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

Master's thesis

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

NTNU
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

NTNU
Norwegian University of
Science and Technology

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 strømningsligningene 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 parabolsk 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. Strømningsligningene har blitt løst på sin dimensjonsløse form, og for den utvidede kanalen ble strømningen modellert for ulike lave Reynoldstall mellom 0.0001 og 400. Lengen på resurkulasjonssonene etter steget stemmer overens med resultater fra literaturen, men grunnet det relativt lave antallet celler brukt i beregningene er ikke resirkulasjonen synlig for de laveste Reynoldstallene. Strømingsmønsteret 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	Pa·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	$Pa{\cdot}m^2$	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	Pa·s	Viscosity
N	-	Number of scalar nodes in x-direction
$\mathbf{n}$	-	Direction vector normal to surface
ν	s· 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
$\mathbf{q}r$	-	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
$\tilde{}$	Adjusted variable
$\hat{}$	Dimensionless variable

Superscripts

Superscript	Description
$*$	Intermediate obtained after matrix inversion
$\circ$	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].

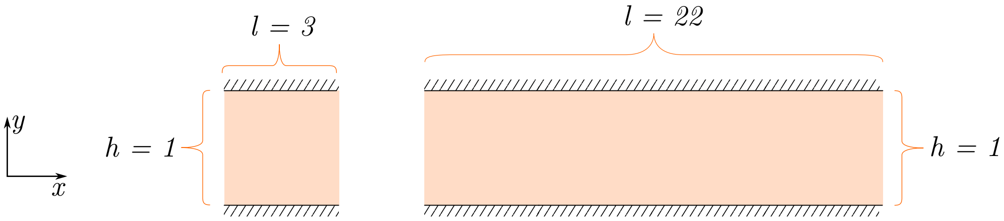


Figure 1.1: Straight channel domains.

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].

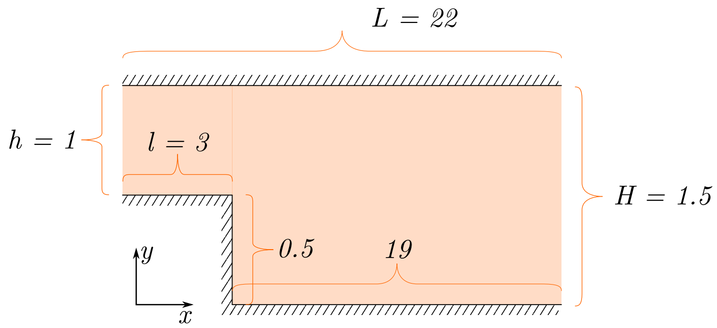


Figure 1.2: Domain as used by Melaaen [3], used to develop the two dimensional model for fluid flow over a backwards facing step.

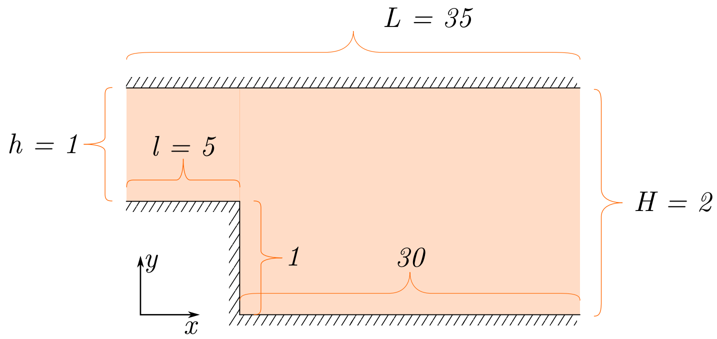


Figure 1.3: Domain as used by Biswas et al. [4], used in the backwards facing step model with a variation of Reynolds numbers for comparison to the results given by Biswas et al. [4].

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.

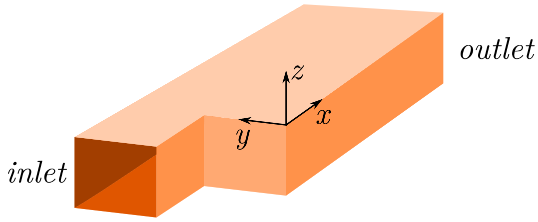


Figure 1.4: Example backwards facing step channel in three dimensions.

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].

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

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.

$$\nabla \cdot \mathbf{u} = 0 \tag{2.1.2}$$

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

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

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 $y = 0$.

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

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

$$u_{\max} = 2u_{avg} \tag{2.1.5}$$

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 $y = \frac{h}{2}$.

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

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

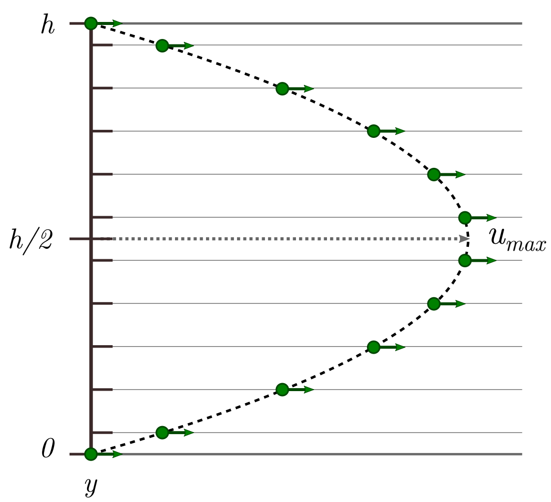


Figure 2.1: A parabolic velocity profile with $u_{\max}$ located at $y = \frac{h}{2}$.

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].

$$Re = \frac{\rho u D}{\mu} \tag{2.1.7}$$

where ρ is the density of the fluid, u is the average velocity defined as the volumetric flow rate devided 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).

$$Re = \frac{\rho u D_{hyd}}{\mu} \tag{2.1.8}$$

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

$$D_{hyd} = \frac{2hw}{h+w} \tag{2.1.9}$$

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).

$$D_{hyd} = \frac{2h}{h+1} \tag{2.1.10}$$

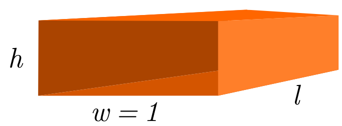


Figure 2.2: Rectangular duct with labels for the height h, width w and length l used in the calculation of the hydraulic diameter.

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].

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

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

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

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.

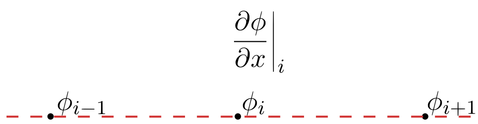


Figure 2.3: Discretisation method where the derivative $\left.\frac{\partial \phi}{\partial x}\right|_i$ is calculated in the same point as ϕ_i.

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

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

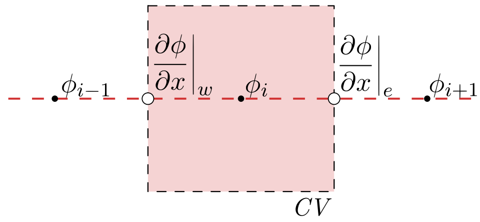


Figure 2.4: Discretisation in the Finite Volume method where the derivatives are calculated at the cell faces of the control volume CV around ϕ_i.

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).

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

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.

$$F_e\phi_e - F_w\phi_w = D_e\left(\phi_E - \phi_P\right) - D_w\left(\phi_P - \phi_W\right) \tag{2.2.3}$$

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).

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

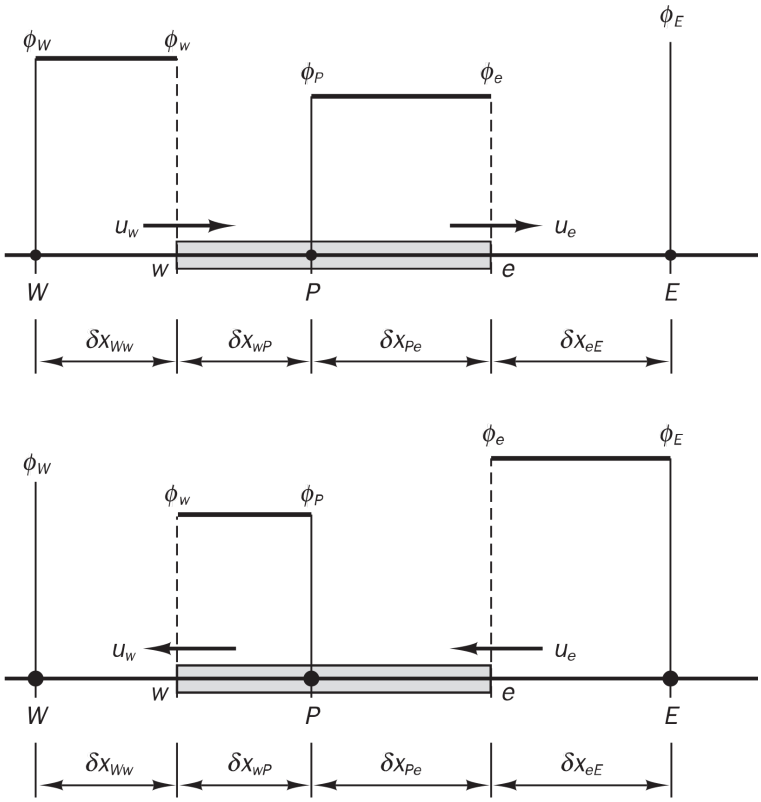


Figure 2.5: The Upwind Differencing Scheme visualised, the top figure shows the scheme for an eastgoing (positive) flow direction and the bottom figure shows the scheme for a westgoing (negative) flow direction. The figure is taken from Versteeg and Malalasekera [2].

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).

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

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, resepctively.

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.

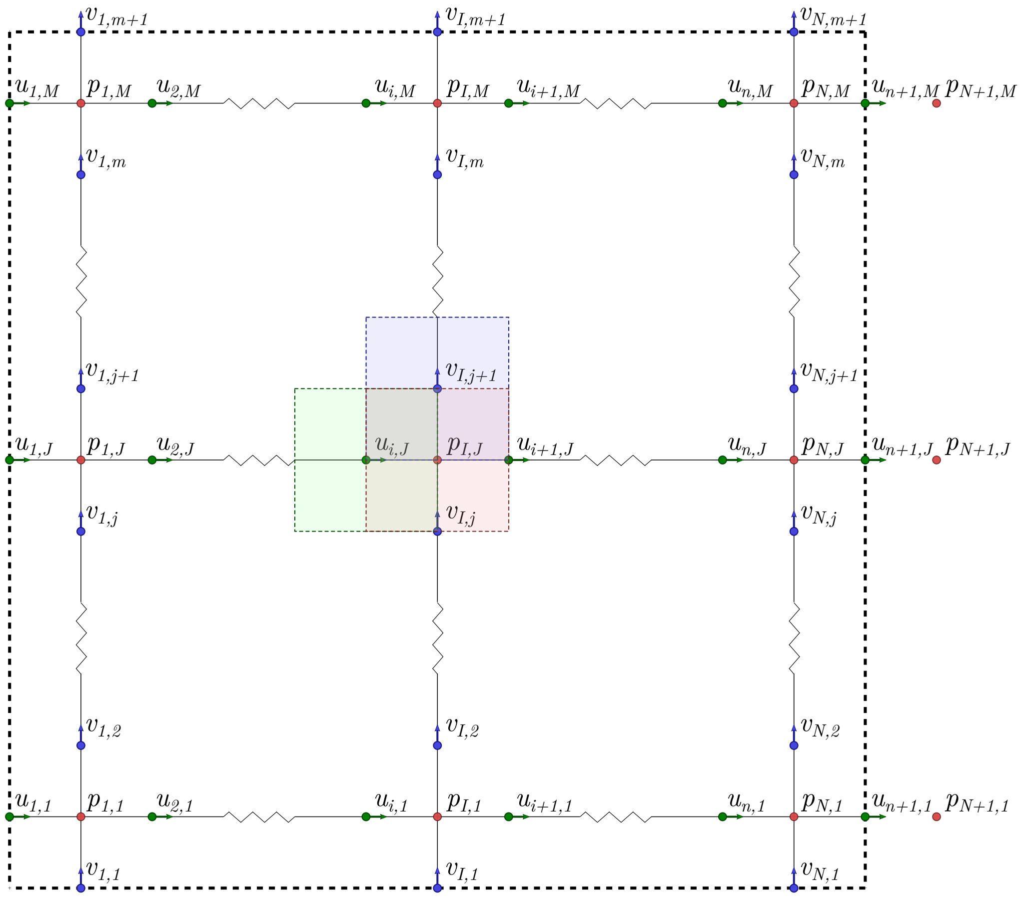


Figure 2.6: Staggered grid in two dimensions showing the locations of the nodes, indices and control volumes for u, v and p.

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 descrtibed 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.

$$\phi = \phi^* + \phi' \tag{2.2.6}$$

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.

$$p = p^* + p' \tag{2.2.7}$$
$$u = u^* + u' \tag{2.2.8}$$
$$v = v^* + v' \tag{2.2.9}$$

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.

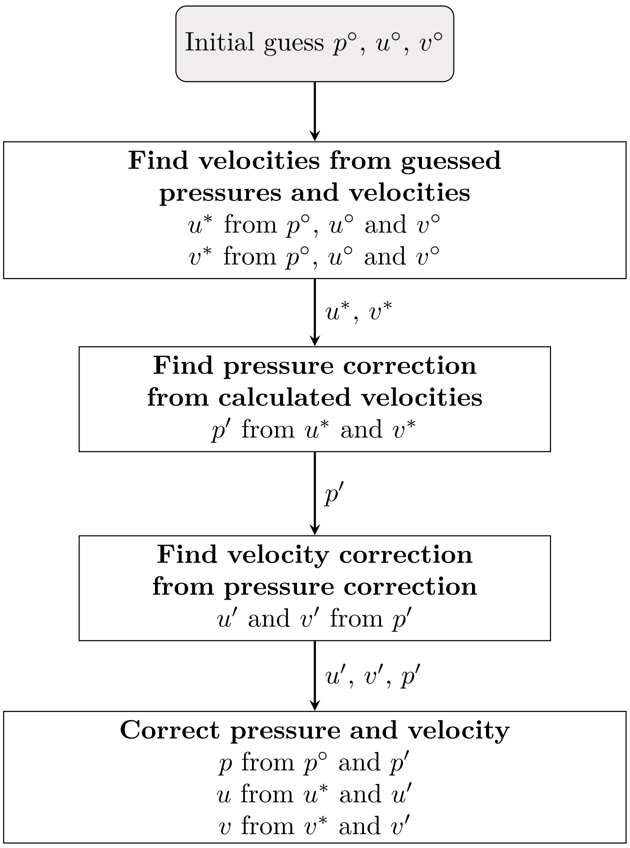


Figure 2.7: Correction cycle in the SIMPLE-algorithm

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.

$$u_{i,J} = u^*_{i,J} - \frac{A_x}{a^{centre}_{i,J}} \left(p'_{I,J} - p'_{I-1,J} \right) \tag{2.2.10}$$

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

$$u'_{i,J} = -\frac{A_x}{a^{centre}_{i,J}} \left(p'_{I,J} - p'_{I-1,J} \right) \tag{2.2.11}$$

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).

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

with

$$\nu_{I,J} = \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} + \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} \tag{2.2.13}$$

$$\nu_{I+1,J} = -\frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} \tag{2.2.14}$$

$$\nu_{I-1,J} = -\frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} \tag{2.2.15}$$

$$\nu_{I,J+1} = -\frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} \tag{2.2.16}$$

$$\nu_{I,J-1} = -\frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} \tag{2.2.17}$$

$$\beta_{I,J} = - A_x F^c_{x,e} + A_x F^c_{x,w} - A_y F^c_{y,n} + A_y F^c_{y,s} \tag{2.2.18}$$

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.

$$p^{new} = p^{\circ} + \alpha_p p' \tag{2.2.19}$$
$$u^{new} = \alpha_u (u^* + u') + (1 - \alpha_u) u^* \tag{2.2.20}$$
$$v^{new} = \alpha_v (v^* + v') + (1 - \alpha_v) v^* \tag{2.2.21}$$

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).

$$\alpha_u + \alpha_p = 1 \tag{2.2.22}$$

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.

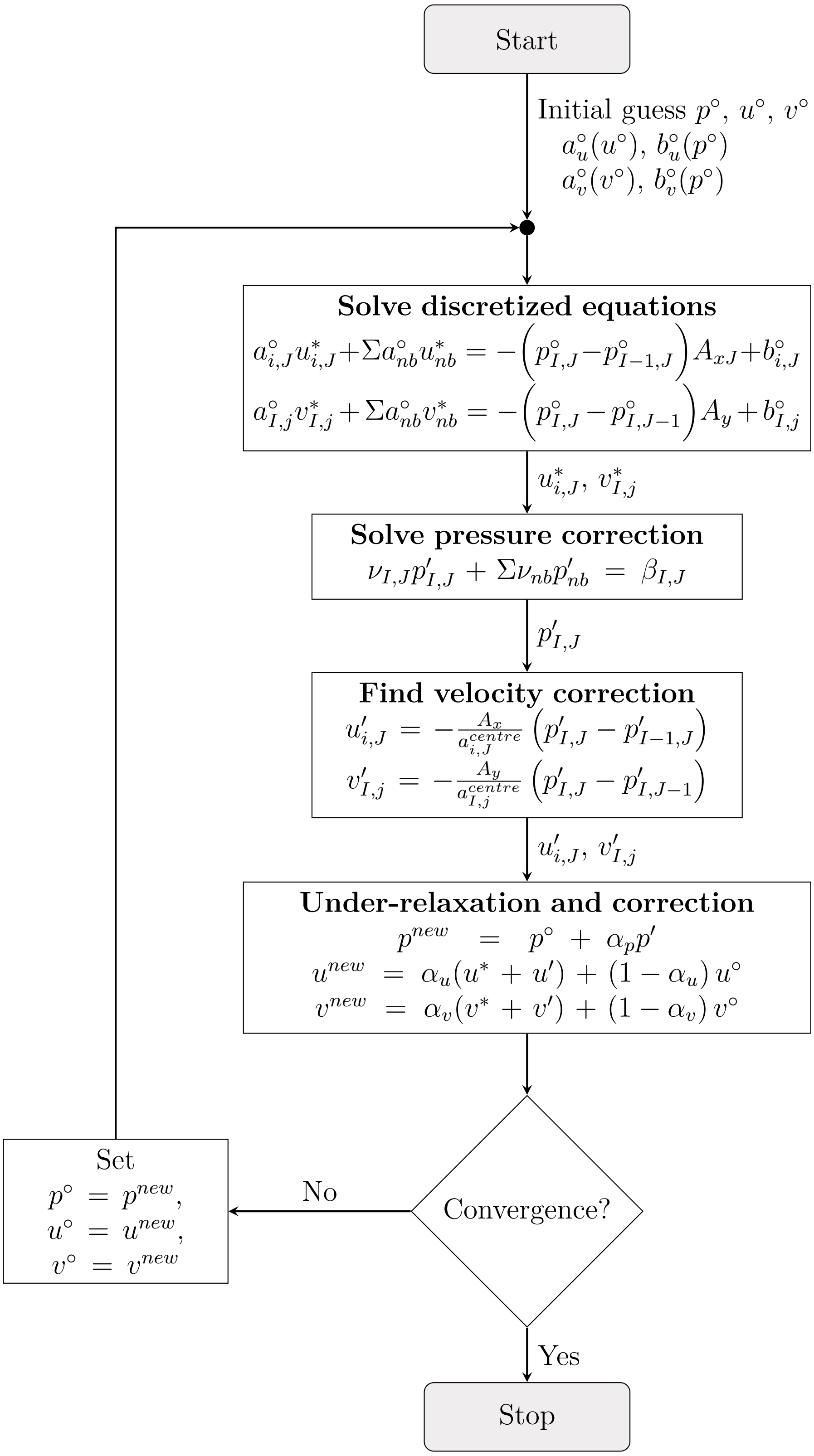


Figure 2.8: Visualisation of the SIMPLE-algorithm and the implemented procedure in `MATLAB`

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 dimensonless 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].

