A_x;
400 U(i, i-1) = W_coeff;
401
402 N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
403 nP_coeff = -N_coeff + F_xn(i)*A_y;
404 U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
405
406
407 %Not at any boundary
408 else
409
410     bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
411     E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
412     eP_coeff = -E_coeff + F_xe(i)*A_x;
413     U(i, i+1) = E_coeff;
414
415     W_coeff = -max(F_xw(i)*A_x,0) - D_x*A_x;
416     wP_coeff = -W_coeff - F_xw(i)*A_x;
417     U(i, i-1) = W_coeff;
418
419     N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
420     nP_coeff = -N_coeff + F_xn(i)*A_y;
421     U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
422
423     S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
424     sP_coeff = -S_coeff - F_xs(i)*A_y;
425     U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
426
427 end % if
428
429 % Filling in the rest of the matrix, adding all point coefficients
430 U(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
431
432 etest = false;
433 wtest = false;
434 ntest = false;
435 stest = false;
436 wwall = false;
437
438 end %for
439 u_star = U\bu'; % Matrix inversion

```

E.5.2.3 BFS_v_velocity_parabolic.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %           v-velocity script for the BFS model           %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  V = zeros(totalpoints_v, totalpoints_v); % Initialisation of coeff. matrix
6  bv = zeros(1, totalpoints_v);          % Initialisation of source term vector
7
8  F_ye = zeros(1, totalpoints_v); % Initialisation of convective mass fluxes
9  F_yw = zeros(1, totalpoints_v);
10 F_yn = zeros(1, totalpoints_v);
11 F_ys = zeros(1, totalpoints_v);
12
13
14
15
16 %% Generation of F_y, Convective mass fluxes
17
18
19 for i = 1:totalpoints_v % Global indexing system
20
21     etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ... % below step
22         || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0);
23     wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
24     ntest = totalpoints_v - N_total < i && i <= totalpoints_v ;
25     wwall = i <= N_wide*m_wide && mod(i-1, N_wide) == 0; %
26     stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
27         || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow);
28     wcorner = i == N_wide*(m_wide-1) + 1; % Only the corner value
29
30
31
32 % Northwestern corner
33 if wtest && ntest && ~stest && ~wwall && ~wcorner
34     F_yw(i) = rho/2*(u_in(getRowNumber(i, N_wide, M_wide, N_total))...
35         +u_in(getRowNumber(i, N_wide, M_wide, N_total)));
36     F_yn(i) = rho/2*v_circ(i);
37
38     F_ye(i) = rho/2*(u_circ(i) + ...
39         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
40     F_ys(i) = rho/2*(v_circ(i) + ...
41         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
42
43 % Southwestern corner at inlet
44 elseif wtest && ~ntest && stest && ~wwall && ~wcorner
45     F_yw(i) = rho*u_in(getRowNumber(i, N_wide, M_wide, N_total));
46     F_ys(i) = rho/2*v_circ(i);
47
48     F_ye(i) = rho/2*(u_circ(i) + ...
49         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
50     F_yn(i) = rho/2*(v_circ(i) + ...
51         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
52
53
54 % Southwestern corner at step
55 elseif ~wtest && ~ntest && stest && wwall && ~wcorner
56     F_yw(i) = rho*0;
57     F_ys(i) = rho/2*v_circ(i);
58
59     F_ye(i) = rho/2*(u_circ(i) + ...
60         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
61     F_yn(i) = rho/2*(v_circ(i) + ...
62         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
63
64
65 % At western boundary (x = 0)
66 elseif wtest && ~ntest && ~stest && ~wwall && ~wcorner
67     F_yw(i) = rho*u_in(getRowNumber(i, N_wide, M_wide, N_total));
68
69     F_ye(i) = rho/2*(u_circ(i) + ...
70         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
71     F_yn(i) = rho/2*(v_circ(i) + ...
72         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
73     F_ys(i) = rho/2*(v_circ(i) + ...
74         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

```

```

75
76
77 % At western wall
78 elseif ~wtest && ~ntest && ~stest && wwall && ~wcorner
79     F_yw(i) = rho*0;
80
81     F_ye(i) = rho/2*(u_circ(i) + ...
82         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
83     F_yn(i) = rho/2*(v_circ(i) + ...
84         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
85     F_ys(i) = rho/2*(v_circ(i) + ...
86         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
87
88
89 % At corner, right point from the corner
90 elseif ~wtest && ~ntest && ~stest && wwall && wcorner
91     F_yw(i)= 0;
92
93     F_ye(i) = rho/2*(u_circ(i) + ...
94         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
95     F_yn(i) = rho/2*(v_circ(i) + ...
96         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
97     F_ys(i) = rho/2*(v_circ(i) + ...
98         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
99
100 % At northern boundary (y = h)
101 elseif ~wtest && ntest && ~stest && ~wwall && ~wcorner
102     F_yn(i) = rho/2*v_circ(i);
103
104     F_ye(i) = rho/2*(u_circ(i) + ...
105         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
106     F_yw(i) = rho/2*(u_circ(i-1) + ...
107         u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
108     F_ys(i) = rho/2*(v_circ(i) + ...
109         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
110
111
112 % At southern boundary (y = 0)
113 elseif ~wtest && ~ntest && stest && ~wwall && ~wcorner
114     F_ys(i) = rho/2*v_circ(i);
115
116     F_ye(i) = rho/2*(u_circ(i) + ...
117         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
118     F_yw(i) = rho/2*(u_circ(i-1) + ...
119         u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
120     F_yn(i) = rho/2*(v_circ(i) + ...
121         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
122
123
124 %Not at any boundary, including eastern boundary
125 else
126     F_ye(i) = rho/2*(u_circ(i) + ...
127         u_circ(getRowOver(i, N_wide, M_wide, N_total)));
128     F_yw(i) = rho/2*(u_circ(i-1) + ...
129         u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
130
131     F_yn(i) = rho/2*(v_circ(i) + ...
132         v_circ(getRowOver(i, N_wide, M_wide, N_total)));
133     F_ys(i) = rho/2*(v_circ(i) + ...
134         v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
135
136 end % if
137 etest = false;
138 wtest = false;
139 ntest = false;
140 stest = false;
141 wwall = false;
142 wcorner = false;
143
144 end % for
145
146 %% v-velocity
147
148 for i = 1:totalpoints_v % Global indexing system
149
150     etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ... % below step