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

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.

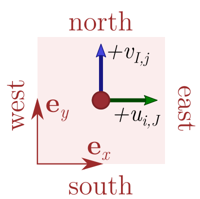


Figure 2.9: Scematic representation of the positive flow direction for the velocity components, as well as a representation of the orientation of the directions *west*, *east*, *north* and *south*.

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.

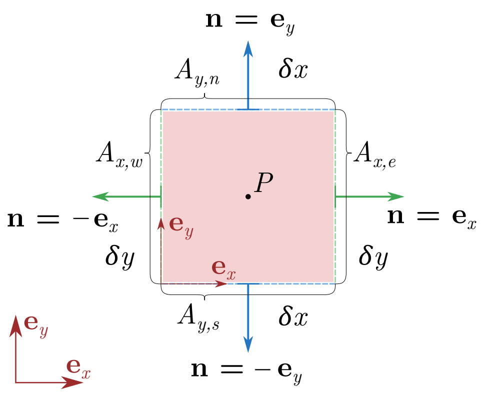


Figure 2.10: Control volume around computational node P with labels for the width δx and height δy of the control volume as well as the normal vectors $\mathbf{n}$ and the cross-sectional areas A_x and A_y. The unit vectors $\mathbf{e}_x$ and $\mathbf{e}_y$ of the coordinate system are also shown.

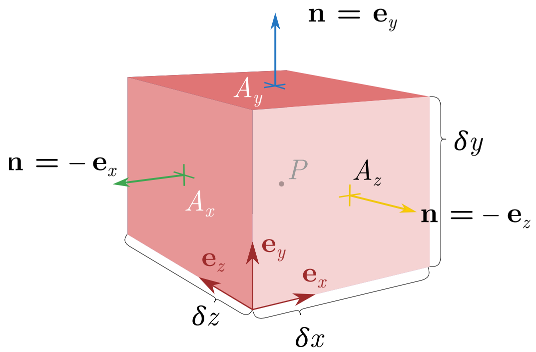


Figure 2.11: The control volume in figure 2.10 seen from a different angle and with labels in all three dimensions.

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).

$$u(j, i) = u(i \cdot (j-1) + i) \tag{2.4.1}$$

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

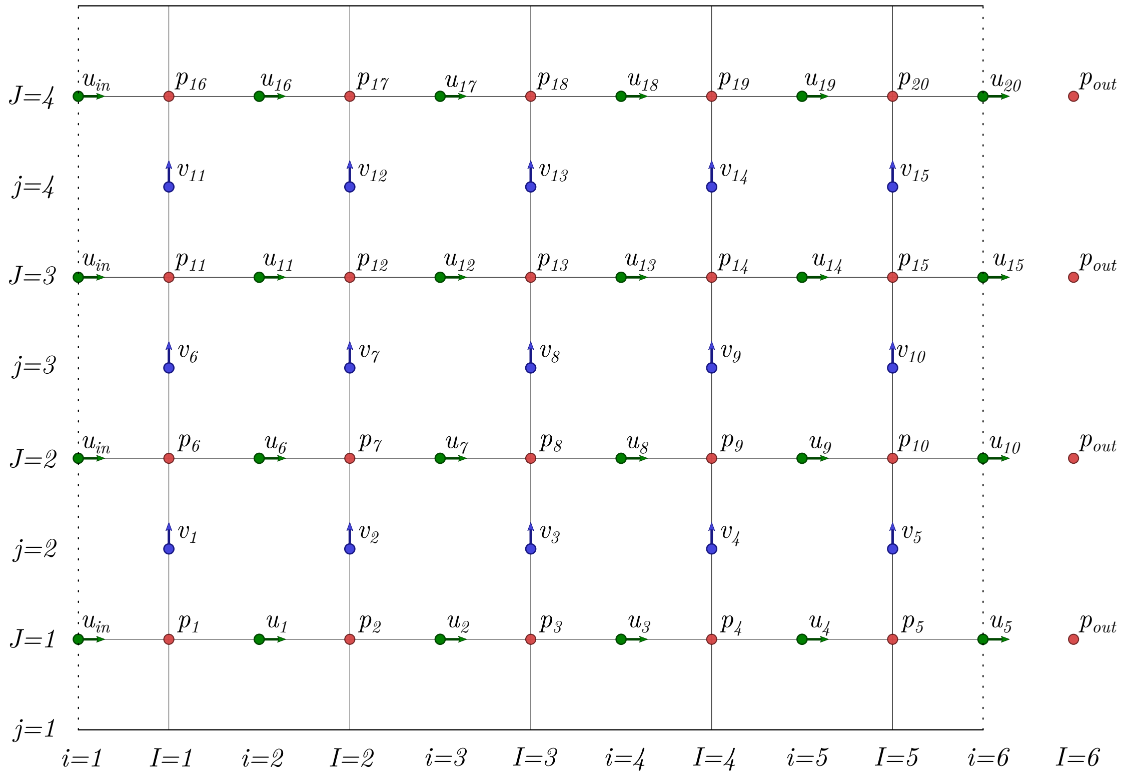


Figure 2.12: Example of a globally indexed system of node points.

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 $\hat{\chi}$.

$$\hat{\chi} = \frac{\chi}{\overline{\chi}} \tag{2.5.1}$$

where $\overline{\chi}$ 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).

$$\hat{\nabla} \cdot \left(\hat{\rho}\hat{\mathbf{u}}\right) = 0 \tag{2.5.2}$$

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].

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

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.

$$a_{i-1}u_{i-1} + a_i u_i + a_{i+1}u_{i+1} = b_i \tag{2.6.1}$$

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).

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

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.

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

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

```
u = A\b
```
(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).

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

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).

$$\int_{A} \mathbf{n} \cdot \Big(\rho \mathbf{u}\Big)\, dA = 0 \tag{3.1.2}$$

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).

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

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).

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

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).

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

$\mathbf{n} \cdot (\rho \mathbf{u})$ 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).

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

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).

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

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.

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

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

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

$$F_x = \rho u \qquad F_y = \rho v \tag{3.2.8}$$

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).

$$F_{x,e} = \rho \frac{u_{i,J} + u_{i+1,J}}{2} \tag{3.2.9}$$

$$F_{x,w} = \rho \frac{u_{i-1,J} + u_{i,J}}{2} \tag{3.2.10}$$

$$F_{x,n} = \rho \frac{v_{I-1,j+1} + v_{I,j+1}}{2} \tag{3.2.11}$$

$$F_{x,s} = \rho \frac{v_{I-1,j} + v_{I,j}}{2} \tag{3.2.12}$$

$$F_{y,e} = \rho \frac{u_{i+1,J-1} + u_{i+1,J}}{2} \tag{3.2.13}$$

$$F_{y,w} = \rho \frac{u_{i,J-1} + u_{i,J}}{2} \tag{3.2.14}$$

$$F_{y,n} = \rho \frac{v_{I,j} + v_{I,j+1}}{2} \tag{3.2.15}$$

$$F_{y,s} = \rho \frac{v_{I,j-1} + v_{I,j}}{2} \tag{3.2.16}$$

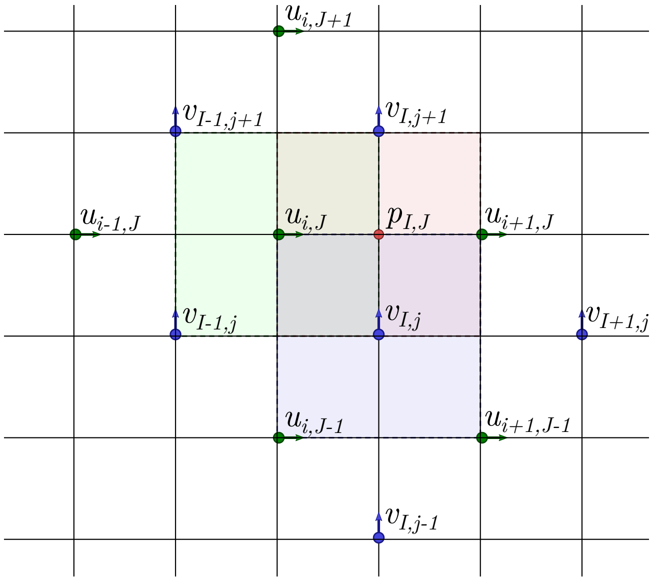


Figure 3.1: Node points with indices used in the expressions for the convective mass flux F.

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).

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

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

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

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

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.

$$F_{x,e} u_e A_x - F_{x,w} u_w A_x + F_{y,n} u_n A_y - F_{y,s} u_s A_y = \mathbf{RHS} \qquad (3.2.25)$$

$$F_{x,e} v_e A_x - F_{x,w} v_w A_x + F_{y,n} v_n A_y - F_{y,s} v_s A_y = \mathbf{RHS} \qquad (3.2.26)$$

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.

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

with

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

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

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.

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

with

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

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

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.

$$\mathbf{LHS} = -\nabla p - \frac{\partial \boldsymbol{\sigma}_x}{\partial x} - \frac{\partial \boldsymbol{\sigma}_y}{\partial y} \tag{3.2.33}$$

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.

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

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

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).

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

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

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).

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

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

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.

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

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

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 $v_{I,j}$.

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

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

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

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

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

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

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

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

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

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

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

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).

$$D_x = \frac{\mu}{\delta x} \qquad D_y = \frac{\mu}{\delta y} \tag{3.2.54}$$

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.

$$\begin{aligned}\mathbf{LHS} = &-\Big(p_{I,J} - p_{I-1,J}\Big)A_x + D_xA_x\Big(u_{i+1,J} - u_{i,J}\Big) - D_xA_x\Big(u_{i,J} - u_{i-1,J}\Big)\\ &+ D_yA_y\Big(u_{i,J+1} - u_{i,J}\Big) - D_yA_y\Big(u_{i,J} - u_{i,J-1}\Big)\end{aligned} \tag{3.2.55}$$

$$\begin{aligned}\mathbf{LHS} = &-\Big(p_{I,J} - p_{I,J-1}\Big)A_y + D_xA_x\Big(v_{I+1,j} - v_{I,j}\Big) - D_xA_x\Big(v_{I,j} - v_{I-1,j}\Big)\\ &+ D_yA_y\Big(v_{I,j+1} - v_{I,j}\Big) - D_yA_y\Big(v_{I,j} - v_{I,j-1}\Big)\end{aligned} \tag{3.2.56}$$

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).

$$a_{i,J}u_{i,J} + a_{i+1,J}u_{i+1,J} + a_{i-1,J}u_{i-1,J} + a_{i,J+1}u_{i,J+1} + a_{i,J-1}u_{i,J-1} = b_{i,J} \tag{3.2.57}$$

with

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

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

$$a_{I,j}v_{I,j} + a_{I+1,j}v_{I+1,j} + a_{I-1,j}v_{I-1,j} + a_{I,j+1}v_{I,j+1} + a_{I,j-1}v_{I,j-1} = b_{I,j} \tag{3.2.59}$$

with

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

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).

$$
\begin{aligned}
a_{i,J}(u_{i,J} - u^*_{i,J}) + a_{i+1,J}(u_{i+1,J} - u^*_{i+1,J}) + a_{i-1,J}(u_{i-1,J} - u^*_{i-1,J}) \\
+ a_{i,J+1}(u_{i,J+1} - u^*_{i,J+1}) + a_{i,J-1}(u_{i,J-1} - u^*_{i,J-1}) \\
= \Big(- p_{I,J} + p_{I-1,J} + p^\circ_{I,J} - p^\circ_{I-1,J}\Big)A_x + \cancel{b^\rho_{i,J} - b^\rho_{i,J}}
\end{aligned}
\tag{3.3.1}
$$

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 $u_{i,J}$.

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

$a^{centre}_{i,J}$ 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.

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

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

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

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

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).

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

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].

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

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

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

with

$$\begin{aligned}
\nu_{I,J} &= \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} + \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} \\
\nu_{I+1,J} &= - \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} \\
\nu_{I-1,J} &= - \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} \\
\nu_{I,J+1} &= - \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} \\
\nu_{I,J-1} &= - \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} \\
\beta_{I,J} &= - A_x \rho u^*_{x,e} + A_x \rho u^*_{x,w} - A_y \rho u^*_{y,n} + A_y \rho u^*_{y,s}
\end{aligned} \tag{3.3.9}$$

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.

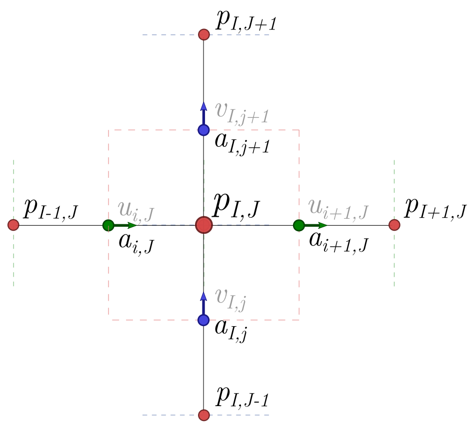


Figure 3.2: Shape of pressure correction equation "molecule" in two dimensions.

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].

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

A diacritic *circumflex* ˆ 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.

$$\tilde{p} = p - p_{out} \tag{3.4.2}$$

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).

$$\hat{\mathbf{u}} = \frac{\mathbf{u}}{u_{in}} \tag{3.4.3}$$

$$\hat{\tilde{p}} = \frac{\tilde{p}}{\overline{p}} \tag{3.4.4}$$

$$\hat{\mu} = \frac{\mu}{\mu_{in}} = \frac{\mu}{\mu} \tag{3.4.5}$$

$$\hat{\rho} = \frac{\rho}{\rho_{in}} = \frac{\rho}{\rho} \tag{3.4.6}$$

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* $^{-}$ is used to signify the scale for a variable. The pressure scale $\overline{p}$ is given by equation (3.4.7) [16].

$$\overline{p} = \rho u_{in}^2 \tag{3.4.7}$$

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.

$$\hat{x} = \frac{x}{D_{hyd}} \tag{3.4.8}$$

$$d\hat{x} = \frac{dx}{D_{hyd}} \tag{3.4.9}$$

$$\delta\hat{x} = \frac{\delta x}{D_{hyd}} \tag{3.4.10}$$

$$\frac{\partial}{\partial \hat{x}} = D_{hyd}\frac{\partial}{\partial x} \tag{3.4.11}$$

$$\hat{y} = \frac{y}{D_{hyd}} \tag{3.4.12}$$

$$d\hat{y} = \frac{dy}{D_{hyd}} \tag{3.4.13}$$

$$\delta\hat{y} = \frac{\delta y}{D_{hyd}} \tag{3.4.14}$$

$$\frac{\partial}{\partial \hat{y}} = D_{hyd}\frac{\partial}{\partial y} \tag{3.4.15}$$

$$\hat{z} = \frac{z}{D_{hyd}} \tag{3.4.16}$$

$$d\hat{z} = \frac{dz}{D_{hyd}} \tag{3.4.17}$$

$$\delta\hat{z} = \frac{\delta z}{D_{hyd}} \tag{3.4.18}$$

$$\frac{\partial}{\partial \hat{z}} = D_{hyd}\frac{\partial}{\partial z} \tag{3.4.19}$$

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.

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

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

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

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

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

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

3.4.1.3 Operators, tensors

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

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

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

$$\hat{\nabla} = D_{hyd}\nabla \tag{3.4.26}$$

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).

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

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

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

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

$$\hat{\boldsymbol{\sigma}} = \frac{\boldsymbol{\sigma}}{\overline{\sigma}} \tag{3.4.30}$$

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).

$$\hat{\sigma}_{\hat{x}\hat{x}} = -\frac{1}{\overline{\sigma}}2\mu\frac{\partial u}{\partial x} = -\frac{1}{\overline{\sigma}}\frac{\mu u_{in}}{D_{hyd}}2\hat{\mu}\frac{\partial \hat{u}}{\partial \hat{x}} \tag{3.4.31}$$

$$\hat{\sigma}_{\hat{y}\hat{y}} = -\frac{1}{\overline{\sigma}}2\mu\frac{\partial v}{\partial y} = -\frac{1}{\overline{\sigma}}\frac{\mu u_{in}}{D_{hyd}}2\hat{\mu}\frac{\partial \hat{v}}{\partial \hat{y}} \tag{3.4.32}$$

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

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

$$\overline{\sigma} = \frac{\mu u_{in}}{D_{hyd}} \tag{3.4.34}$$

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

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

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

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

3.4.2 Variables as Functions of their Dimensionless Form

All varibles, 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).

$$\mathbf{u} = u_{in}\hat{\mathbf{u}} \tag{3.4.38}$$

$$\tilde{p} = \rho u_{in}^2 \hat{\tilde{p}} \tag{3.4.39}$$

$$\mu = \mu\hat{\mu} \tag{3.4.40}$$

$$\rho = \rho\hat{\rho} \tag{3.4.41}$$

$$\delta x = D_{hyd}\ \delta\hat{x} \tag{3.4.42}$$

$$\delta y = D_{hyd}\ \delta\hat{y} \tag{3.4.43}$$

$$\frac{\partial}{\partial x} = \frac{1}{D_{hyd}}\frac{\partial}{\partial \hat{x}} \tag{3.4.44}$$

$$\frac{\partial}{\partial y} = \frac{1}{D_{hyd}}\frac{\partial}{\partial \hat{y}} \tag{3.4.45}$$

$$\nabla = \frac{1}{D_{hyd}}\hat{\nabla} \tag{3.4.46}$$

$$A_x = D_{hyd}^2\ \hat{A}_x \tag{3.4.47}$$

$$A_y = D_{hyd}^2\ \hat{A}_y \tag{3.4.48}$$

$$dA = D_{hyd}^2\ d\hat{A} \tag{3.4.49}$$

$$V = D_{hyd}^3\ \hat{V} \tag{3.4.50}$$

$$dV = D_{hyd}^3\ d\hat{V} \tag{3.4.51}$$

$$\boldsymbol{\sigma} = \overline{\sigma}\hat{\boldsymbol{\sigma}} \tag{3.4.52}$$

$$\sigma_{xx} = -\frac{\mu u_{in}}{D_{hyd}}2\hat{\mu}\frac{\partial \hat{u}}{\partial \hat{x}} \tag{3.4.53}$$

$$\sigma_{yy} = -\frac{\mu u_{in}}{D_{hyd}}2\hat{\mu}\frac{\partial \hat{v}}{\partial \hat{y}} \tag{3.4.54}$$

$$\sigma_{xy} = -\frac{\mu u_{in}}{D_{hyd}}\hat{\mu}\left[\frac{\partial \hat{u}}{\partial \hat{x}} + \frac{\partial \hat{v}}{\partial \hat{y}}\right] \tag{3.4.55}$$

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).