```

```

151         || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0);
152 wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
153 ntest = totalpoints_v - N_total < i && i <= totalpoints_v ;
154 wwall = i <= N_wide*m_wide && mod(i-1, N_wide) == 0; %
155 stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
156         || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow);
157 wcorner = i == N_wide*(m_wide-1) + 1; % Only the corner value
158
159
160
161
162 % Northeastern corner
163 if etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner
164
165     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
166             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
167
168     % At eastern boundary (x = L)
169     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
170     eP_coeff = F_ye(i)*A_x;
171
172
173     % At northern boundary
174     nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y;
175
176     W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
177     wP_coeff = -W_coeff - F_yw(i)*A_x;
178     V(i, i-1) = W_coeff;
179
180     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
181     sP_coeff = -S_coeff - F_ys(i)*A_y;
182     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
183
184 % Southeastern corner
185 elseif etest && ~wtest && ~ntest && stest && ~wwall && ~wcorner
186     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
187             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
188
189     % At eastern boundary (x = L)
190     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
191     eP_coeff = F_ye(i)*A_x;
192
193     % At southern boundary (y = 0),
194     sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
195
196     W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
197     wP_coeff = -W_coeff - F_yw(i)*A_x;
198     V(i, i-1) = W_coeff;
199
200     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
201     nP_coeff = -N_coeff + F_yn(i)*A_y;
202     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
203
204
205 % Northwestern corner
206 elseif ~etest && wtest && ntest && ~stest && ~wwall && ~wcorner
207     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))-...
208             p_circ(i))*A_y + rho*g_y*del_y*A_y;
209
210     % At western boundary (x = 0)
211     wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
212
213     % At northern boundary
214     nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;
215
216     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
217     eP_coeff = -E_coeff + F_ye(i)*A_x;
218     V(i, i+1) = E_coeff;
219
220     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
221     sP_coeff = -S_coeff - F_ys(i)*A_y;
222     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
223
224 % Southwestern corner at inlet
225 elseif ~etest && wtest && ~ntest && stest && ~wwall && ~wcorner
226

```

```

227     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
228         -p_circ(i))*A_y + rho*g_y*del_y*A_y;
229
230     % At western boundary (x = 0)
231     wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
232
233     % At southern boundary (y = 0),
234     sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
235
236     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
237     eP_coeff = -E_coeff + F_ye(i)*A_x;
238     V(i, i+1) = E_coeff;
239
240     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
241     nP_coeff = -N_coeff + F_yn(i)*A_y;
242     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
243
244     % Southwestern corner at step
245     elseif ~etest && ~wtest && ~ntest && stest && wwall && ~wcorner
246
247     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
248         -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
249         0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);
250
251     % At western boundary (x = 0)
252     W_coeff = -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
253     wP_coeff = -W_coeff - F_yw(i)*A_x;
254
255     % At southern boundary (y = 0),
256     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
257     sP_coeff = -S_coeff - F_ys(i)*A_y;
258
259     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
260     eP_coeff = -E_coeff + F_ye(i)*A_x;
261     V(i, i+1) = E_coeff;
262
263     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
264     nP_coeff = -N_coeff + F_yn(i)*A_y;
265     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
266
267     % At eastern boundary (x = L)
268     elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~wcorner
269
270     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
271         -p_circ(i))*A_y + rho*g_y*del_y*A_y;
272
273     % At eastern boundary (x = L)
274     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
275     eP_coeff = F_ye(i)*A_x;
276
277     W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
278     wP_coeff = -W_coeff - F_yw(i)*A_x;
279     V(i, i-1) = W_coeff;
280
281     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
282     nP_coeff = -N_coeff + F_yn(i)*A_y;
283     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
284
285     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
286     sP_coeff = -S_coeff - F_ys(i)*A_y;
287     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
288
289     % At western boundary (x = 0)
290     elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~wcorner
291
292     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
293         -p_circ(i))*A_y + rho*g_y*del_y*A_y;
294
295     % At western boundary (x = 0)
296     wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;
297
298     E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
299     eP_coeff = -E_coeff + F_ye(i)*A_x;
300     V(i, i+1) = E_coeff;
301
302     N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;

```

```

303     nP_coeff = -N_coeff + F_yn(i)*A_y;
304     V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
305
306     S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
307     sP_coeff = -S_coeff - F_ys(i)*A_y;
308     V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
309
310 % At west wall (x = 0) [EXCLUDED CORNER]
311 elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~wcorner
312
313     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
314             -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
315             0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);
316
317 % At western boundary (x = 0)
318 W_coeff = -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
319 wP_coeff = -W_coeff - F_yw(i)*A_x;
320
321 E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
322 eP_coeff = -E_coeff + F_ye(i)*A_x;
323 V(i, i+1) = E_coeff;
324
325 N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
326 nP_coeff = -N_coeff + F_yn(i)*A_y;
327 V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
328
329 S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
330 sP_coeff = -S_coeff - F_ys(i)*A_y;
331 V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
332
333 % At corner
334 elseif ~etest && ~wtest && ~ntest && ~stest && wwall && wcorner
335
336     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
337             -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
338             0*(-max(F_yw(i)*A_x,0) - D_x*A_x);
339
340 % At western boundary (x = 0)
341 W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
342 wP_coeff = -W_coeff - F_yw(i)*A_x;
343
344 E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
345 eP_coeff = -E_coeff + F_ye(i)*A_x;
346 V(i, i+1) = E_coeff;
347
348 N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
349 nP_coeff = -N_coeff + F_yn(i)*A_y;
350 V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
351
352 S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
353 sP_coeff = -S_coeff - F_ys(i)*A_y;
354 V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
355
356
357 % At northern boundary (y = h)
358 elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner
359
360     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
361             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
362
363 % At northern boundary
364 nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;
365
366 E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
367 eP_coeff = -E_coeff + F_ye(i)*A_x;
368 V(i, i+1) = E_coeff;
369
370 W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
371 wP_coeff = -W_coeff - F_yw(i)*A_x;
372 V(i, i-1) = W_coeff;
373
374 S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
375 sP_coeff = -S_coeff - F_ys(i)*A_y;
376 V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
377
378