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

Integration over the dimensionless control volume $\hat{CV}$ 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.

$$\int_{\hat{CV}} \frac{1}{D_{hyd}} \hat{\nabla} \cdot \left(\rho u_{in} \hat{\rho} \hat{\mathbf{u}} \right) \, d\hat{V} = 0 \tag{3.4.57}$$

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

$$\int_{\hat{A} \mathbf{n} \cdot \left(\hat{\rho} \hat{\mathbf{u}} \right) \, d\hat{A}} = 0 \tag{3.4.59}$$

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

$$\hat{F}^c_{x,e} \hat{A}_{x,e} - \hat{F}^c_{x,w} \hat{A}_{x,w} + \hat{F}^c_{y,n} \hat{A}_{y,n} - \hat{F}^c_{y,s} \hat{A}_{y,s} = 0 \tag{3.4.60}$$

with

$$\hat{F}^c_x = \hat{\rho} \hat{u} \qquad \hat{F}^c_y = \hat{\rho} \hat{v} \tag{3.4.61}$$

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).

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

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

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

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).

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

3.4.4.1 Left Hand Side

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

$$\int_{CV} \hat{\nabla} \cdot (\hat{\rho} \hat{\mathbf{u}} \hat{\mathbf{u}}) \, d\hat{V} = \mathbf{RHS} \tag{3.4.65}$$

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

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).

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

with

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

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

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).

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

with

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

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

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).

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

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.

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

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

Equations (3.4.74) and (3.4.75) can then be integrated over the dimensionless control volume $\hat{CV}$. 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).

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

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

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).

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

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)

$$\textbf{LHS} = -\left(\hat{\hat{p}}_{I,J} - \hat{\hat{p}}_{I-1,J}\right)\hat{A}_x + \hat{D}_x\hat{A}_x\left(\hat{u}_{i+1,J} - \hat{u}_{i,J}\right) - \hat{D}_x\hat{A}_x\left(\hat{u}_{i,J} - \hat{u}_{i-1,J}\right) + \hat{D}_y\hat{A}_y\left(\hat{u}_{i,J+1} - \hat{u}_{i,J}\right) - \hat{D}_y\hat{A}_y\left(\hat{u}_{i,J} - \hat{u}_{i,J-1}\right) \quad (3.4.79)$$

$$\textbf{LHS} = -\left(\hat{\hat{p}}_{I,J} - \hat{\hat{p}}_{I,J-1}\right)\hat{A}_y + \hat{D}_x\hat{A}_x\left(\hat{v}_{I+1,j} - \hat{v}_{I,j}\right) - \hat{D}_x\hat{A}_x\left(\hat{v}_{I,j} - \hat{v}_{I-1,j}\right) + \hat{D}_y\hat{A}_y\left(\hat{v}_{I,j+1} - \hat{v}_{I,j}\right) - \hat{D}_y\hat{A}_y\left(\hat{v}_{I,j} - \hat{v}_{I,j-1}\right) \quad (3.4.80)$$

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

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

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

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

3.4.5 Dimensionless SIMPLE-Equations

The discretised dimensionlesss 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 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 $v_{I,j}$.

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

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

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 $\hat{\tilde{p}}$ following equation (3.4.87).

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

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).

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

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.

$$\rho = 997 \left[\text{kg/m}^3 \right] \text{ at } 25°\text{C} \qquad \mu = 8.90 \cdot 10^{-4} \; [\text{ Pa}\cdot \text{ s }] \tag{4.1.1}$$

4.1.2 Domain Size

Scematic representations of the doimains 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).

$$\text{Expansion ratio} = \frac{H}{h} \tag{4.1.2}$$

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 heigth	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.

$$\hat{u}_{guess} = \hat{u}_{in} = 1 \tag{4.3.1}$$
$$\hat{v}_{guess} = \hat{v}_{in} = 0 \tag{4.3.2}$$
$$\hat{\hat{p}}_{guess} = \hat{\hat{p}}_{out} = 0 \tag{4.3.3}$$

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.

$$\hat{u}_{guess}^{narrow} = \hat{u}_{in} = 1 \tag{4.3.4}$$
$$\hat{v}_{guess}^{narrow} = \hat{v}_{in} = 0 \tag{4.3.5}$$

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

$$\hat{u}_{guess}^{narrow} = \hat{u}_{max} \tag{4.3.6}$$
$$\hat{v}_{guess}^{narrow} = \hat{v}_{in} = 0 \tag{4.3.7}$$

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.

$$\hat{u}_{guess}^{wide} = \hat{u}_{guess}^{narrow} \frac{M_{narrow}}{M_{total}} \tag{4.3.8}$$
$$\hat{v}_{guess}^{wide} = \hat{v}_{guess}^{narrow} \frac{m_{narrow}}{m_{total}} \tag{4.3.9}$$

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).

$$\hat{\hat{p}}_{guess} = \hat{\hat{p}}_{out} = 0 \tag{4.3.10}$$

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 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.

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

$$\hat{F}_{y,w} = \hat{\rho}\hat{u}_{in} \tag{4.4.2}$$

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.

$$\hat{a}_P\hat{u}_P + \hat{a}_E\hat{u}_E + \hat{a}_N\hat{u}_N + \hat{a}_S\hat{u}_S = \hat{b}_P \tag{4.4.3}$$

with

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

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

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

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

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

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.

$$\hat{a}_P\hat{v}_P + \hat{a}_E\hat{v}_E + \hat{a}_N\hat{v}_N + \hat{a}_S\hat{v}_S = \hat{b}_P \tag{4.4.9}$$

with

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

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

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

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

$$\hat{b}_P = -\Big(\hat{\tilde{p}}_P - \hat{\tilde{p}}_S\Big)\hat{A}_y \tag{4.4.14}$$

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).

$$\hat{\nu}_P\hat{\tilde{p}}'_P + \hat{\nu}_E\hat{\tilde{p}}'_E + \hat{\nu}_N\hat{\tilde{p}}'_N + \hat{\nu}_S\hat{\tilde{p}}'_S = \hat{\beta}_P \tag{4.4.15}$$

with

$$\hat{\nu}_P = -\hat{\nu}_E - \hat{\nu}_N - \hat{\nu}_S \tag{4.4.16}$$

$$\hat{\nu}_E = -\frac{\hat{\rho}\hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \tag{4.4.17}$$

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

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

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

4.4.2 Outlet

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

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.

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

$$\tag{4.4.22}$$

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].

$$\hat{a}_P\hat{u}_P + \hat{a}_W\hat{u}_W + \hat{a}_N\hat{u}_N + \hat{a}_S\hat{u}_S = \hat{b}_P \tag{4.4.23}$$

with

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

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

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

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

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

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].

$$\hat{a}_P\hat{v}_P + \hat{a}_W\hat{v}_W + \hat{a}_N\hat{v}_N + \hat{a}_S\hat{v}_S = \hat{b}_P \tag{4.4.29}$$

with

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

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

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

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

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

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).

$$\hat{\nu}_P\hat{\tilde{p}}'_P + \hat{\nu}_W\hat{\tilde{p}}'_W + \hat{\nu}_N\hat{\tilde{p}}'_N + \hat{\nu}_S\hat{\tilde{p}}'_S = \hat{\beta}_P \tag{4.4.35}$$

with

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

$$\hat{\nu}_W = -\frac{\hat{\rho}\hat{A}_x^2}{\hat{a}_{u,P}^{centre}} \tag{4.4.37}$$

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

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

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

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).

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

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.

$$\hat{F}_{x,n} = 0 \tag{4.4.42}$$

$$\hat{F}_{y,n} = \hat{\rho}\hat{v}_P \tag{4.4.43}$$

$$\hat{F}_{x,s} = 0 \tag{4.4.44}$$

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

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.

$$\mathbf{LHS} = -\left.\frac{\partial\hat{\hat{p}}}{\partial\hat{x}}\right|_P \delta\hat{x}\hat{A}_x + \frac{1}{Re}\hat{\mu}\left.\frac{\partial\hat{u}}{\partial\hat{x}}\right|_e \hat{A}_{x,e} - \frac{1}{Re}\hat{\mu}\left.\frac{\partial\hat{u}}{\partial\hat{x}}\right|_w \hat{A}_{x,w} + \frac{1}{Re}\hat{\mu}\left.\frac{\partial\hat{u}}{\partial\hat{y}}\right|_n \hat{A}_{y,n} - \frac{1}{Re}\hat{\mu}\left.\frac{\partial\hat{u}}{\partial\hat{y}}\right|_s \hat{A}_{y,s} \tag{4.4.46}$$

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.

$$\left.\frac{\partial\hat{u}}{\partial\hat{y}}\right|_n = \frac{\hat{u}_{wall} - \hat{u}_P}{\delta\hat{y}/2} \tag{4.4.47}$$

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].

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

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

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).

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

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).

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

with

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

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

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

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

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

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

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

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).

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

with

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

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

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

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

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

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).

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_W \hat{v}_W + \hat{a}_S \hat{v}_S = \hat{b}_P \tag{4.4.64}$$

with

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

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

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

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

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

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

$$\hat{a}_P \hat{v}_P + \hat{a}_E \hat{v}_E + \hat{a}_W \hat{v}_W + \hat{a}_N \hat{v}_N = \hat{b}_P \tag{4.4.70}$$

with

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

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

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

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

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

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.

$$\hat{\nu}_P \hat{\tilde{p}}'_P + \hat{\nu}_E \hat{\tilde{p}}'_E + \hat{\nu}_W \hat{\tilde{p}}'_W + \hat{\nu}_S \hat{\tilde{p}}'_S = \hat{\beta}_P \tag{4.4.76}$$

with

$$\hat{\nu}_P = -\hat{\nu}_E - \hat{\nu}_W - \hat{\nu}_S \tag{4.4.77}$$

$$\hat{\nu}_E = -\frac{\hat{\rho}\hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \tag{4.4.78}$$

$$\hat{\nu}_W = -\frac{\hat{\rho}\hat{A}_x^2}{\hat{a}_{u,P}^{centre}} \tag{4.4.79}$$

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

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

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.

$$\hat{\nu}_P \hat{\tilde{p}}'_P + \hat{\nu}_E \hat{\tilde{p}}'_E + \hat{\nu}_W \hat{\tilde{p}}'_W + \hat{\nu}_N \hat{\tilde{p}}'_N = \hat{\beta}_P \tag{4.4.82}$$

with

$$\hat{\nu}_P = -\hat{\nu}_E - \hat{\nu}_W - \hat{\nu}_N \tag{4.4.83}$$

$$\hat{\nu}_E = -\frac{\hat{\rho}\hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \tag{4.4.84}$$

$$\hat{\nu}_W = -\frac{\hat{\rho}\hat{A}_x^2}{\hat{a}_{u,P}^{centre}} \tag{4.4.85}$$

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

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

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.

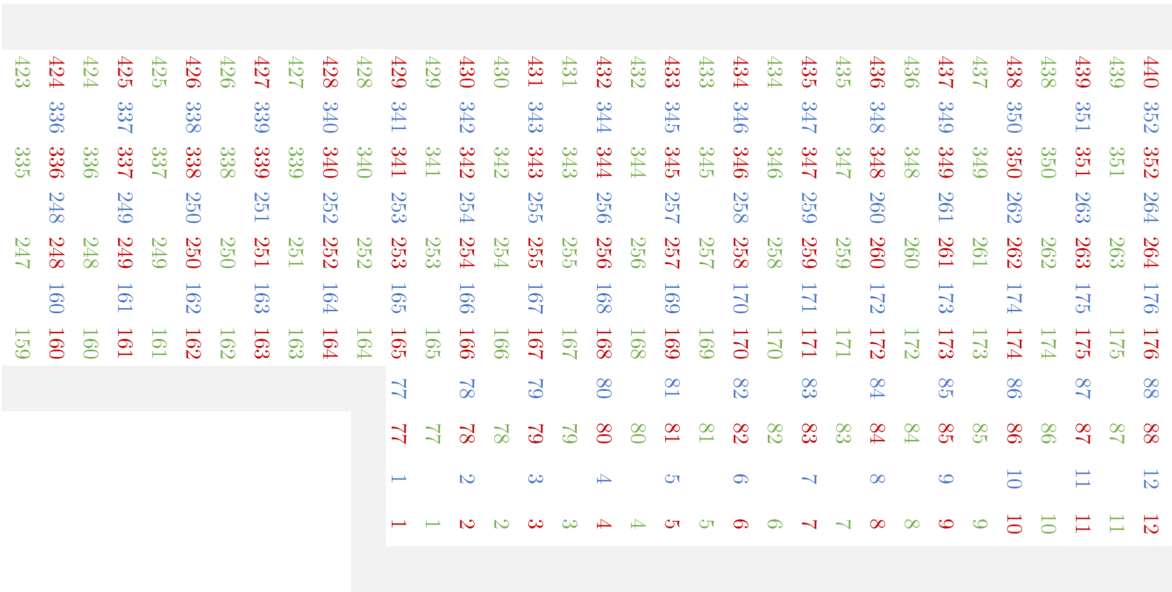


Figure 4.1: Global indexing in the backwards facing step domains.

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.

$$\hat{a}_P\hat{u}_P + \hat{a}_E\hat{u}_E + \hat{a}_N\hat{u}_N + \hat{a}_S\hat{u}_S = \hat{b}_P \tag{4.5.1}$$

with

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

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

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

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

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

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.

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

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

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

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.

$$\hat{a}_P\hat{v}_P + \hat{a}_E\hat{v}_E + \hat{a}_N\hat{v}_N + \hat{a}_S\hat{v}_S = \hat{b}_P \tag{4.5.9}$$

with

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

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

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

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

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

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.

$$\hat{\nu}_P\hat{\tilde{p}}'_P + \hat{\nu}_E\hat{\tilde{p}}'_E + \hat{\nu}_N\hat{\tilde{p}}'_N + \hat{\nu}_S\hat{\tilde{p}}'_S = \hat{\beta}_P \tag{4.5.15}$$

with

$$\hat{\nu}_P = -\hat{\nu}_E - \hat{\nu}_N - \hat{\nu}_S \tag{4.5.16}$$

$$\hat{\nu}_E = -\frac{\rho\hat{A}_x^2}{\hat{a}_{u,E}^{centre}} \tag{4.5.17}$$

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

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

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

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.

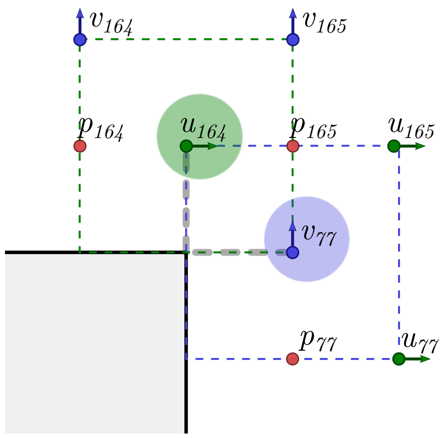


Figure 4.2: Indexed computational points around the backwards facing step.

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)

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

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

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

with

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

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

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

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

$$\hat{b}_P = -\Big(\hat{\tilde{p}}_P - \hat{\tilde{p}}_W\Big)\hat{A}_x \tag{4.5.27}$$

Simuilarly, 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) .

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

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

$$\hat{a}_P\hat{v}_P + \hat{a}_E\hat{v}_E + \hat{a}_N\hat{v}_N + \hat{a}_S\hat{v}_S = \hat{b}_P \tag{4.5.29}$$

with

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

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

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

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

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

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).

$$\mathbf{u} = u_{avg}\hat{\mathbf{u}} \tag{4.6.1}$$

$$\tilde{p} = \rho u_{avg}^2 \hat{\tilde{p}} \tag{4.6.2}$$

$$\mu = \mu\hat{\mu} \tag{4.6.3}$$

$$\rho = \rho\hat{\rho} \tag{4.6.4}$$

$$\delta x = h\ \delta\hat{x} \tag{4.6.5}$$

$$\delta y = h\ \delta\hat{y} \tag{4.6.6}$$

$$\frac{\partial}{\partial x} = \frac{1}{h}\frac{\partial}{\partial\hat{x}} \tag{4.6.7}$$

$$\frac{\partial}{\partial y} = \frac{1}{h}\frac{\partial}{\partial\hat{y}} \tag{4.6.8}$$

$$\nabla = \frac{1}{h}\hat{\nabla} \tag{4.6.9}$$

$$A_x = h^2\ \hat{A}_x \tag{4.6.10}$$

$$A_y = h^2\ \hat{A}_y \tag{4.6.11}$$

$$dA = h^2\ d\hat{A} \tag{4.6.12}$$

$$V = h^3\ \hat{V} \tag{4.6.13}$$

$$dV = h^3\ d\hat{V} \tag{4.6.14}$$

$$\sigma = \overline{\sigma}\hat{\sigma} \tag{4.6.15}$$

$$\sigma_{xx} = -\frac{\mu u_{avg}}{h} 2\hat{\mu}\frac{\partial\hat{u}}{\partial\hat{x}} \tag{4.6.16}$$

$$\sigma_{yy} = -\frac{\mu u_{avg}}{h} 2\hat{\mu}\frac{\partial\hat{v}}{\partial\hat{y}} \tag{4.6.17}$$

$$\sigma_{xy} = -\frac{\mu u_{avg}}{h}\hat{\mu}\left[\frac{\partial\hat{u}}{\partial\hat{x}} + \frac{\partial\hat{v}}{\partial\hat{y}}\right] \tag{4.6.18}$$

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).

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

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

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

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

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

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

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

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.

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

with

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

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

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

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

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

$$\hat{b}_{i,J} = -\Big(\hat{\tilde{p}}_{I,J} - \hat{\tilde{p}}_{I-1,J}\Big)\hat{A}_x \tag{4.6.32}$$

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

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

with

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

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

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

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

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

$$\hat{b}_{I,j} = -\Big(\hat{\tilde{p}}_{I,J} - \hat{\tilde{p}}_{I,J-1}\Big)\hat{A}_y \tag{4.6.39}$$

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

$$\hat{D}_x = \frac{2}{Re}\frac{\hat{\mu}}{\delta\hat{x}} \qquad \hat{D}_y = \frac{2}{Re}\frac{\hat{\mu}}{\delta\hat{y}} \tag{4.6.40}$$

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).

$$\mathbf{U} \cdot u^* - b_u = R_u \tag{4.7.1}$$

$\mathbf{U}$ 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).

$$C_1 = \sqrt{R_u \cdot R_u^T} \tag{4.7.2}$$

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

$$C_2 = \sqrt{R_v \cdot R_v^T} \tag{4.7.3}$$

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)

$$C_3 = \left| \sum \beta \right| \tag{4.7.4}$$

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).

$$C_4 = \max(|u^\circ - u^*|) \tag{4.7.5}$$

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

$$C_5 = \max(|v^\circ - v^*|) \tag{4.7.6}$$

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).

$$C_1 < 10^{-8} \tag{4.7.7}$$
$$C_2 < 10^{-8} \tag{4.7.8}$$
$$C_3 < 10^{-10} \tag{4.7.9}$$
$$C_4 < 10^{-8} \tag{4.7.10}$$
$$C_5 < 10^{-8} \tag{4.7.11}$$

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.

$$\mathbf{u} = u_{in}\hat{\mathbf{u}} \tag{4.8.1}$$

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

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

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

$$p = \rho u_{in}^2 \hat{\tilde{p}} + p_{out} \tag{4.8.3}$$

The pressure correction is obtained by equation (4.8.4).

$$p' = \rho u_{in}^2 \hat{p}' \tag{4.8.4}$$

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).

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

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

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

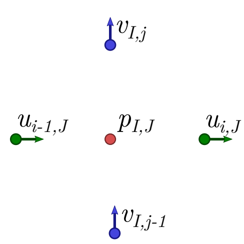


Figure 4.3: The points included in the calculation of velocity for quiver/contour plots.

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 $\delta x/2$a halv 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].

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

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.

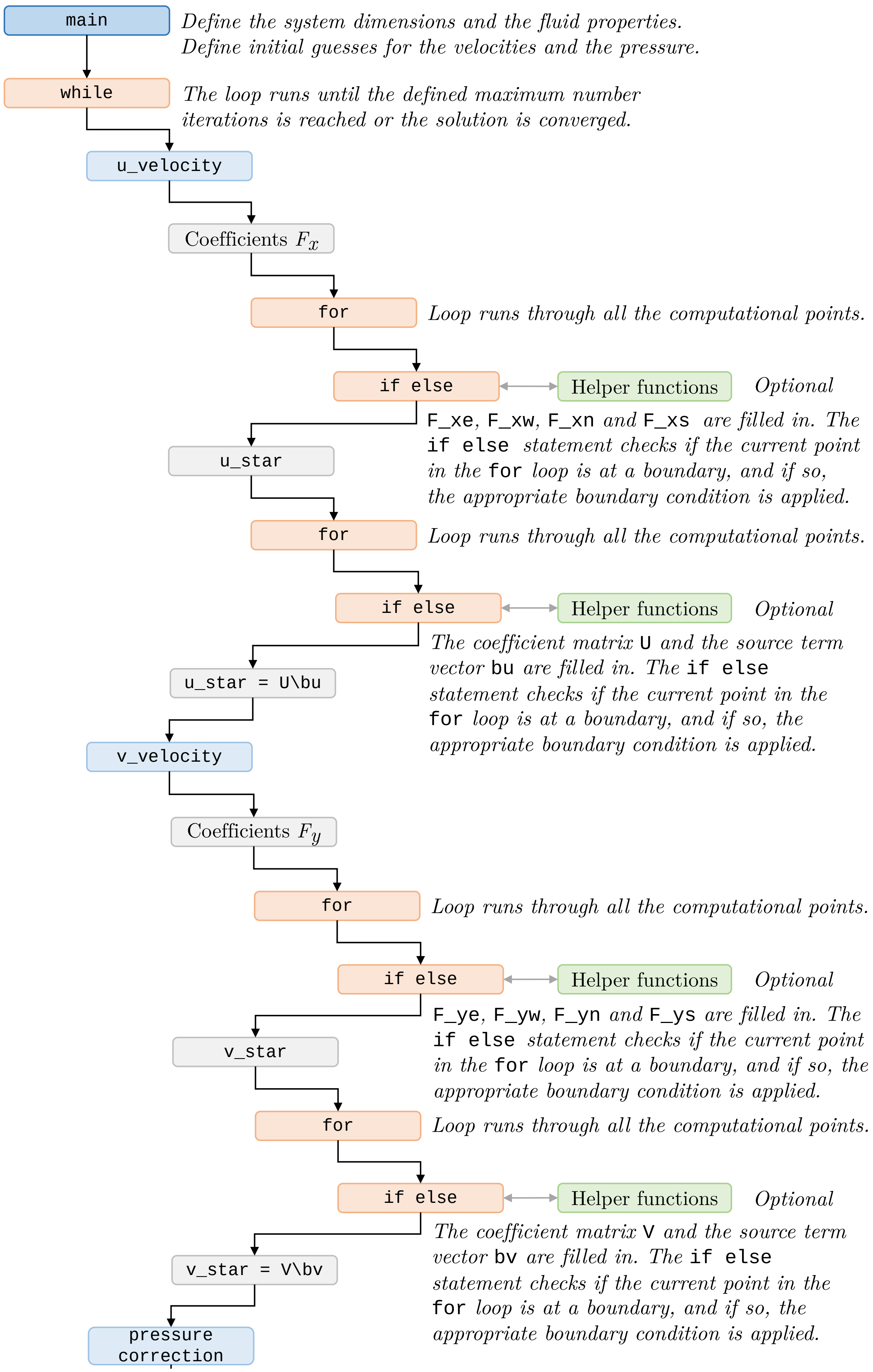
4.9.1.1 Plot Initial Guesses

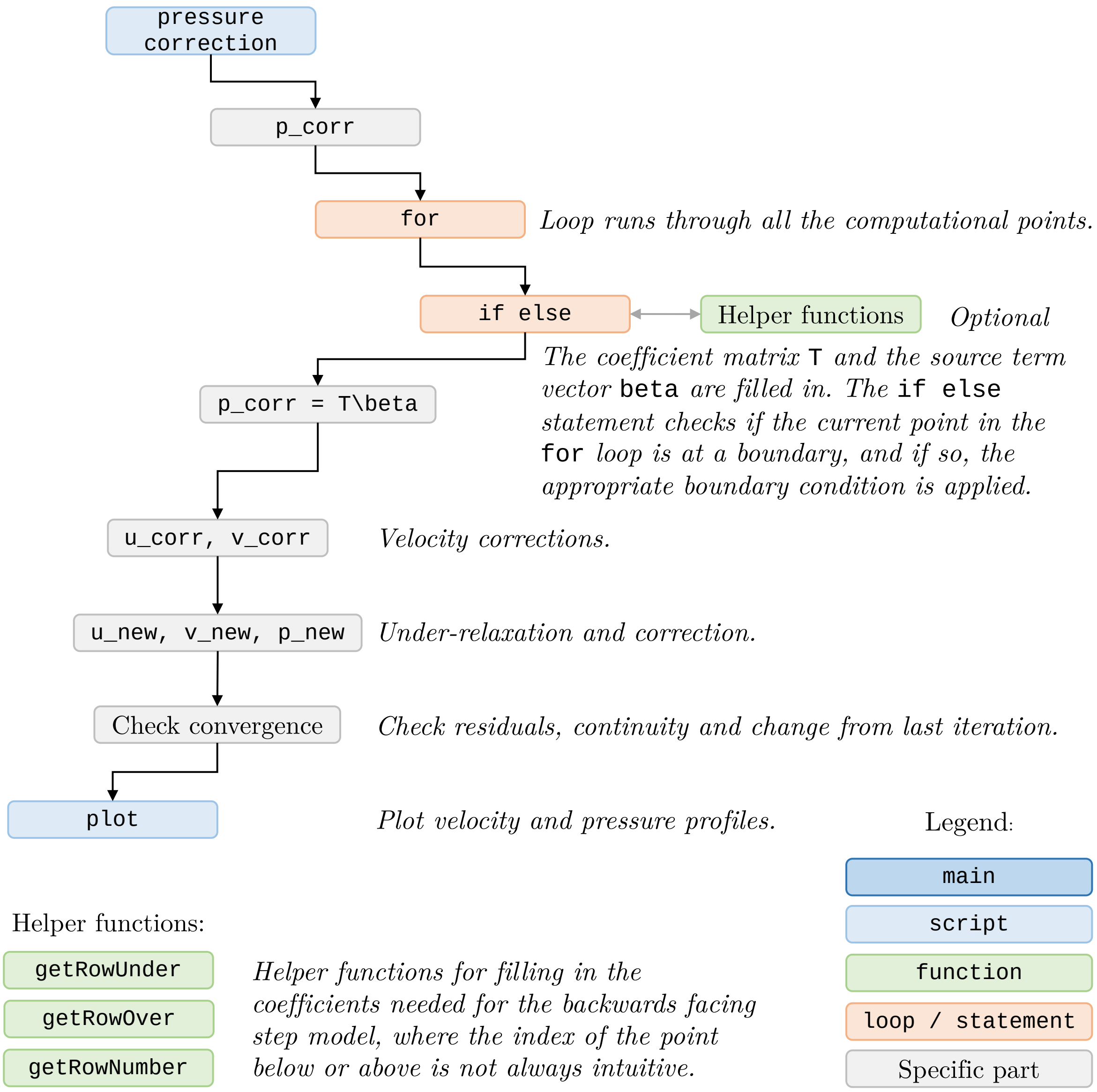
The option `plotInitialProfiles` plots the initial guesses of the velocities and the pressure.

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.


main
Define the system dimensions and the fluid properties.
Define initial guesses for the velocities and the pressure.
while
The loop runs until the defined maximum number iterations is reached or the solution is converged.
u_velocity
Coefficients F_x
for
Loop runs through all the computational points.
if else
Helper functions
Optional
F_xe, F_xw, F_xn and F_xs are filled in. The if else statement checks if the current point in the for loop is at a boundary, and if so, the appropriate boundary condition is applied.
u_star
for
Loop runs through all the computational points.
if else
Helper functions
Optional
The coefficient matrix U and the source term vector bu are filled in. The if else statement checks if the current point in the for loop is at a boundary, and if so, the appropriate boundary condition is applied.
u_star = U\bu
v_velocity
Coefficients F_y
for
Loop runs through all the computational points.
if else
Helper functions
Optional
F_ye, F_yw, F_yn and F_ys are filled in. The if else statement checks if the current point in the for loop is at a boundary, and if so, the appropriate boundary condition is applied.
v_star
for
Loop runs through all the computational points.
if else
Helper functions
Optional
The coefficient matrix V and the source term vector bv are filled in. The if else statement checks if the current point in the for loop is at a boundary, and if so, the appropriate boundary condition is applied.
v_star = V\bv
pressure correction



4.9.1.3 Disable Solution of v-velocity

With the option `solvvel`, 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 came 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.