```

```

379 % At southern boundary (y = 0)
380 elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~wcorner
381     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
382             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
383
384 % At southern boundary (y = 0),
385 sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;
386
387 E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
388 eP_coeff = -E_coeff + F_ye(i)*A_x;
389 V(i, i+1) = E_coeff;
390
391 W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
392 wP_coeff = -W_coeff - F_yw(i)*A_x;
393 V(i, i-1) = W_coeff;
394
395 N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
396 nP_coeff = -N_coeff + F_yn(i)*A_y;
397 V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
398
399 %Not at any boundary
400 else
401     bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
402             -p_circ(i))*A_y + rho*g_y*del_y*A_y;
403
404 E_coeff = -max(0,-F_ye(i)*A_x) - D_x*A_x;
405 eP_coeff = -E_coeff + F_ye(i)*A_x;
406 V(i, i+1) = E_coeff;
407
408 W_coeff = -max(F_yw(i)*A_x,0) - D_x*A_x;
409 wP_coeff = -W_coeff - F_yw(i)*A_x;
410 V(i, i-1) = W_coeff;
411
412 N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
413 nP_coeff = -N_coeff + F_yn(i)*A_y;
414 V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
415
416 S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
417 sP_coeff = -S_coeff - F_ys(i)*A_y;
418 V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
419
420 end % if
421
422 % Filling in the rest of the matrix, adding all point coefficients
423 V(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
424
425 etest = false;
426 wtest = false;
427 ntest = false;
428 stest = false;
429 wwall = false;
430
431 end % for
432 v_star = V\bv'; % Matrix inversion

```

E.5.2.4 BFS_pressurecorrection_parabolic.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 % Pressure correction script for the BFS model %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5 T = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
6 beta = zeros(1, totalpoints); % Initialisation of source term vector
7
8 au = diag(U); % a^center-coefficients from the momentum equations
9 av = diag(V);
10
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12 % Calculation
13 for i = 1:totalpoints % Global indexing system
14
15     etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 ) ... % below step
16     || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);

```

```

17 ntest = totalpoints - N_total < i && i <= totalpoints ;
18 wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
19 wwall = i <= N_wide*M_wide && mod(i-1, N_wide) == 0;
20 stest = (1 <= i && i <= N_wide) ... % Excluding the corner value
21      || (N_wide*M_wide < i && i <= N_wide*M_wide + N_narrow);
22
23
24
25 % Northeastern corner
26 if etest && ~wtest && ntest && ~stest && ~wwall
27
28     beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1) ...
29             + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
30
31     % At eastern boundary (x = L)
32     eP_coeff = rho*A_x^2/au(i);
33
34     % At northern boundary (y = h) (y = H)
35     nP_coeff = 0 ;
36
37     W_coeff = -rho*A_x^2/au(i-1);
38     wP_coeff = -W_coeff;
39     T(i, i-1) = W_coeff;
40
41     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
42     sP_coeff = -S_coeff;
43     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
44
45
46 % Southeastern corner
47 elseif etest && ~wtest && ~ntest && stest && ~wwall
48
49     beta(i) = rho*(-A_x*u_star(i) + A_x*u_star(i-1) ...
50             - A_y*v_star(i));
51
52     % At eastern boundary (x = L)
53     eP_coeff = rho*A_x^2/au(i);
54
55     % At southern boundary (y = 0)
56     sP_coeff = 0;
57
58     W_coeff = -rho*A_x^2/au(i-1);
59     wP_coeff = -W_coeff;
60     T(i, i-1) = W_coeff;
61
62     N_coeff = -rho*A_y^2/av(i);
63     nP_coeff = -N_coeff;
64     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
65
66
67 % Northwestern corner
68 elseif ~etest && wtest && ntest && ~stest && ~wwall
69
70     beta(i) = rho*(-A_x*u_star(i) ...
71             + A_x*u_in(getRowNumber(i, N_wide, M_wide, N_total)) ...
72             + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
73
74     % At western boundary (x = 0)
75     wP_coeff = 0;
76
77     % At northern boundary (y = h) (y = H)
78     nP_coeff = 0 ;
79
80     E_coeff = -rho*A_x^2/au(i);
81     eP_coeff = -E_coeff ;
82     T(i, i+1) = E_coeff;
83
84     S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
85     sP_coeff = -S_coeff;
86     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
87
88
89 % Southwestern corner at inlet
90 elseif ~etest && wtest && ~ntest && stest && ~wwall
91
92     beta(i) = rho*(-A_x*u_star(i) ...

```



```

93         +A_x*u_in(getRowNumber(i, N_wide, M_wide, N_total)) ...
94         -A_y*v_star(i));
95
96     % At western boundary (x = 0)
97     wP_coeff = 0;
98
99     % At southern boundary (y = 0)
100    sP_coeff = 0;
101
102    E_coeff = -rho*A_x^2/au(i);
103    eP_coeff = -E_coeff ;
104    T(i, i+1) = E_coeff;
105
106    N_coeff = -rho*A_y^2/av(i);
107    nP_coeff = -N_coeff;
108    T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
109
110
111    % Southwestern corner at step
112    elseif ~etest && ~wtest && ~ntest && stest && wwall
113
114        beta(i) = rho*(-A_x*u_circ(i)...
115                +A_x*0 -A_y*v_circ(i)); % wall/"inlet" velocity is zero
116
117        % At western boundary (x = 0)
118        wP_coeff = 0;
119
120        % At southern boundary (y = 0)
121        sP_coeff = 0;
122
123        E_coeff = -rho*A_x^2/au(i);
124        eP_coeff = -E_coeff ;
125        T(i, i+1) = E_coeff;
126
127        N_coeff = -rho*A_y^2/av(i);
128        nP_coeff = -N_coeff;
129        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
130
131
132    % At eastern boundary (x = L)
133    elseif etest && ~wtest && ~ntest && ~stest && ~wwall
134
135        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
136                -A_y*v_star(i) + ...
137                A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
138
139        % At eastern boundary (x = L)
140        eP_coeff = rho*A_x^2/au(i);
141
142
143        W_coeff = -rho*A_x^2/au(i-1);
144        wP_coeff = -W_coeff;
145        T(i, i-1) = W_coeff;
146
147        N_coeff = -rho*A_y^2/av(i);
148        nP_coeff = -N_coeff;
149        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
150
151        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
152        sP_coeff = -S_coeff;
153        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
154
155
156    % At western boundary at inlet (x = 0)
157    elseif ~etest && wtest && ~ntest && ~stest && ~wwall
158
159        beta(i) = rho*(-A_x*u_star(i) ...
160                +A_x*u_in(getRowNumber(i, N_wide, M_wide, N_total)) ...
161                -A_y*v_star(i) ...
162                + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
163
164        % At western boundary (x = 0)
165        wP_coeff = 0;
166
167        E_coeff = -rho*A_x^2/au(i);
168        eP_coeff = -E_coeff ;

```