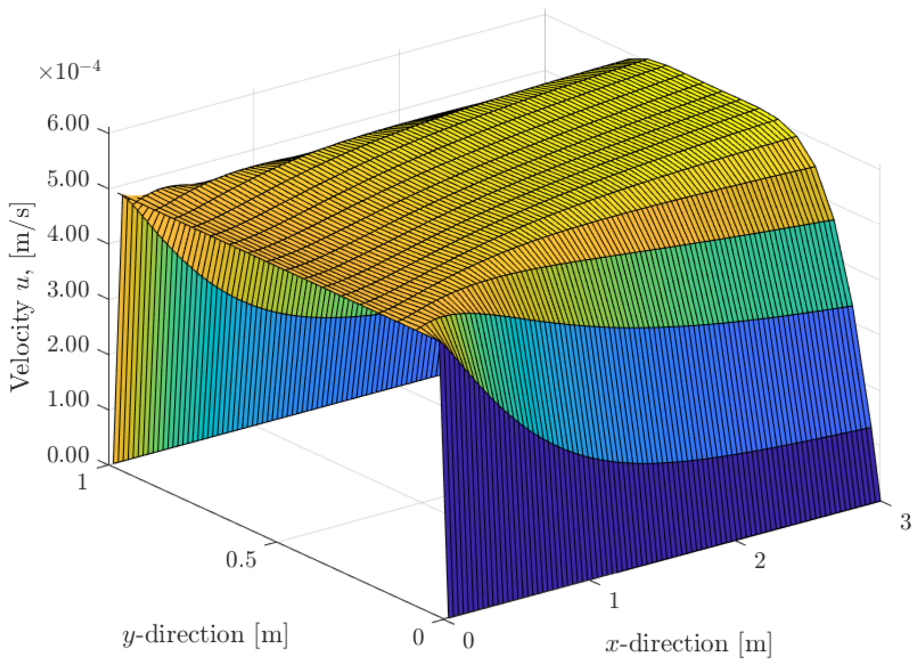


Figure 5.1: u-velocity seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

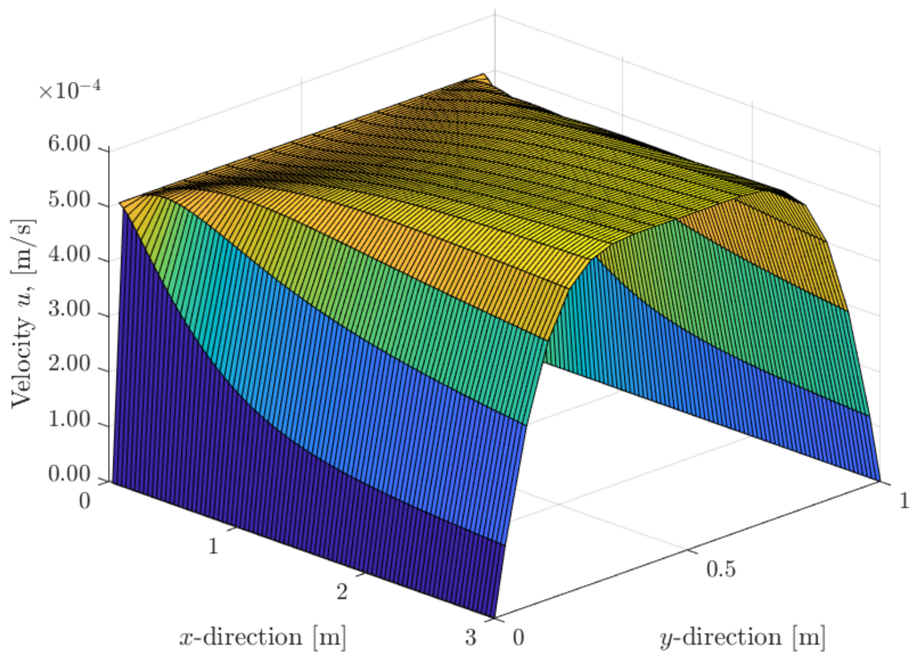


Figure 5.2: u-velocity seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

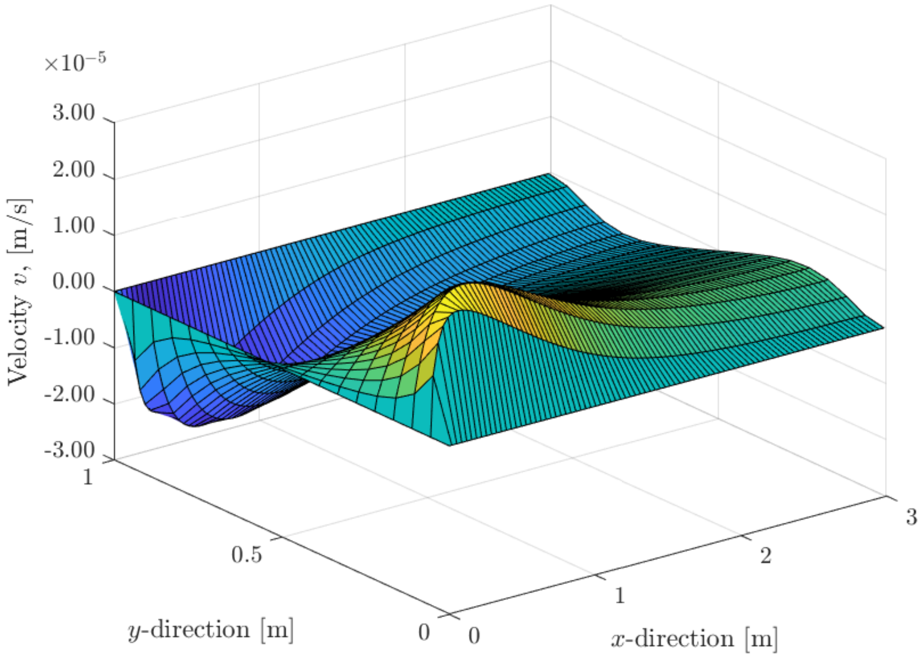


Figure 5.3: v-velocity seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

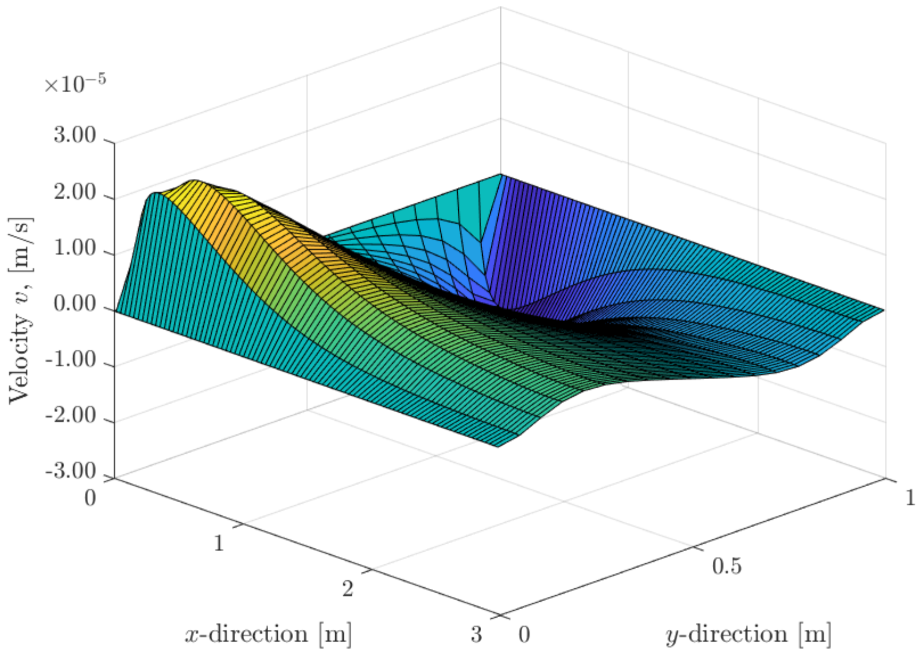


Figure 5.4: v-velocity seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

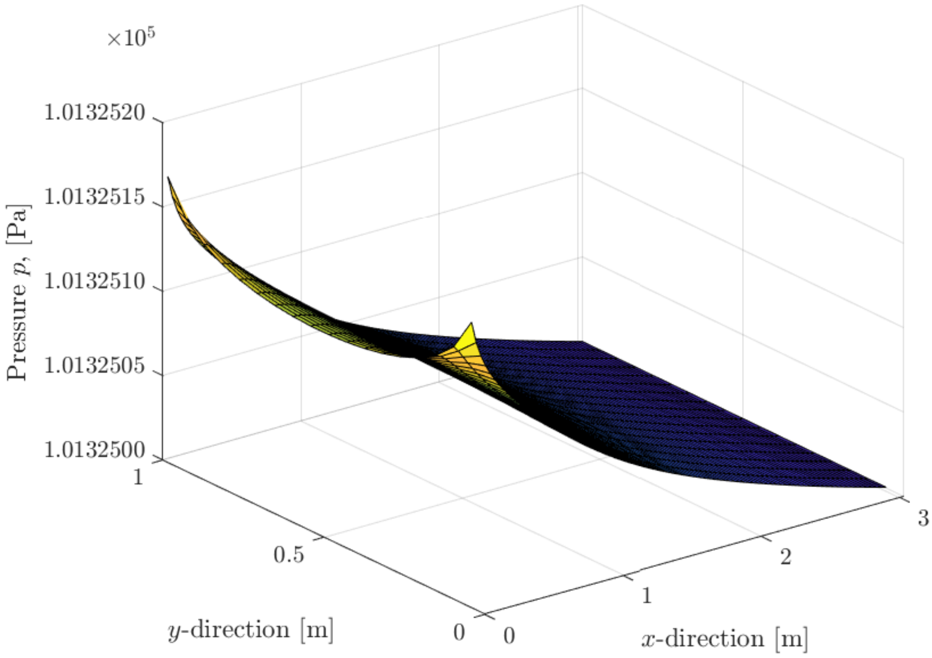


Figure 5.5: Pressure p seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

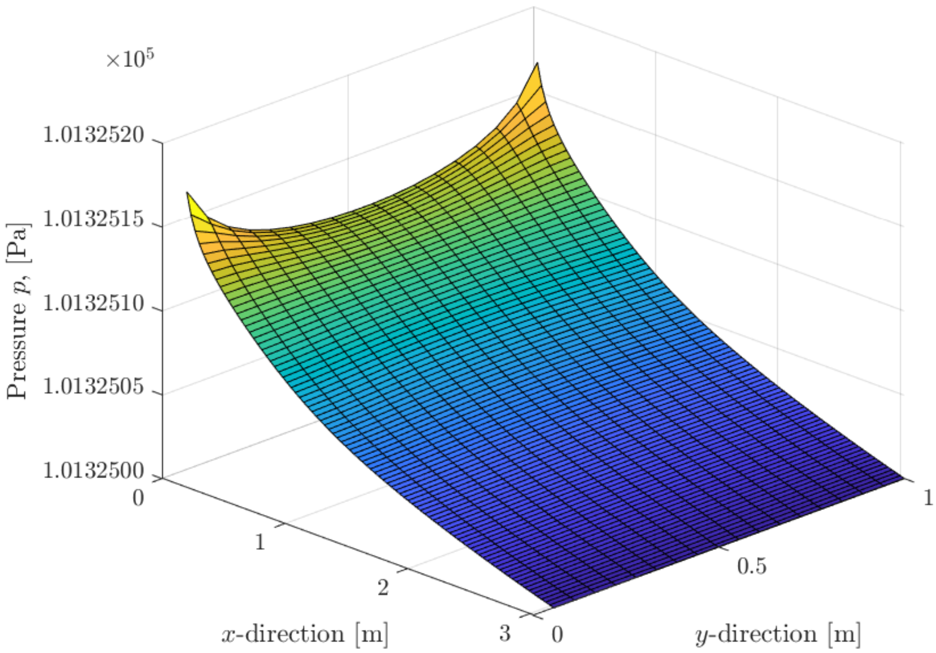


Figure 5.6: Pressure p seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

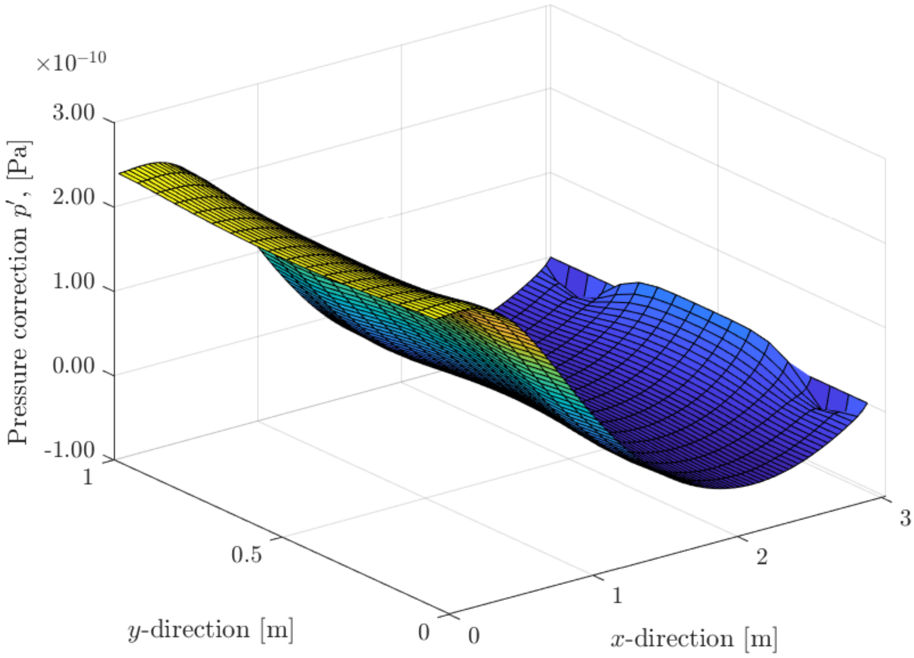


Figure 5.7: Pressure correction p' seen from the inlet for the two dimensional model in a straight channel with $L = 3$.

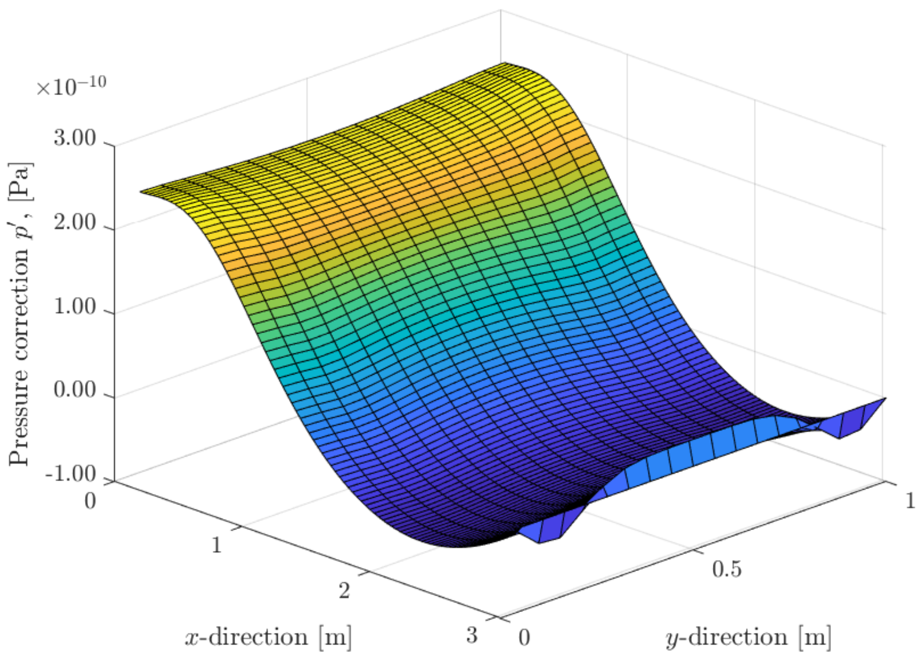


Figure 5.8: Pressure correction p' seen from the outlet for the two dimensional model in a straight channel with $L = 3$.

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, were 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 do 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.

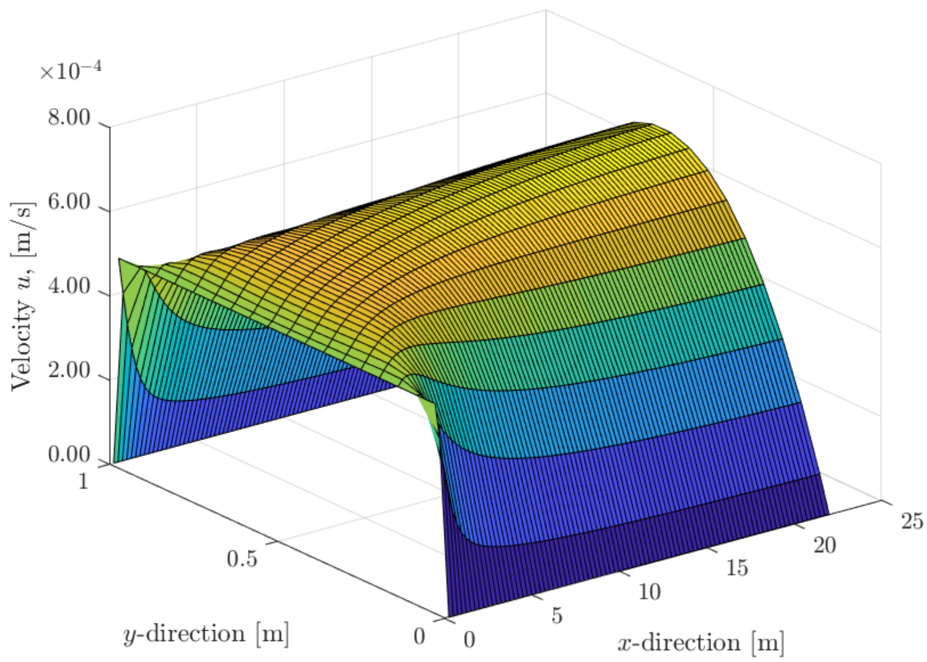


Figure 5.9: u-velocity seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

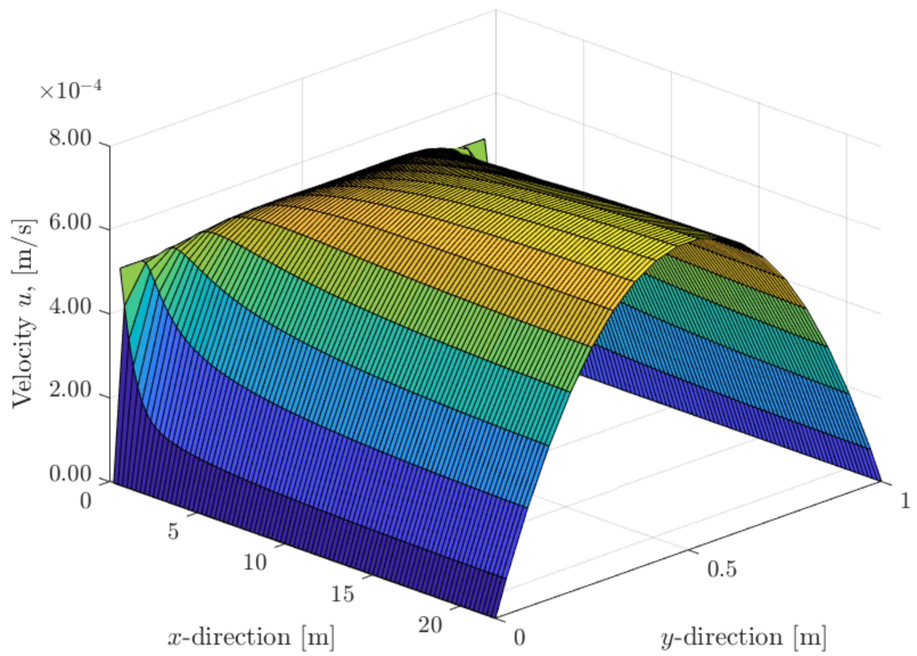


Figure 5.10: u-velocity seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

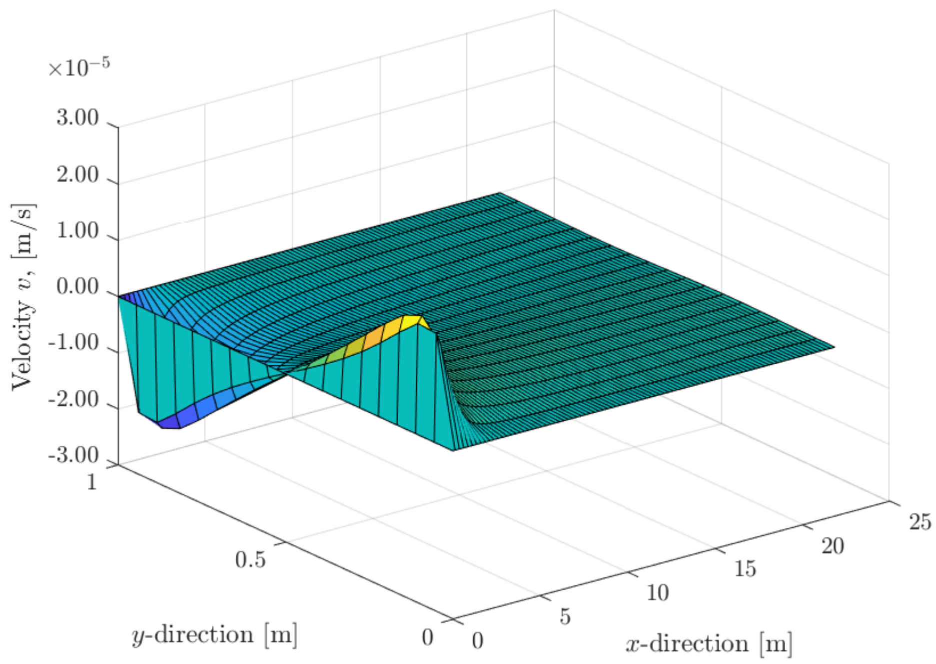


Figure 5.11: v-velocity seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

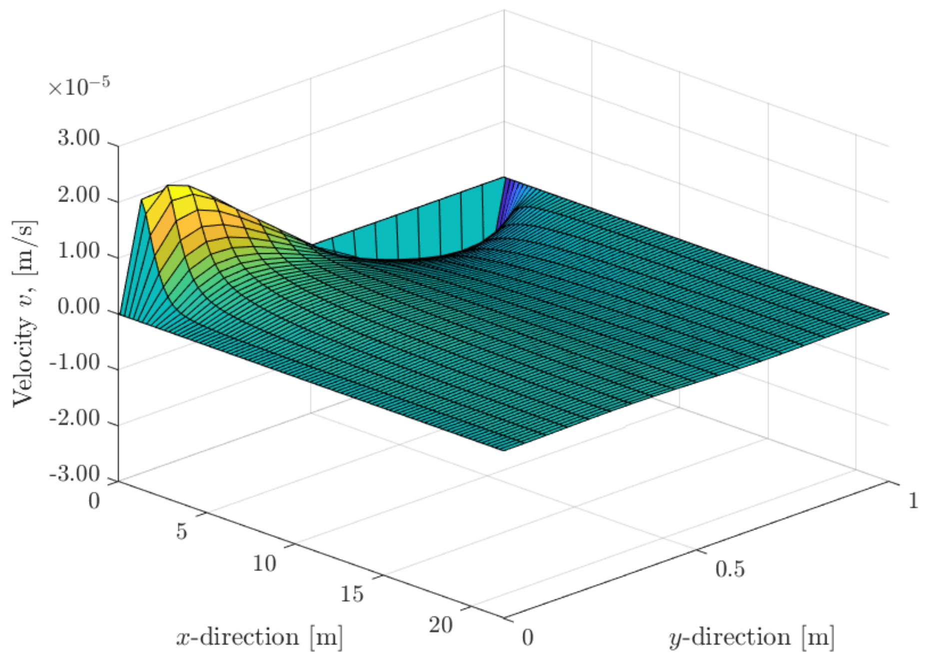


Figure 5.12: v-velocity seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

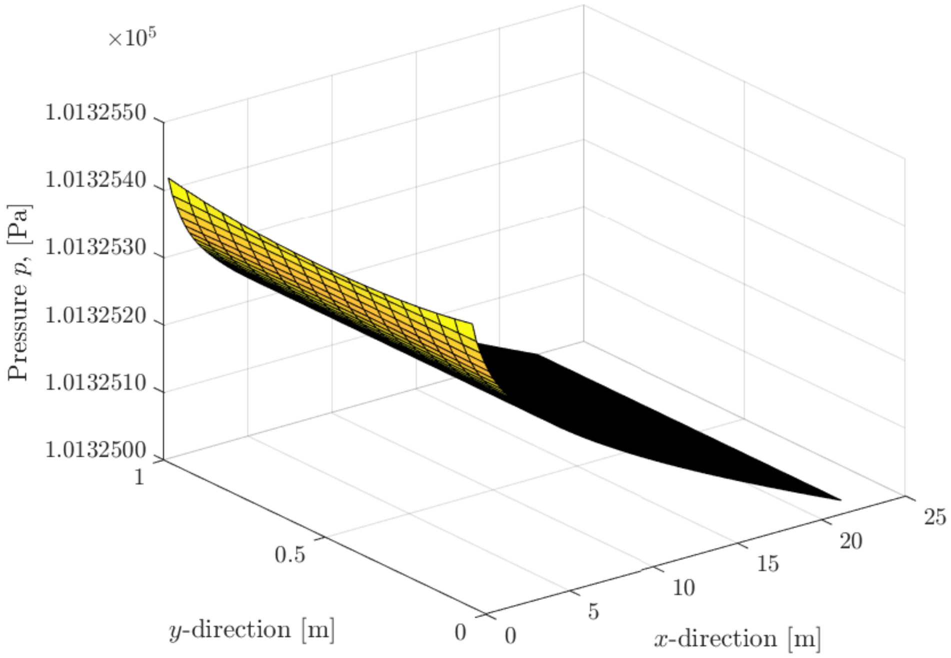


Figure 5.13: Pressure p seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

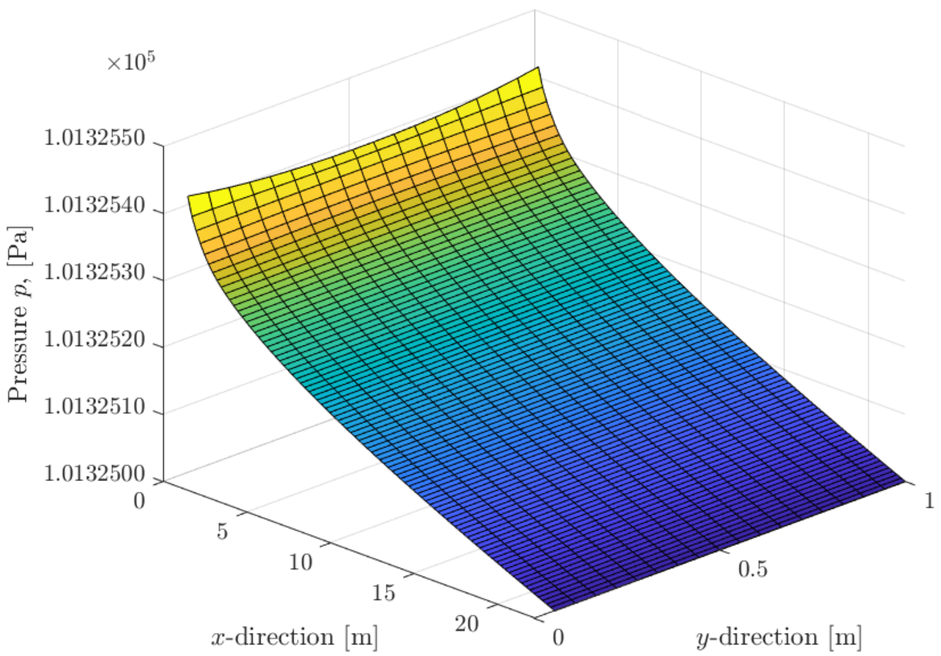


Figure 5.14: Pressure p seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

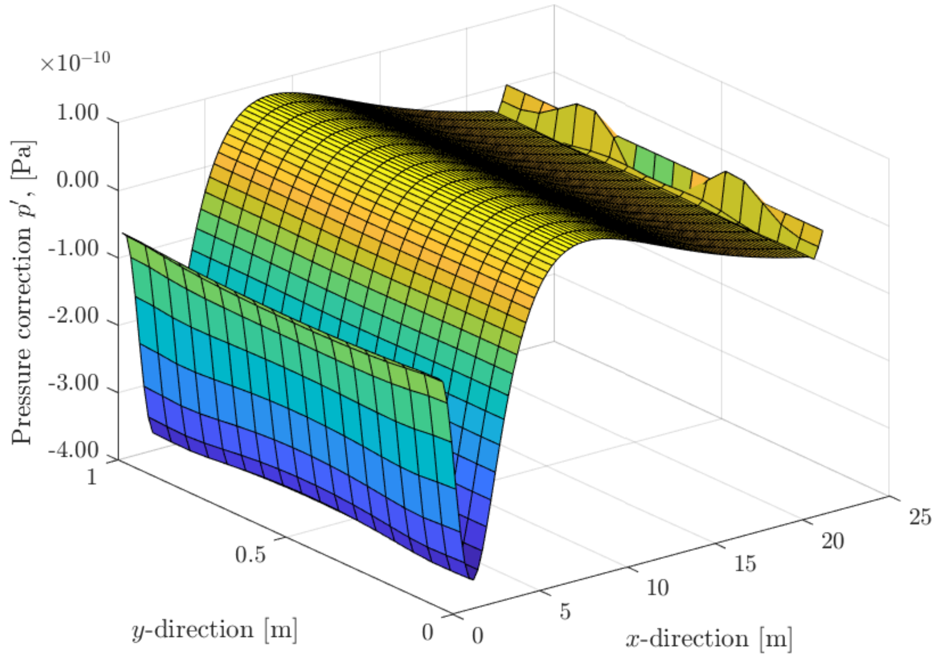


Figure 5.15: Pressure correction p' seen from the inlet for the two dimensional model in a straight channel with $L = 22$.