```

169     T(i, i+1) = E_coeff;
170
171     N_coeff = -rho*A_y^2/av(i);
172     nP_coeff = -N_coeff;
173     T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
174
175     S_coeff =- rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
176     sP_coeff = -S_coeff;
177     T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
178
179
180 % At western wall
181 elseif ~etest && ~wtest && ~ntest && ~stest && wwall
182
183     beta(i) = rho*(-A_x*u_circ(i)...
184               +A_x*0 -A_y*v_circ(i) +...
185               A_y*v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
186
187 % At western boundary (x = 0)
188 wP_coeff = 0;
189
190 E_coeff = -rho*A_x^2/au(i);
191 eP_coeff = -E_coeff ;
192 T(i, i+1) = E_coeff;
193
194 N_coeff = -rho*A_y^2/av(i);
195 nP_coeff = -N_coeff;
196 T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
197
198 S_coeff =- rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
199 sP_coeff = -S_coeff;
200 T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
201
202
203 % At northern boundary (y = h)
204 elseif ~etest && ~wtest && ntest && ~stest && ~wwall
205
206     beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
207               + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
208
209 % At northern boundary (y = h)
210 nP_coeff = 0 ;
211
212 E_coeff = -rho*A_x^2/au(i);
213 eP_coeff = -E_coeff ;
214 T(i, i+1) = E_coeff;
215
216 W_coeff = -rho*A_x^2/au(i-1);
217 wP_coeff = -W_coeff;
218 T(i, i-1) = W_coeff;
219
220 S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
221 sP_coeff = -S_coeff;
222 T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
223
224
225 % At southern boundary (y = 0)
226 elseif ~etest && ~wtest && ~ntest && stest && ~wwall
227
228     beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
229               -A_y*v_star(i));
230
231 % At southern boundary (y = 0)
232 sP_coeff = 0;
233
234 E_coeff = -rho*A_x^2/au(i);
235 eP_coeff = -E_coeff ;
236 T(i, i+1) = E_coeff;
237
238 W_coeff = -rho*A_x^2/au(i-1);
239 wP_coeff = -W_coeff;
240 T(i, i-1) = W_coeff;
241
242 N_coeff = -rho*A_y^2/av(i);
243 nP_coeff = -N_coeff;
244 T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

```

```

245
246
247     %Not at any boundary
248     else
249
250         beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) -A_y*v_star(i) + ...
251             A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));
252
253         E_coeff = -rho*A_x^2/au(i);
254         eP_coeff = -E_coeff ;
255         T(i, i+1) = E_coeff;
256
257         W_coeff = -rho*A_x^2/au(i-1);
258         wP_coeff = -W_coeff;
259         T(i, i-1) = W_coeff;
260
261         N_coeff = -rho*A_y^2/av(i);
262         nP_coeff = -N_coeff;
263         T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
264
265         S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
266         sP_coeff = -S_coeff;
267         T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;
268
269     end % if
270
271     % Filling in the rest of the matrix, adding all point coefficients
272     T(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;
273
274
275     etest = false;
276     wtest = false;
277     ntest = false;
278     stest = false;
279     wwall = false;
280
281 end %for
282 p_corr = T\beta'; % Matrix inversion

```

E.5.2.5 plotColoredQuiver_parabolic.m

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                               Colored velocity quiver plots                               %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  filler = 0; % For the quiver plots, the velocities at the step are set to
5              % zero and not Inf, rectangles are therefore used to block
6              % out the step from the plots afterwards.
7  levels = 50; % Number of different colors for the representation
8  showvals = false; % Show the value of each color
9  lines = 'none'; % Show lines in between each color
10
11 % u-velocity
12 u_fullplot = zeros(M_total+2, N_total+1);
13 u_fullplot(2:end-1,1) = u_in;
14 u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
15     global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
16 u_fullplot(M_wide+2:end-1,2:end) = ...
17     global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
18 u_fullplot(1:M_wide, 1:N_narrow) = 0;
19
20 % Transformation from dimensionless to regular
21 u_fullplot = u_fullplot*u_in_true;
22
23
24 % v-velocity
25 v_fullplot = zeros(m_total+2, N_total+1);
26 v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
27     global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
28 v_fullplot(m_wide+2:end-1,2:end) = ...
29     global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
30 v_fullplot(1:m_wide, 1:N_narrow) = filler;
31
32 % Transformation from dimensionless to regular
33 v_fullplot = v_fullplot*u_in_true;
34
35

```

```

36 uSN = zeros(M_total, N_total);
37 vSN = zeros(M_total, N_total);
38 for i = 2:N_total+1
39     for j = 1:M_total
40         uSN(j,i-1) = 1/2*(u_fullplot(j+1,i-1) + u_fullplot(j+1,i));
41     end %for
42 end %for
43 for j = 2:M_total+1
44     for i = 1:N_total
45         vSN(j-1,i) = 1/2*(v_fullplot(j-1,i) + v_fullplot(j,i));
46     end %for
47 end %for
48
49 % Need to make a combined velocity vector
50 combvel = sqrt(uSN.^2 + vSN.^2);
51
52 % Create a mesh for the plotting
53 [xSN,ySN] = meshgrid(...
54     x_0+ del_x_true/2:del_x_true:x_N-del_x_true/2, ...
55     y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);
56
57 combvelwall = [zeros(1,N_total); combvel ; zeros(1,N_total)];
58
59
60 fq1 = figure;
61 % Contour plot
62 [M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
63     [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
64     combvelwall,levels);
65 c.LineColor = lines;
66 hold on
67 qn = quiver( xSN, ySN , uSN , vSN,'LineWidth',0.5,'Color','k');
68
69 %Block out the step
70 r = rectangle('Position',[0.03 0 1 1]);
71 r.FaceColor = [1 1 1];
72 r.EdgeColor = 'none';%'k';
73 r.LineWidth = .0000010;
74
75 hold on
76 set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
77     'Marker','o','MarkerSize',1,'ShowArrowHead','on')
78 s = sprintf('Plot of velocities as vectors after %d iterations', it );
79 % f = title(s);
80 ax = gca;
81 % set(f,'interpreter','latex','fontSize', 16)
82 set(gca,'TickLabelInterpreter','latex')
83 ax.FontSize = 12;
84 xlabel('$x$-direction [m]', 'interpreter', 'latex')
85 xlim([0,L_total])
86 ylabel('$y$-direction [m]', 'interpreter', 'latex')
87 ytickformat('%1f')
88 set(fq1,'Position',[3 250 717 420]);
89 saveas(gcf,'velocityquiver.png')
90 ax.Layer = 'top';
91
92
93 fq2 = figure;
94 [M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
95     [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
96     combvelwall,levels);
97 c.LineColor = lines;
98 hold on
99 qn = quiver(...
100     xSN, ySN , uSN , vSN,...%u_fullplot(1:end-1,:)
101     'LineWidth',0.5,'Color','k');
102
103 r = rectangle('Position',[0.03 0 1 1]);
104 r.FaceColor = [1 1 1];
105 r.EdgeColor = 'none';%'k';
106 r.LineWidth = .0000010;
107
108
109 hold on
110 set(qn,'AutoScale','on', 'AutoScaleFactor', 2.1,'Marker','o',...
111     'MarkerSize',1,'MaxHeadSize',0.01);%'ShowArrowHead','off')