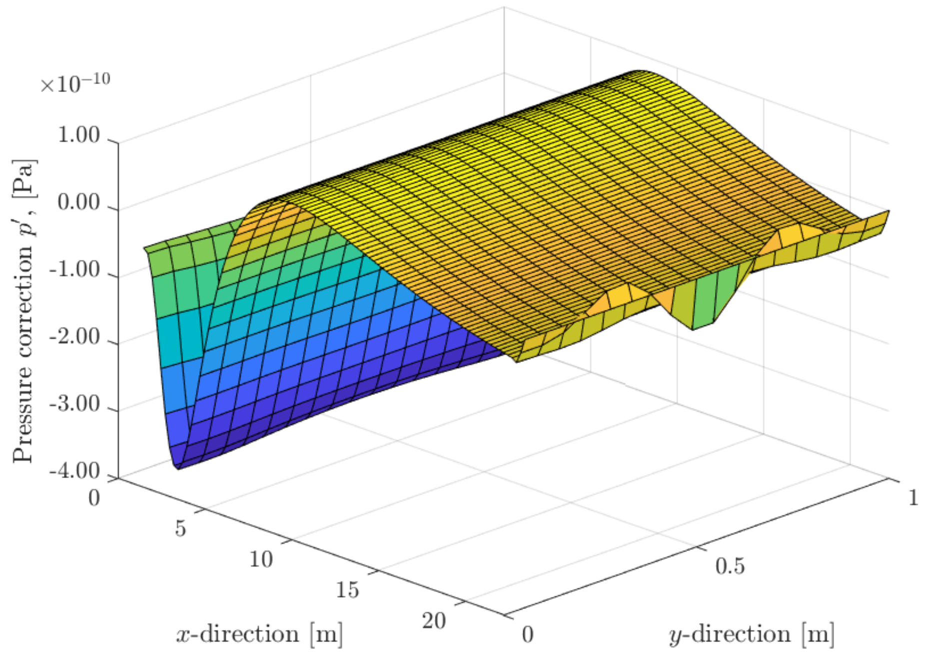


Figure 5.16: Pressure correction p' seen from the outlet for the two dimensional model in a straight channel with $L = 22$.

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.

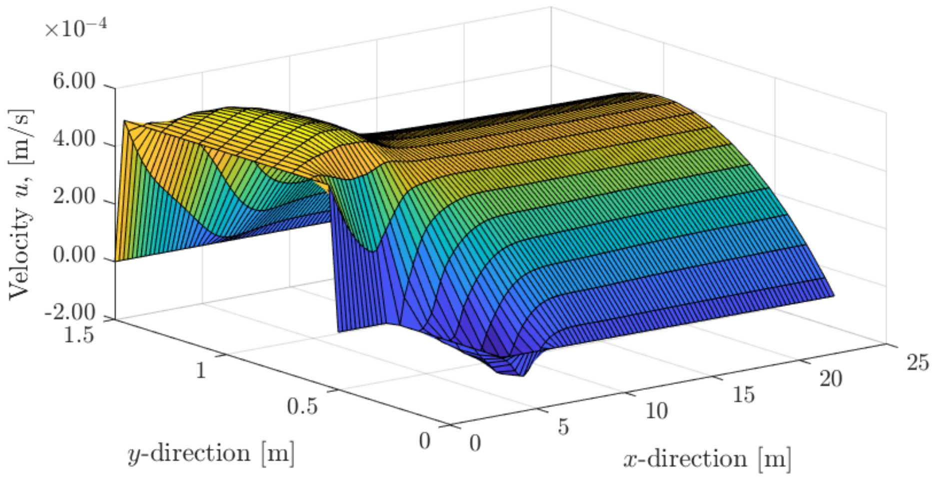


Figure 5.17: u-velocity seen from the inlet for the backwards facing step model.

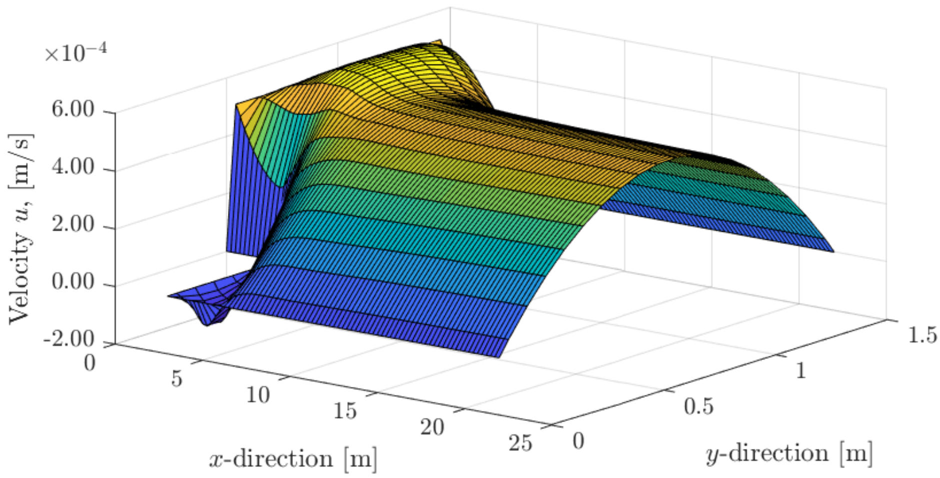


Figure 5.18: u-velocity seen from the outlet for the backwards facing step model.

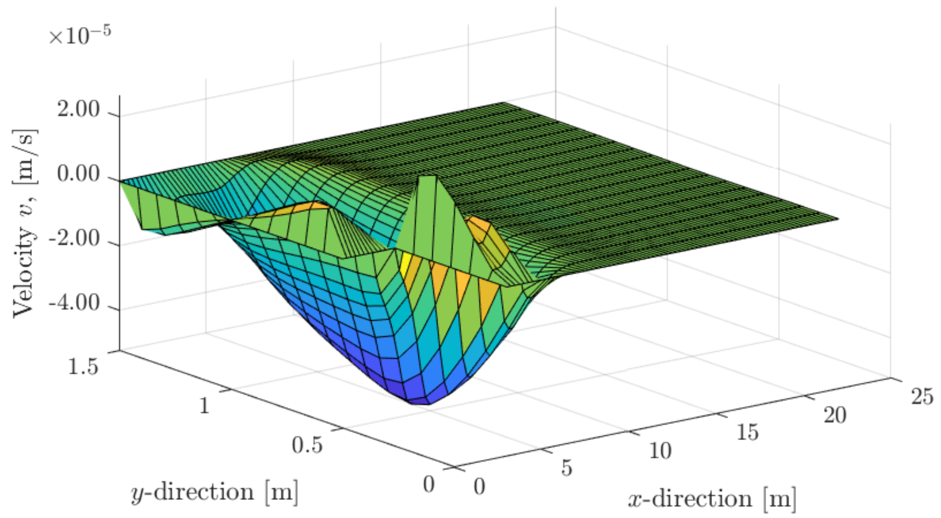


Figure 5.19: v-velocity seen from the inlet for the backwards facing step model.

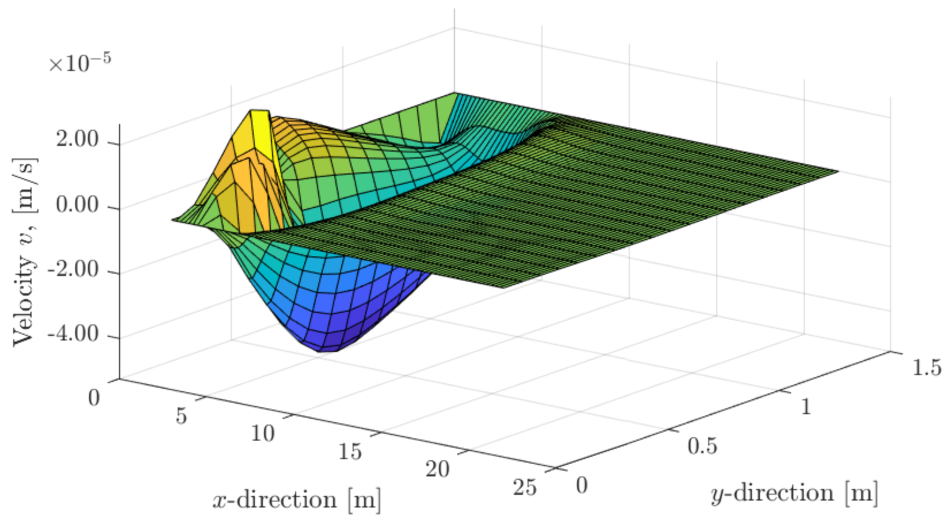


Figure 5.20: v-velocity seen from the outlet for the backwards facing step model.

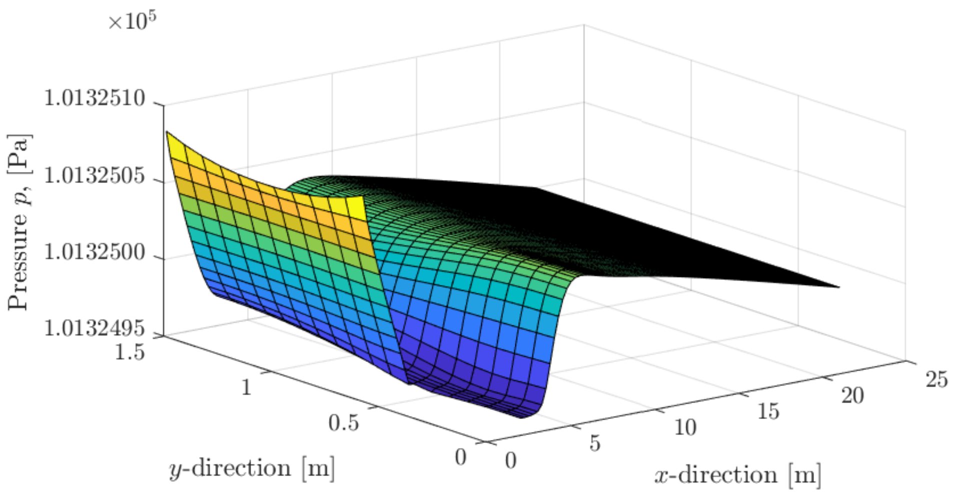


Figure 5.21: Pressure p seen from the inlet for the backwards facing step model.

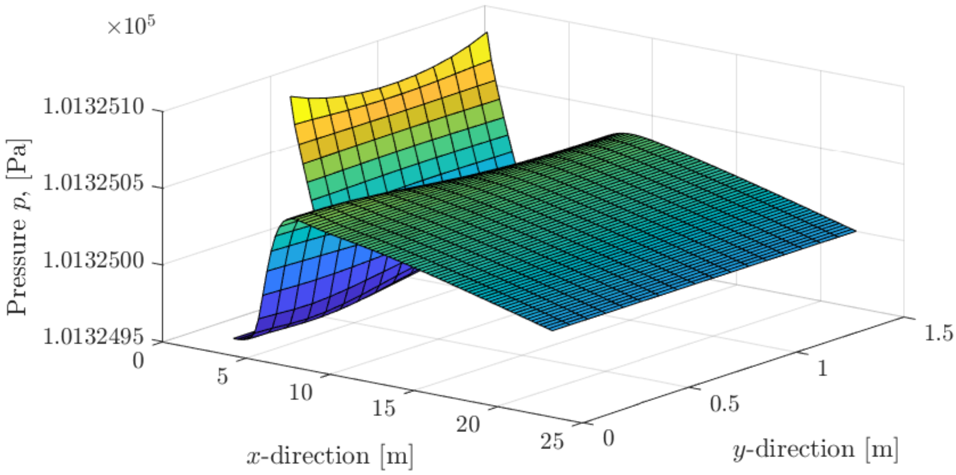


Figure 5.22: Pressure p seen from the outlet for the backwards facing step model.

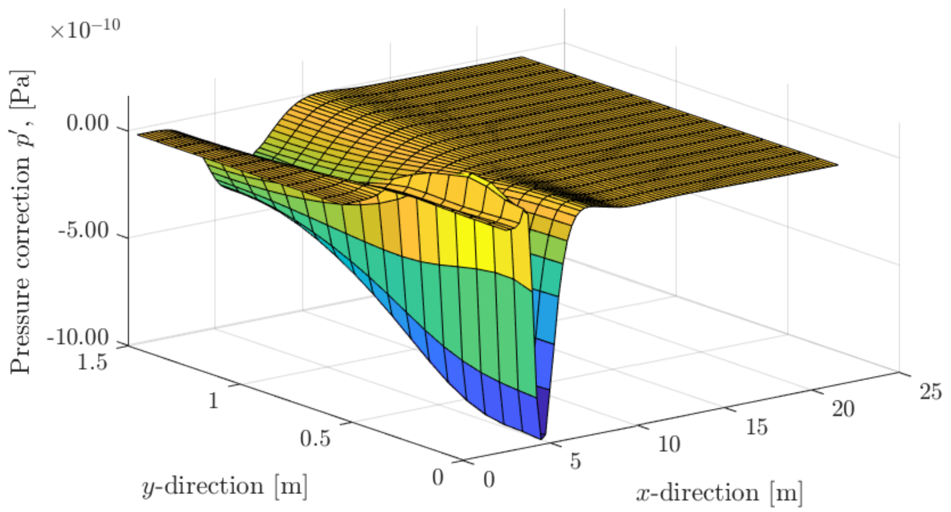


Figure 5.23: Pressure correction p' seen from the inlet for the backwards facing step model.

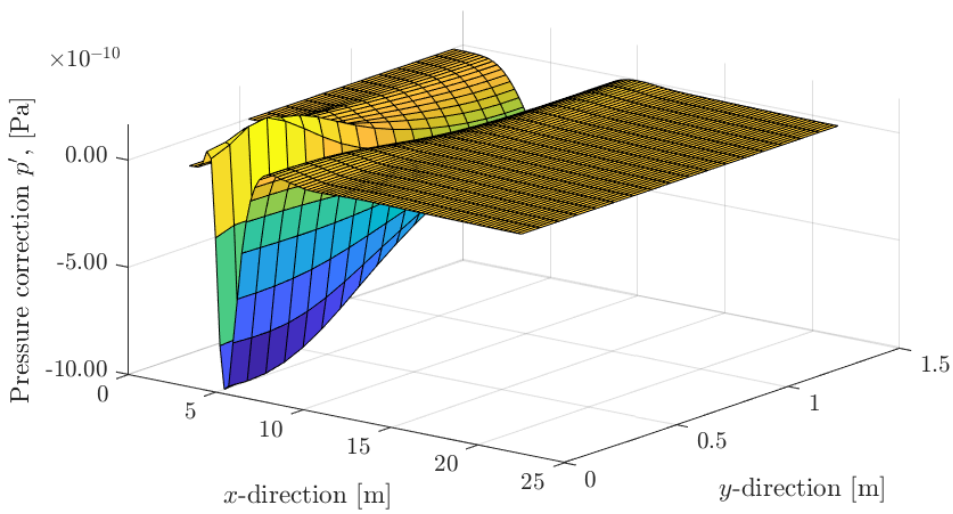


Figure 5.24: Pressure correction p' seen from the outlet for the backwards facing step model.

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.

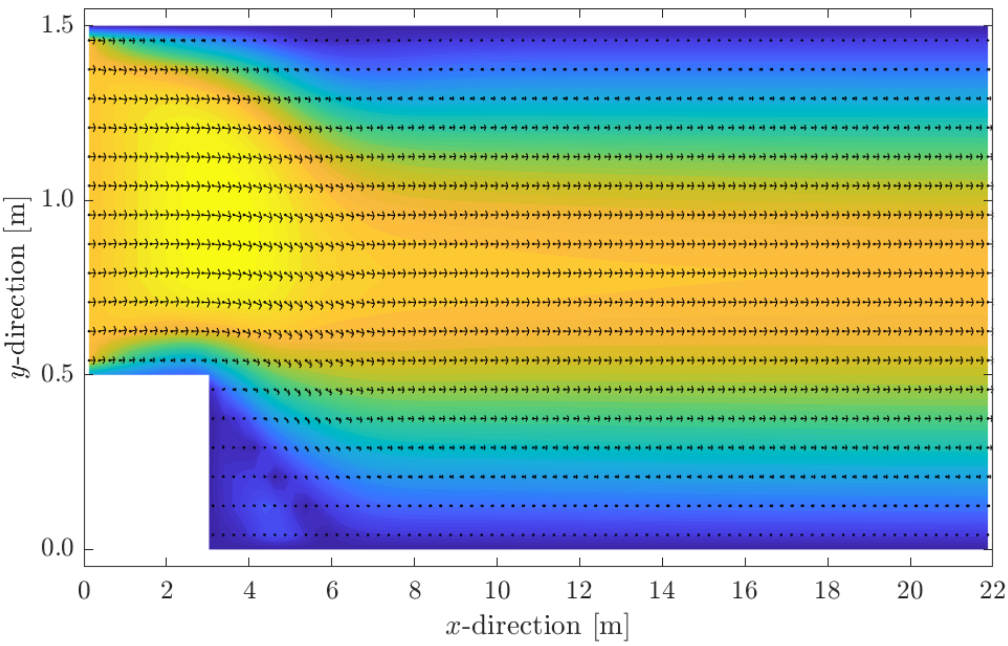


Figure 5.26: Velocity vector plot for the backwards facing step model.

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.