```

```

112 % qw = quiver(...
113 %     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
114 %     'LineWidth',0.5,'Color','k');
115 s = sprintf(...
116     'Plot of velocities as vectors after %d iterations scales x 1.5', it );
117 % f = title(s);
118 ax = gca;
119 % set(f, 'interpreter', 'latex', 'fontsize', 16)
120 set(gca, 'TickLabelInterpreter', 'latex')
121 ax.FontSize = 12;
122 xlabel('$x$-direction [m]', 'interpreter', 'latex')
123 xlim([1-1/4,1*3])
124 ylabel('$y$-direction [m]', 'interpreter', 'latex')
125 ylim([0,H+H/4])
126 ytickformat('%1f')
127 set(fq2, 'Position', [724 250 560 420]);
128 saveas(gcf, 'velocityquiver1zoomed.png')
129 ax.Layer = 'top';
130
131
132 fq3 = figure;
133 [M,c] = contourf([xSN(1,:) ; xSN ; xSN(end,:)],...
134     [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
135     combvelwall, levels);
136 c.LineColor = lines;
137 hold on
138 qn = quiver(...
139     xSN(1:M_wide,N_narrow+1:N_narrow*2), ...
140     ySN(1:M_wide,N_narrow+1:N_narrow*2), ...
141     uSN(1:M_wide,N_narrow+1:N_narrow*2) , ...
142     vSN(1:M_wide,N_narrow+1:N_narrow*2) ...%u_fullplot(1:end-1,:)
143     'LineWidth',0.5,'Color','k');
144
145 r = rectangle('Position',[0.03 0 1 1]);
146 r.FaceColor = [1 1 1];
147 r.EdgeColor = 'none';%'k';
148 r.LineWidth = .0000010;
149
150
151 hold on
152 set(qn, 'AutoScale', 'on', 'LineWidth',0.1, 'AutoScaleFactor', 2.1,...
153     'Marker', 'o', 'MarkerSize', 1, 'ShowArrowHead', 'on')
154 % qw = quiver(...
155 %     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
156 %     'LineWidth',0.5,'Color','k');
157 s = sprintf(...
158     'Plot of velocities as vectors after %d iterations, scaled * 2', it );
159 % f = title(s);
160 ax = gca;
161 % set(f, 'interpreter', 'latex', 'fontsize', 16)
162 set(gca, 'TickLabelInterpreter', 'latex')
163 ax.FontSize = 12;
164 xlabel('$x$-direction [m]', 'interpreter', 'latex')
165 xlim([1,2*1])
166 ylabel('$y$-direction [m]', 'interpreter', 'latex')
167 ylim([0,H])
168 ytickformat('%1f')
169 set(fq3, 'Position', [724 250 560 420]);
170 saveas(gcf, 'velocityquiver2zoomed.png')
171 ax.Layer = 'top';

```

E.6 Grid Generation

The code `transfinite.m` is used to get the algebraic grid by use of Transfinite Interpolation. The code `elliptic.m` is used to get the grid by use of the elliptic grid generation equation. The code `getCol.m` is used to get the column of the initial guess matrix for each point in the globally indexed vector when filling in the coefficient matrix. The code `getRow.m` is used to get the row of the initial guess matrix for each point in the globally indexed vector when filling in the coefficient matrix. The code `matrix2global.m` is used to convert the matrices into their corresponding globally

indexed vectors given the dimensions of the matrix. The code `global2matrix.m` is used to convert the globally indexed vectors into their corresponding matrices given the dimensions of the matrix.

E.6.1 Codes

E.6.1.1 `transfinite.m`

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                               Transfinite Interpolation                               %
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  close all
5  clc
6  %% Settings
7  N = 71; % Number of points in q1/x-direction
8  M = 21; % Number of points in q2/y-direction
9
10 x_max = 35; % Total length of physical domain (including step)
11 y_max = 2; % Total height of physical domain (including step)
12
13 h = 1; % Height of the step
14 l = 5; % Length / width of the step
15
16 % Placement of points E and F splits the line segment AD in s equal pieces.
17 s = 3;
18
19
20 %% Boundary points
21 q1 = 0:N; % Specifying the q1-points with spacing of delta q1 = 1
22 q2 = 0:M; % Specifying the q1-points with spacing of delta q1 = 1
23
24 % Specifying the locations of points A-F in the physical domain
25 xA = 0;
26 xB = 0;
27 xC = x_max;
28 xD = x_max;
29 xE = 1;
30 xF = 1;
31
32 yA = h;
33 yB = y_max;
34 yC = y_max;
35 yD = 0;
36 yE = 0;
37 yF = h;
38
39 % Place points E and F to split the line segment AD in s equal pieces.
40 AFfrac = 1/s; % Fraction of total width of q1
41 AEfrac = 1/s; % Fraction of total width of q1
42 AFpoints = ceil(AFfrac * N); % Number of q1-points in line segment AF
43 FEpoints = floor(AEfrac * N); % Number of q1-points in line segment FE
44
45 q1AF = 0:AFpoints; % Vector of coordinates q1 for the line segment AF
46 q1FE = 0:FEpoints; % Vector of coordinates q1 for the line segment FE
47 q1ED = 0:(N-AFpoints-FEpoints); % Vector of coordinates q1 for ...
48                                     % the line segment ED
49
50 % Calculation of the boundary points:
51 xAB = (1-q2/q2(end))* xA + q2/q2(end)*xB;
52 xBC = (1-q1/q1(end))* xB + q1/q1(end)*xC;
53 xDC = (1-q2/q2(end))* xD + q2/q2(end)*xC;
54 xED = (1-q1ED/q1ED(end))* xE + q1ED/q1ED(end)*xD;
55 xFE = (1-q1FE/q1FE(end))* xF + q1FE/q1FE(end)*xE;
56 xAF = (1-q1AF/q1AF(end))* xA + q1AF/q1AF(end)*xF;
57
58 yAB = (1-q2/q2(end))* yA + q2/q2(end)*yB;
59 yBC = (1-q1/q1(end))* yB + q1/q1(end)*yC;
60 yDC = (1-q2/q2(end))* yD + q2/q2(end)*yC;
61 yED = (1-q1ED/q1ED(end))* yE + q1ED/q1ED(end)*yD;
62 yFE = (1-q1FE/q1FE(end))* yF + q1FE/q1FE(end)*yE;
63 yAF = (1-q1AF/q1AF(end))* yA + q1AF/q1AF(end)*yF;
64
65 % Plot with the boundary points

```