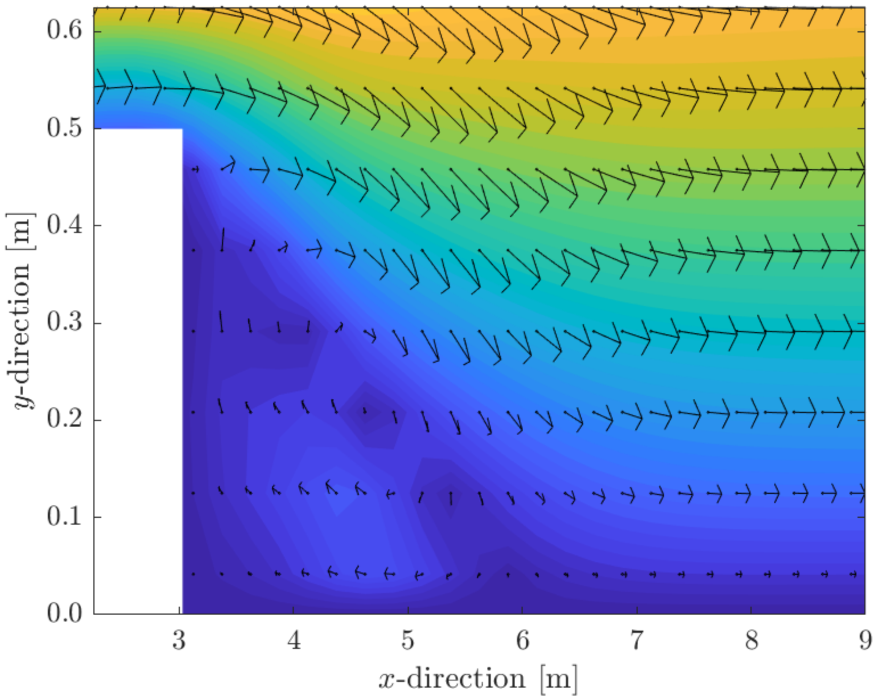


Figure 5.27: Velocity vector plot for the backwards facing step model zoomed in on the recirculation zone after the step.

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.

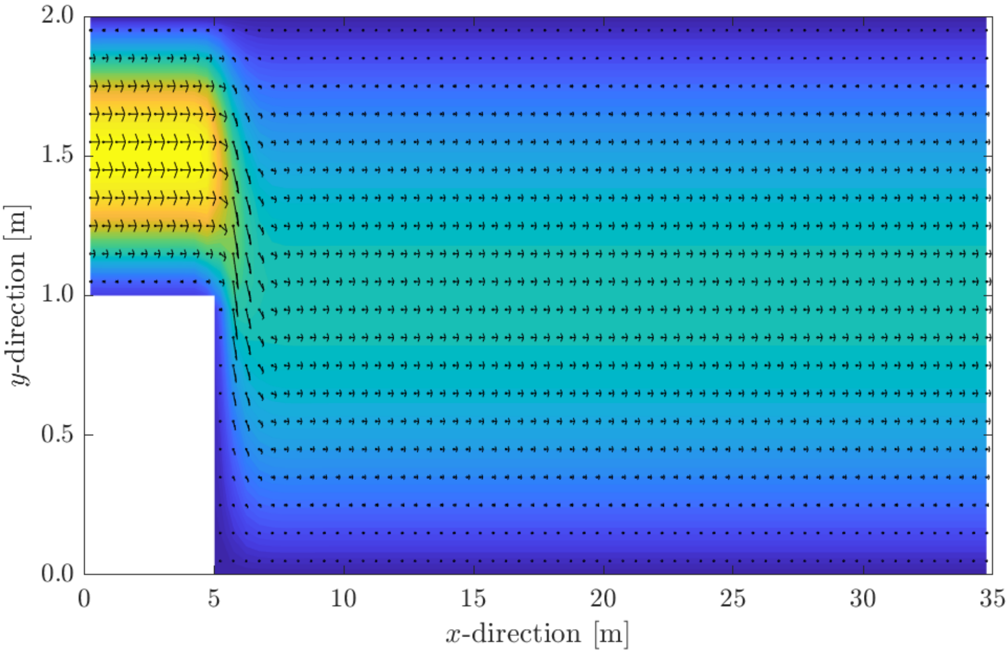


Figure 5.28: Velocity vector plot for the backwards facing step model with $Re = 0.0001$.

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

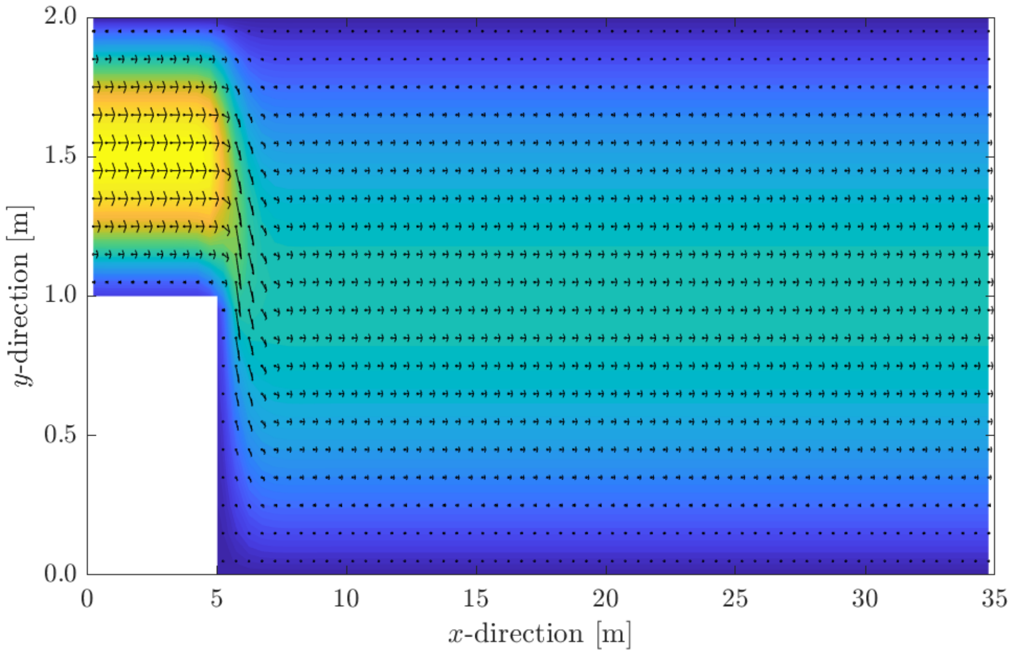


Figure 5.29: Velocity vector plot for the backwards facing step model with $Re = 0.1$.

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.

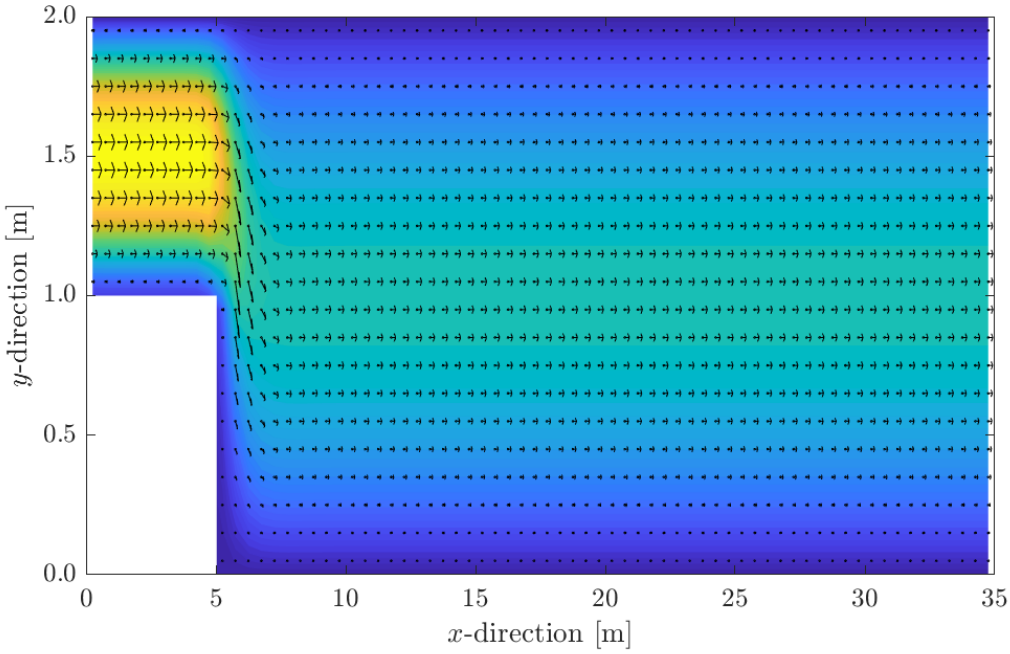


Figure 5.30: Velocity vector plot for the backwards facing step model with $Re = 1$.

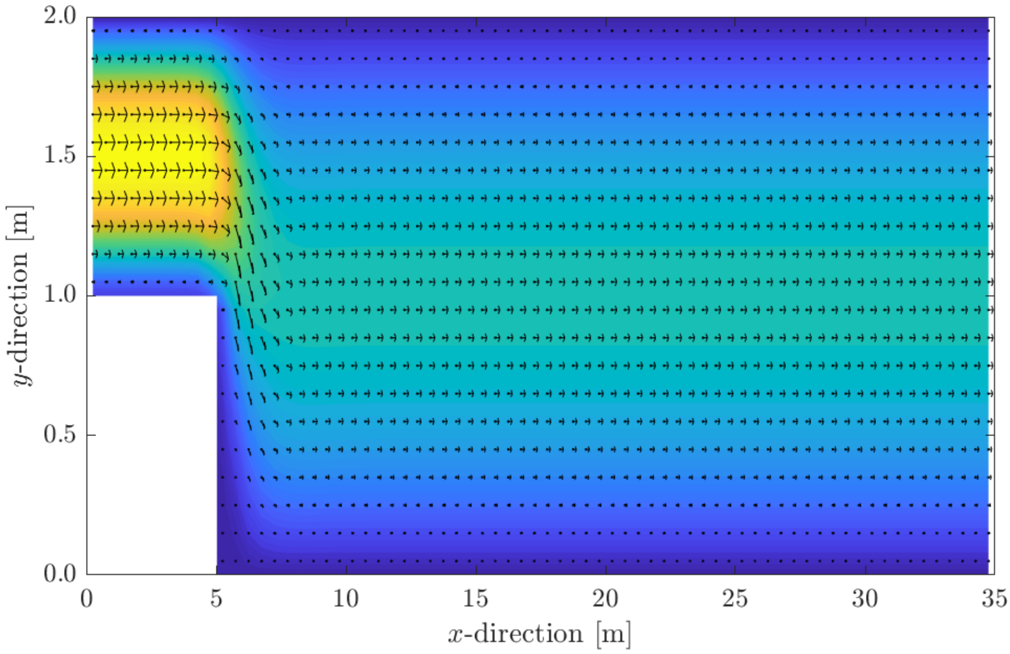


Figure 5.31: Velocity vector plot for the backwards facing step model with $Re = 10$.

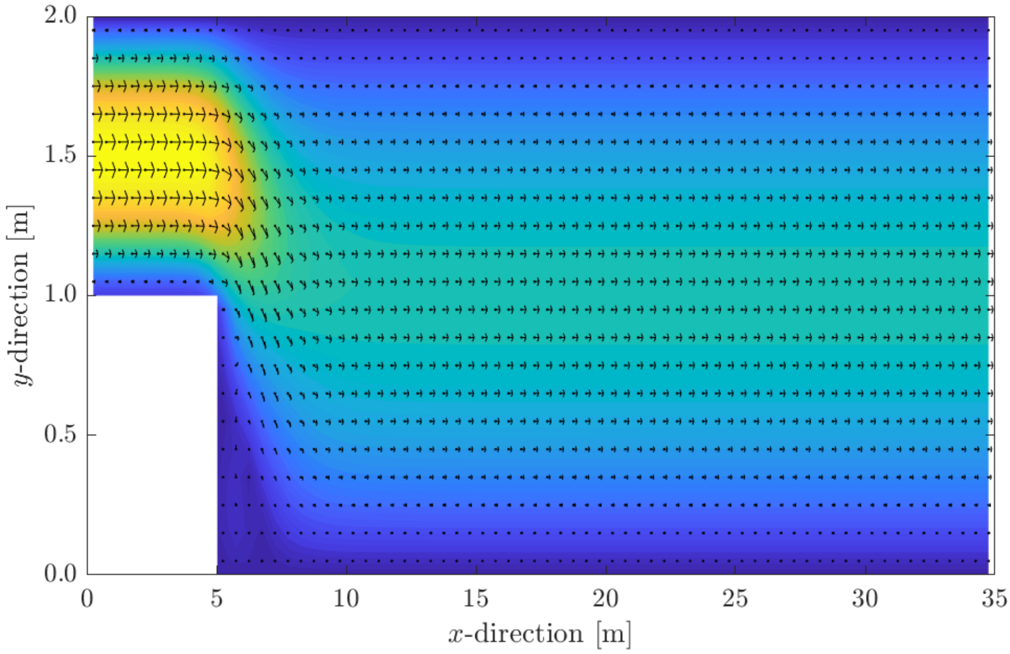


Figure 5.32: Velocity vector plot for the backwards facing step model with $Re = 50$.

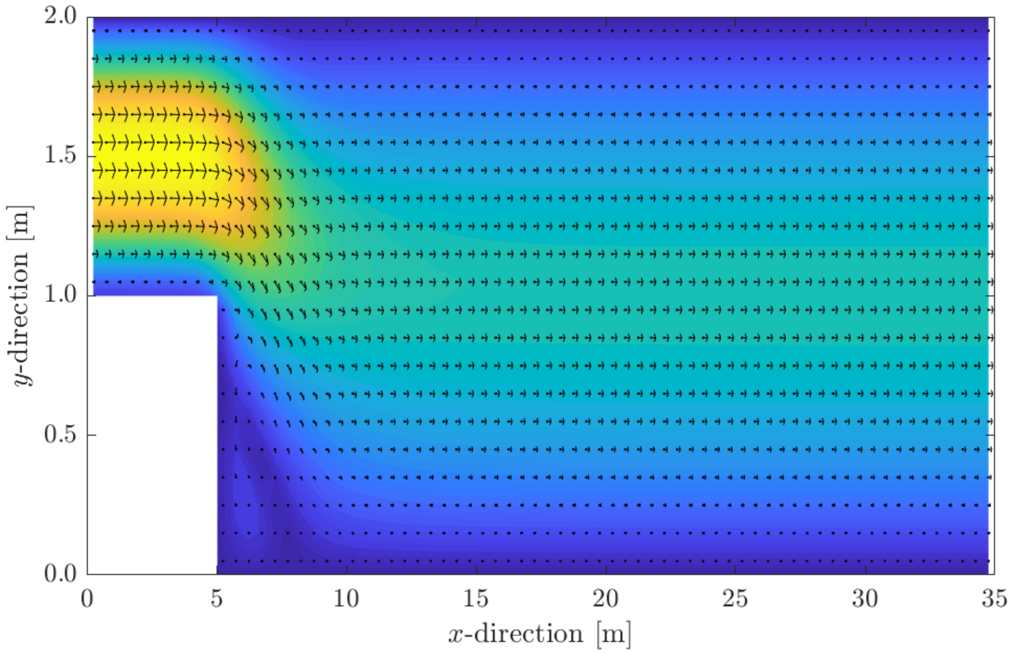


Figure 5.33: Velocity vector plot for the backwards facing step model with $Re = 100$.

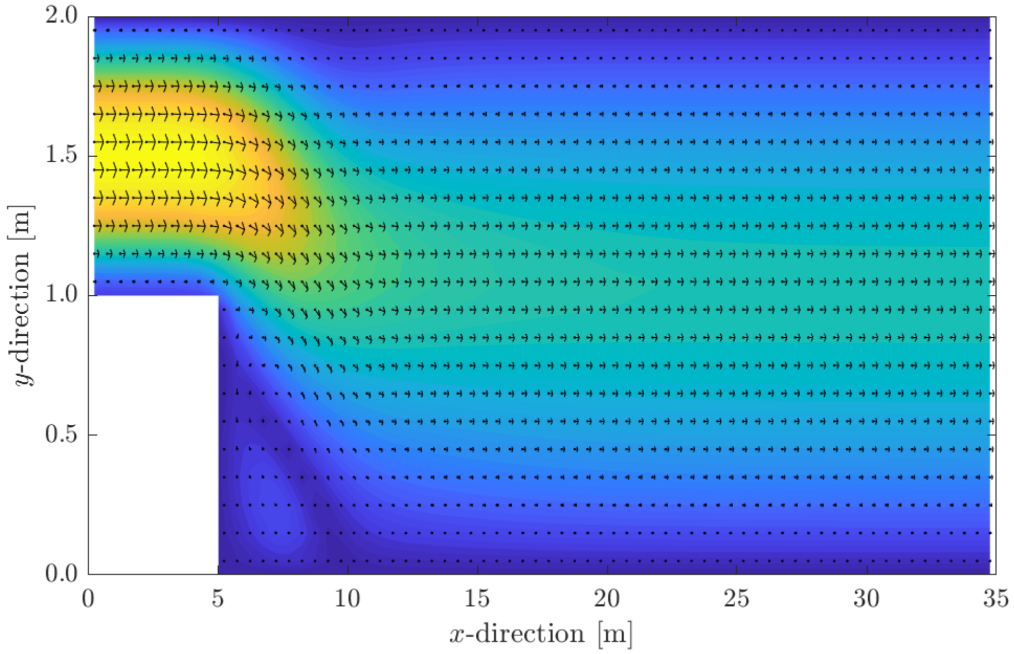


Figure 5.34: Velocity vector plot for the backwards facing step model with $Re = 200$.

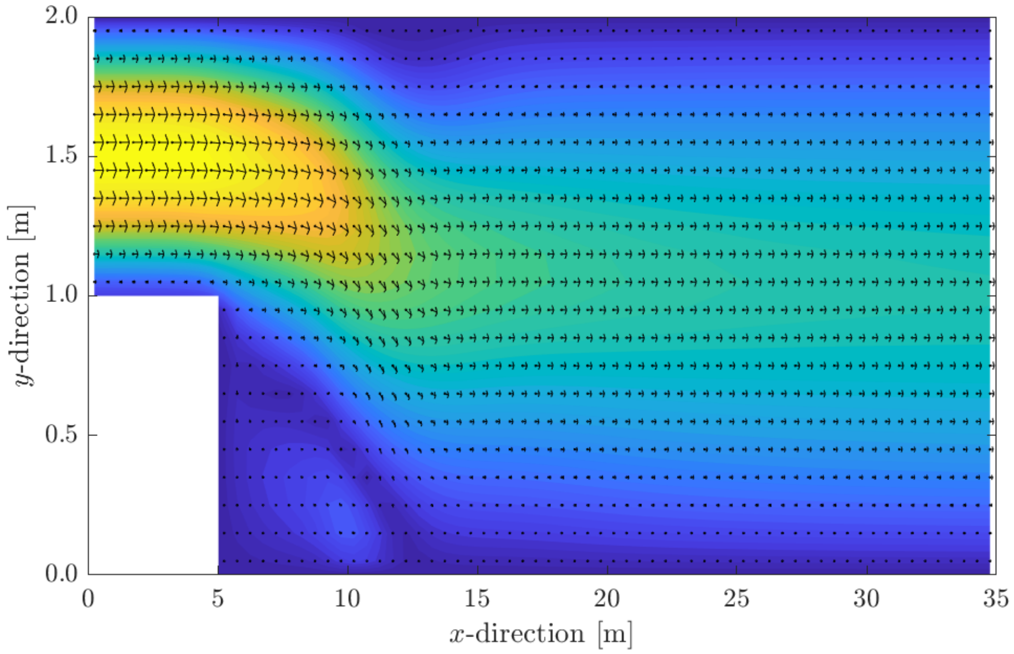


Figure 5.35: Velocity vector plot for the backwards facing step model with $Re = 400$.