```

66 % figure
67 % plot(xAB,yAB,'x',xBC,yBC,'x',xDC,yDC,'x',...
68 %      xED,yED,'x',xFE,yFE,'x',xAF,yAF,'x')
69 % % xlim([-0.1,1.1])
70 % % ylim([-0.1,1.1])
71 % legend({'$AB$', '$BC$', '$CD$', '$DE$', '$EF$', '$FA$', ...
72 %        'Interpreter', 'latex', 'Location', 'best'})
73
74 %% Center domain points
75 % Combining the x- and y-points for the line segments AF, FE and ED to ...
76 % one vector for AD. The points located exactly at F and E are ...
77 % overlapping and removed from xFE by taking xFE(2:end-1). Likewise for y.
78
79 xAD = [xAF xFE(2:end-1) xED];% Combining the x-points for the line segment
80 yAD = [yAF yFE(2:end-1) yED];% Combining the y-points for the line segment
81
82 % Initialising the matrix x of points in the physical domain
83 x = zeros(length(q2),length(q1));
84 % Initialising the matrix y of points in the physical domain
85 y = zeros(length(q2),length(q1));
86
87 % Calculating the center points
88 for j =1:length(q2)
89     for i = 1:length(q1)
90         x(j,i) = (1-q1(i)/q1(end))* xAB(j) +(q1(i)/q1(end)) *xDC(j)...
91             +(1-q2(j)/q2(end))*xAD(i) +(q2(j)/q2(end))* xBC(i)...
92             -(1-q1(i)/q1(end))* (1-q2(j)/q2(end))* xA...
93             -(1-q1(i)/q1(end))* (q2(j)/q2(end))* xB...
94             -(q1(i)/q1(end))*(1-q2(j)/q2(end))* xD...
95             -(q1(i)/q1(end))*(q2(j)/q2(end))* xC;
96         y(j,i) = (1-q1(i)/q1(end))* yAB(j) +(q1(i)/q1(end)) *yDC(j)...
97             +(1-q2(j)/q2(end))*yAD(i) +(q2(j)/q2(end))* yBC(i)...
98             -(1-q1(i)/q1(end))* (1-q2(j)/q2(end))* yA...
99             -(1-q1(i)/q1(end))* (q2(j)/q2(end))* yB...
100            -(q1(i)/q1(end))*(1-q2(j)/q2(end))* yD...
101            -(q1(i)/q1(end))*(q2(j)/q2(end))* yC;
102     end %for
103 end %for
104
105 % Plotting the resulting grid
106 figure
107 plot(x,y,'k','x','y','k')
108 xlim([xA,xD])
109 ylim([yD,yC])
110 set(gca,'TickLabelInterpreter','latex')
111 xlabel('$x$-direction [m]', 'interpreter', 'latex')
112 ylabel('$y$-direction [m]', 'interpreter', 'latex')
113 saveas(gcf,'transfinite.png')

```

E.6.1.2 elliptic.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %                               Elliptic grid generation %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 close all
5 clc
6 clear
7
8 maxits = 75;
9
10 P1 = 0; % Poisson control function
11 P2 = 0; % Poisson control function
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 %% Create the algebraic grid for an initial guess
14 transfinite
15 N = length(q1);
16 M = length(q2);
17 n = N-2; % dimensions of the inner point matrix to be solved for
18 m = M-2; % with the elliptic grid generation equations below
19 alpha = 0.001;
20
21 conv = 0;
22 it = 1;
23
24 while conv == 0
25

```

```

26 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
27 % Area Components
28 AM11 = zeros(m,n); % Indexed top bottom
29 AM12 = zeros(m,n); % A^1_2
30 AM21 = zeros(m,n); % A^2_1
31 AM22 = zeros(m,n);
32 for i = 2:N-1
33     for j = 2:M-1
34         AM11(j-1,i-1) = 1/2*(y(j+1,i) - y(j-1,i));
35         AM21(j-1,i-1) = -1/2*(y(j,i+1) - y(j,i-1));
36         AM12(j-1,i-1) = -1/2*(x(j+1,i) - x(j-1,i));
37         AM22(j-1,i-1) = 1/2*(x(j,i+1) - x(j,i-1));
38     end %for
39 end %for
40
41 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
42 % Jacobi Determinant
43 J2 = zeros(m,n);
44 for i = 2:N-1
45     for j = 2:M-1
46         J2(j-1,i-1) = (1/4*(x(j,i+1)-x(j,i-1))*(y(j+1,i)-y(j-1,i))...
47             - 1/4*(y(j,i+1)-y(j,i-1))*(x(j+1,i)-x(j-1,i)))^2;
48     end %for
49 end %for
50
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
52 % Contravariant Tensor Components
53 gM11 = zeros(m,n);
54 gM12 = zeros(m,n);
55 gM21 = zeros(m,n);
56 gM22 = zeros(m,n);
57 for i = 1:n
58     for j = 1:m
59         gM11(j,i) = 1/J2(j,i)*...
60             (AM11(j,i)*AM11(j,i) + AM12(j,i)*AM12(j,i));
61         gM21(j,i) = 1/J2(j,i)*...
62             (AM21(j,i)*AM11(j,i) + AM22(j,i)*AM12(j,i));
63         gM12(j,i) = 1/J2(j,i)*...
64             (AM11(j,i)*AM21(j,i) + AM12(j,i)*AM22(j,i));
65         gM22(j,i) = 1/J2(j,i)*...
66             (AM21(j,i)*AM21(j,i) + AM22(j,i)*AM22(j,i));
67     end %for
68 end %for
69
70 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
71 % Matrices 2 Globals
72 A11 = matrix2global(AM11,n,m);
73 A12 = matrix2global(AM12,n,m);
74 A21 = matrix2global(AM21,n,m);
75 A22 = matrix2global(AM22,n,m);
76
77 g11 = matrix2global(gM11,n,m);
78 g12 = matrix2global(gM12,n,m);
79 g21 = matrix2global(gM21,n,m);
80 g22 = matrix2global(gM22,n,m);
81
82 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
83 % New x and y
84 X = zeros(n*m,n*m);
85 Y = zeros(n*m,n*m);
86
87 % The source term is zero and is updated if the point is at a boundary
88 bx = zeros(1,n*m);
89 by = zeros(1,n*m);
90
91
92 for i = 1:n*m
93
94     etest = mod(i, n) == 0;
95     ntest = n*m - n < i;
96     wtest = mod(i-1, n) == 0;
97     stest = i <= n;
98
99     % Northeastern corner
100     if etest && ~wtest && ntest && ~stest

```



```

102     X(i,i) = -2*g11(i)-2*g22(i);
103     % X(i,i+1)=(g11(i)+P1/2) ;
104     bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
105     X(i,i-1)=(g11(i)-P1/2) ;
106     % X(i+n,i)=(g22(i)+P2/2) ;
107     bx(i) = bx(i) - x(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
108     X(i-n,i)=(g22(i)-P2/2) ;
109     % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
110     bx(i) = bx(i)...
111         - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
112     % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
113     bx(i) = bx(i)...
114         - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
115     % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
116     bx(i) = bx(i)...
117         - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
118     X(i-n,i-1)=(g12(i)/4+g21(i)/4);
119
120     Y(i,i) = -2*g11(i)-2*g22(i);
121     % Y(i,i+1)=(g11(i)+P1/2) ;
122     by(i) = by(i) - y(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
123     Y(i,i-1)=(g11(i)-P1/2) ;
124     % Y(i+n,i)=(g22(i)+P2/2) ;
125     by(i) = by(i) - y(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
126     Y(i-n,i)=(g22(i)-P2/2) ;
127     % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
128     by(i) = by(i)...
129         - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
130     % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
131     by(i) = by(i)...
132         - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
133     % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
134     by(i) = by(i)...
135         - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
136     Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
137
138     % Southeastern corner
139     elseif etest && ~wtest && ~ntest && stest
140
141     X(i,i) = -2*g11(i)-2*g22(i);
142     % X(i,i+1)=(g11(i)+P1/2) ;
143     bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
144     X(i,i-1)=(g11(i)-P1/2) ;
145     X(i+n,i)=(g22(i)+P2/2) ;
146     % X(i-n,i)=(g22(i)-P2/2) ;
147     bx(i) = bx(i) - x(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
148     % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
149     bx(i) = bx(i)...
150         - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
151     X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
152     % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
153     bx(i) = bx(i)...
154         - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
155     % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
156     bx(i) = bx(i)...
157         - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
158
159
160     Y(i,i) = -2*g11(i)-2*g22(i);
161     % Y(i,i+1)=(g11(i)+P1/2) ;
162     by(i) = by(i) - y(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
163     Y(i,i-1)=(g11(i)-P1/2) ;
164     Y(i+n,i)=(g22(i)+P2/2) ;
165     % Y(i-n,i)=(g22(i)-P2/2) ;
166     by(i) = by(i) - y(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
167     % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
168     by(i) = by(i)...
169         - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
170     Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
171     % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
172     by(i) = by(i)...
173         - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
174     % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
175     by(i) = by(i)...
176         - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
177

```

```

178 % Northwestern corner
179 elseif ~etest && wtest && ntest && ~stest
180
181     X(i,i) = -2*g11(i)-2*g22(i);
182     X(i,i+1)=(g11(i)+P1/2) ;
183     % X(i,i-1)=(g11(i)-P1/2) ;
184     bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
185     % X(i+n,i)=(g22(i)+P2/2) ;
186     bx(i) = bx(i) - x(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
187     X(i-n,i)=(g22(i)-P2/2) ;
188     % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
189     bx(i) = bx(i)...
190         - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
191     % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
192     bx(i) = bx(i)...
193         - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
194     X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
195     % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
196     bx(i) = bx(i)...
197         - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
198
199     Y(i,i) = -2*g11(i)-2*g22(i);
200     Y(i,i+1)=(g11(i)+P1/2) ;
201     % Y(i,i-1)=(g11(i)-P1/2) ;
202     by(i) = by(i) - y(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
203     % Y(i+n,i)=(g22(i)+P2/2) ;
204     by(i) = by(i) - y(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
205     Y(i-n,i)=(g22(i)-P2/2) ;
206     % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
207     by(i) = by(i)...
208         - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
209     % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
210     by(i) = by(i)...
211         - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
212     Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
213     % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
214     by(i) = by(i)...
215         - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
216
217 % Southwestern corner
218 elseif ~etest && wtest && ~ntest && stest
219
220
221     X(i,i) = -2*g11(i)-2*g22(i);
222     X(i,i+1)=(g11(i)+P1/2) ;
223     % X(i,i-1)=(g11(i)-P1/2) ;
224     bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
225     X(i+n,i)=(g22(i)+P2/2) ;
226     % X(i-n,i)=(g22(i)-P2/2) ;
227     bx(i) = bx(i) - x(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
228     X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
229     % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
230     bx(i) = bx(i)...
231         - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
232     % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
233     bx(i) = bx(i)...
234         - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4) ;
235     % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
236     bx(i) = bx(i)...
237         - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
238
239     Y(i,i) = -2*g11(i)-2*g22(i);
240     Y(i,i+1)=(g11(i)+P1/2) ;
241     % Y(i,i-1)=(g11(i)-P1/2) ;
242     by(i) = by(i) - y(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
243     Y(i+n,i)=(g22(i)+P2/2) ;
244     % Y(i-n,i)=(g22(i)-P2/2) ;
245     by(i) = by(i)...
246         - y(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
247     Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
248     % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
249     by(i) = by(i)...
250         - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
251     % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
252     by(i) = by(i)...
253         - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);

```

```

254         % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
255         by(i) = by(i)...
256             - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
257
258     % At eastern boundary (x = L)
259     elseif etest && ~wtest && ~ntest && ~stest
260
261         X(i,i) = -2*g11(i)-2*g22(i);
262         % X(i,i+1)=(g11(i)+P1/2) ;
263         bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
264         X(i,i-1)=(g11(i)-P1/2) ;
265         X(i+n,i)=(g22(i)+P2/2) ;
266         X(i-n,i)=(g22(i)-P2/2) ;
267         % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
268         bx(i) = bx(i)...
269             - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
270         X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
271         % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
272         bx(i) = bx(i)...
273             - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
274         X(i-n,i-1)=(g12(i)/4+g21(i)/4);
275
276         Y(i,i) = -2*g11(i)-2*g22(i);
277         % Y(i,i+1)=(g11(i)+P1/2) ;
278         by(i) = by(i) - y(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
279         Y(i,i-1)=(g11(i)-P1/2) ;
280         Y(i+n,i)=(g22(i)+P2/2) ;
281         Y(i-n,i)=(g22(i)-P2/2) ;
282         % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
283         by(i) = by(i)...
284             - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
285         Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
286         % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
287         by(i) = by(i)...
288             - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
289         Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
290
291     % At western boundary
292     elseif ~etest && wtest && ~ntest && ~stest
293
294         X(i,i) = -2*g11(i)-2*g22(i);
295         X(i,i+1)=(g11(i)+P1/2) ;
296         % X(i,i-1)=(g11(i)-P1/2) ;
297         bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
298         X(i+n,i)=(g22(i)+P2/2) ;
299         X(i-n,i)=(g22(i)-P2/2) ;
300         X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
301         % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
302         bx(i) = bx(i)...
303             - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4);
304         X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
305         % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
306         bx(i) = bx(i)...
307             - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
308
309         Y(i,i) = -2*g11(i)-2*g22(i);
310         Y(i,i+1)=(g11(i)+P1/2) ;
311         % Y(i,i-1)=(g11(i)-P1/2) ;
312         by(i) = by(i) - y(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
313         Y(i+n,i)=(g22(i)+P2/2) ;
314         Y(i-n,i)=(g22(i)-P2/2) ;
315         Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
316         % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
317         by(i) = by(i)...
318             - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4);
319         Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
320         % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
321         by(i) = by(i)...
322             - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
323
324     % At northern boundary (y = h)
325     elseif ~etest && ~wtest && ntest && ~stest
326
327         X(i,i) = -2*g11(i)-2*g22(i);
328         X(i,i+1)=(g11(i)+P1/2) ;
329         X(i,i-1)=(g11(i)-P1/2) ;

```

```

330     % X(i+n,i)=(g22(i)+P2/2) ;
331     bx(i) = bx(i) - x(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
332     X(i-n,i)=(g22(i)-P2/2) ;
333     % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
334     bx(i) = bx(i)...
335         - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
336     % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
337     bx(i) = bx(i)...
338         - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
339     X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
340     X(i-n,i-1)=(g12(i)/4+g21(i)/4);
341
342     Y(i,i) = -2*g11(i)-2*g22(i);
343     Y(i,i+1)=(g11(i)+P1/2) ;
344     Y(i,i-1)=(g11(i)-P1/2) ;
345     % Y(i+n,i)=(g22(i)+P2/2) ;
346     by(i) = by(i) - y(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
347     Y(i-n,i)=(g22(i)-P2/2) ;
348     % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
349     by(i) = by(i)...
350         - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
351     % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
352     by(i) = by(i)...
353         - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
354     Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
355     Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
356
357
358     % At southern boundary (y = 0)
359     elseif ~etest && ~wtest && ~ntest && stest
360
361         X(i,i) = -2*g11(i)-2*g22(i);
362         X(i,i+1)=(g11(i)+P1/2) ;
363         X(i,i-1)=(g11(i)-P1/2) ;
364         X(i+n,i)=(g22(i)+P2/2) ;
365         % X(i-n,i)=(g22(i)-P2/2) ;
366         bx(i) = bx(i) - x(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
367         X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
368         X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
369         % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
370         bx(i) = bx(i)...
371             - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
372         % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
373         bx(i) = bx(i)...
374             - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
375
376         Y(i,i) = -2*g11(i)-2*g22(i);
377         Y(i,i+1)=(g11(i)+P1/2) ;
378         Y(i,i-1)=(g11(i)-P1/2) ;
379         Y(i+n,i)=(g22(i)+P2/2) ;
380         % Y(i-n,i)=(g22(i)-P2/2) ;
381         by(i) = by(i) - y(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
382         Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
383         Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
384         % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
385         by(i) = by(i)...
386             - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
387         % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
388         by(i) = by(i)...
389             - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);
390
391     %Not at any boundary
392     else
393
394         X(i,i) = -2*g11(i)-2*g22(i);
395         X(i,i+1)=(g11(i)+P1/2) ;
396         X(i,i-1)=(g11(i)-P1/2) ;
397         X(i+n,i)=(g22(i)+P2/2) ;
398         X(i-n,i)=(g22(i)-P2/2) ;
399         X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
400         X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
401         X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
402         X(i-n,i-1)=(g12(i)/4+g21(i)/4);
403
404         Y(i,i) = -2*g11(i)-2*g22(i);
405         Y(i,i+1)=(g11(i)+P1/2) ;

```



```

4 function rownumber = getRow(a, N)
5 rownumber = zeros(length(a),1);
6 for j = 1:length(a)
7     i = a(j);
8     rownumber(j) = floor((N+i-1)/N);
9 end %for
10 % Adjusting since the x matrix also contains boundary points
11 rownumber = rownumber + 1;
12 end %function

```

E.6.1.5 matrix2global.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %      Function transforming a matrix into a globally indexed vector      %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function res = matrix2global(vec, N, M)
5     for j = 1:M
6         vstart = 1;
7         rowstartpoint = N*M + (j-M)*N + 1;
8         rowendpoint = N*M + (j-M)*N;
9         res(rowstartpoint:rowendpoint) = vec(j,vstart:N);
10    end %for
11 end %function

```

E.6.1.6 global2matrix.m

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %      Function transforming a globally indexed vector into a matrix      %
3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4 function [matrix] = global2matrix(glob, N, M)
5     for j = 1:M % "down"           % the rest of the points are zero
6         for i = 1:N % "left"
7             matrix(j,i) = glob((j-1)*N + i);
8         end % for
9     end % for
10 end %function

```