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.

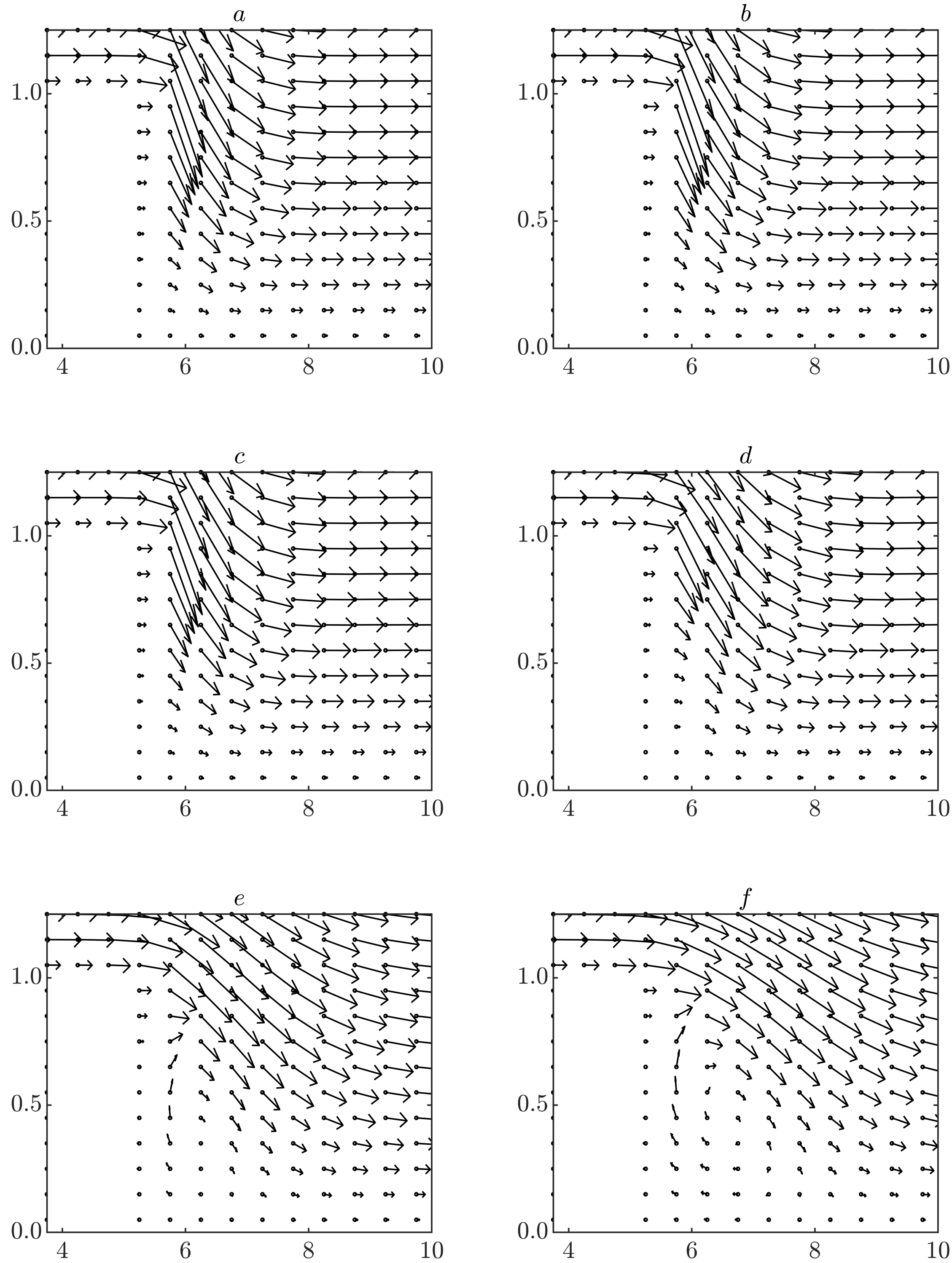


Figure 5.36: Comparison of the recirculation zone over the backwards facing step for different Reynolds numbers. *a*) $Re = 0.0001$, *b*) $Re = 0.1$, *c*) $Re = 1$, *d*) $Re = 10$, *e*) $Re = 50$ and *f*) $Re = 100$.

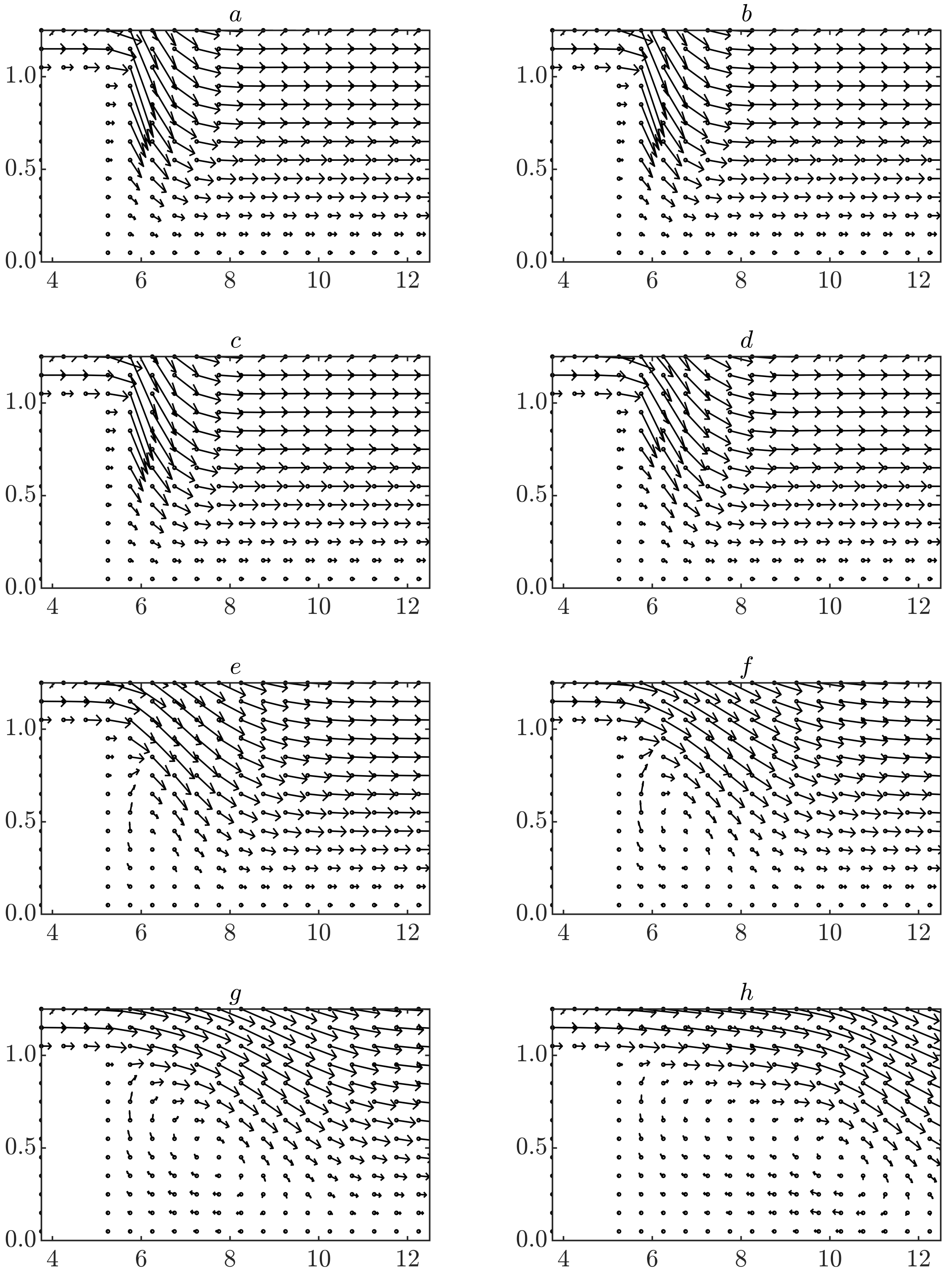


Figure 5.37: Comparison of the recirculation zone over the backwards facing step for different Reynolds numbers. *a*) $Re = 0.0001$, *b*) $Re = 0.1$, *c*) $Re = 1$, *d*) $Re = 10$, *e*) $Re = 50$, *f*) $Re = 100$, *g*) $Re = 200$ and *h*) $Re = 400$.

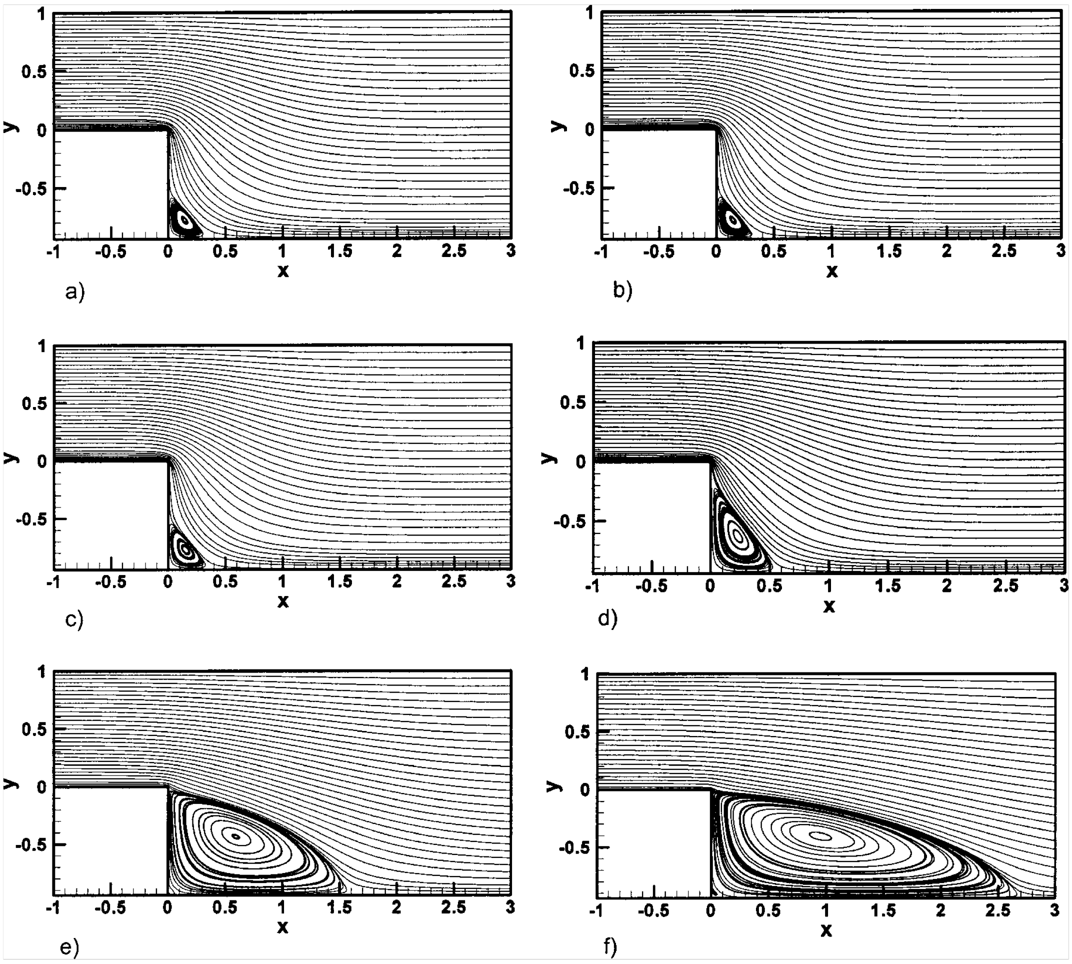


Figure 5.38: Flow over the step as found by Biswas et al. [4]. *a)* $Re = 0.0001$, *b)* $Re = 0.1$, *c)* $Re = 1$, *d)* $Re = 10$, *e)* $Re = 50$, *f)* $Re = 100$.

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.

$$(q^1, q^2) \rightarrow (x, y) \tag{7.1.1}$$

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].

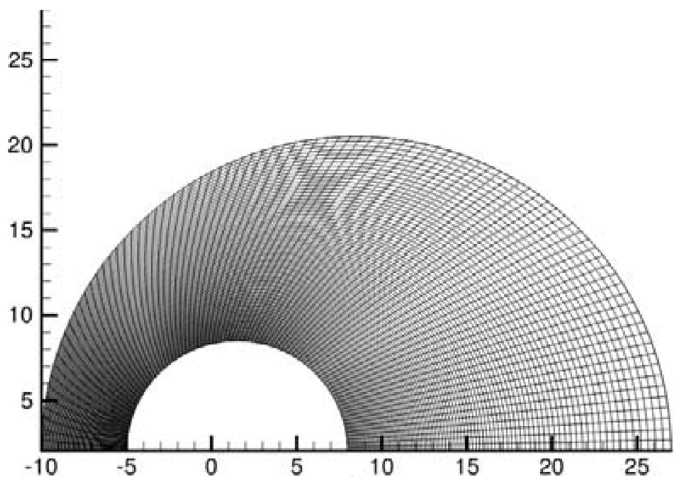


Figure 7.1: Example of a structured grid obtained by use of the elliptic grid generation.

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.

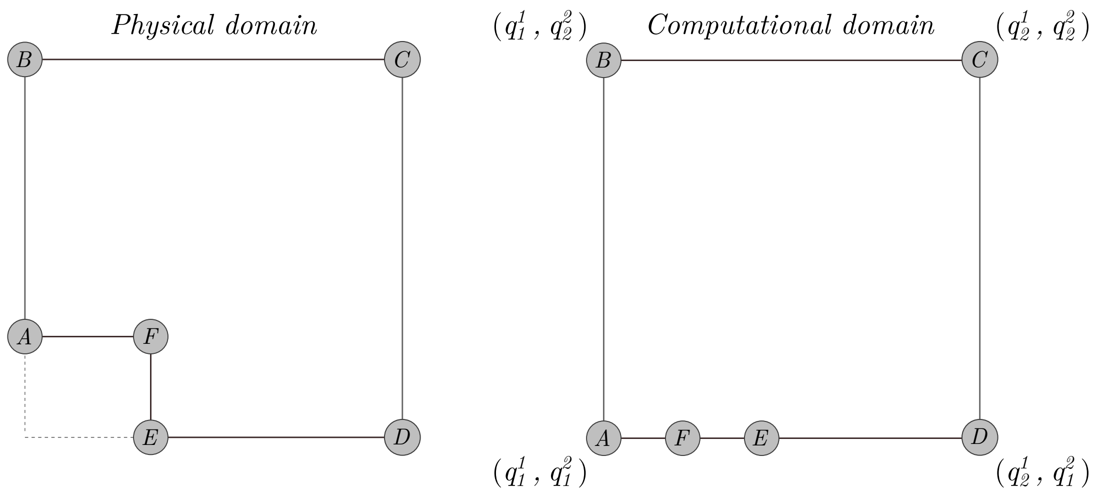


Figure 7.2: Transformation between the physical and the computational domain when using a grid generator.

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.

$$\text{Line segment } AB: \quad q^1 = q^1_1 \tag{7.1.2}$$
$$\text{Line segment } BC: \quad q^2 = q^2_2 \tag{7.1.3}$$
$$\text{Line segment } CD: \quad q^1 = q^1_2 \tag{7.1.4}$$
$$\text{Line segment } DA: \quad q^2 = q^2_1 \tag{7.1.5}$$

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.

$$\mathbf{r}(q^1, q^2) = \sum_{n=1}^{2} \phi_n \left(\frac{q^1}{I}\right) \mathbf{r}\left(q_n^1, q^2\right) \tag{7.1.6}$$

$$\mathbf{r}(q^1, q^2) = \sum_{m=1}^{2} \psi_m \left(\frac{q^2}{J}\right) r\left(q^1, q_m^2\right) \tag{7.1.7}$$

$\mathbf{r}$ 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.

$$\mathbf{r}(q^1, q^2) = \sum_{n=1}^{2} \phi_n \left(\frac{q^1}{I}\right) \mathbf{r}\left(q_n^1, q^2\right) + \sum_{m=1}^{2} \psi_m \left(\frac{q^2}{J}\right) \mathbf{r}\left(q^1, q_m^2\right) - \sum_{n=1}^{2}\sum_{m=1}^{2} \phi_n \left(\frac{q^1}{I}\right) \psi_m \left(\frac{q^2}{J}\right) \mathbf{r}\left(q_n^1, q_m^2\right) \tag{7.1.8}$$

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.

$$g^{ij} \frac{\partial^2 \mathbf{r}}{\partial q^i \partial q^j} + P^j \frac{\partial \mathbf{r}}{\partial q^j} = 0 \tag{7.1.9}$$

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.

$$\frac{\partial}{\partial t}(\rho u_k) + \frac{\partial}{\partial x_i}(\rho u_i u_k) = \frac{\partial}{\partial x_i}\left(\mu \frac{\partial u_k}{\partial x_i}\right) + S_{u_k} \tag{7.2.1}$$

The source term S_{u_k} is defined as in equation (7.2.2).

$$S_{u_k} = \frac{\partial}{\partial x_i}\left(-p\delta_{ik} + \mu \frac{\partial u_i}{\partial x_k} - \frac{2}{3}\mu\delta_{ik}\frac{\partial u_l}{\partial x_l}\right) + B_k \tag{7.2.2}$$

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).

$$\int_{\delta V} -\frac{\partial p}{\partial x^k} dV = -\left(\frac{\partial p}{\partial x^k}\right)_P \delta V_P = -\left(A_k^j \frac{\partial p}{\partial \xi^j}\right)_P \tag{7.2.3}$$

The second term in the source term S_{u_k} in equation (7.2.2) becomes equation (7.2.5)

$$\int_{\delta V} \nabla \cdot \left(\mu \frac{\partial U}{\partial x^k}\right) dV = \int_{\delta A} \mu \frac{\partial U}{\partial x^k} \cdot d\mathbf{A} \tag{7.2.4}$$

$$= \left[\mu \frac{\partial U}{\partial x^k} \cdot \mathbf{A}\right]_w^e + \left[\mu \frac{\partial U}{\partial x^k} \cdot \mathbf{A}\right]_s^n \tag{7.2.5}$$

where the last terms are given in equation (7.2.6).

$$\mu \frac{\partial U}{\partial x^k} \cdot \mathrm{A}\bigg|_{nn} = \mu \frac{A_m^i A_k^j}{J} \frac{\partial u_m}{\partial \xi^j}\bigg|_{nn} = \mu \frac{A_m^i}{J} \frac{\partial u_m}{\partial \xi^j} A_k^j\bigg|_{nn} \tag{7.2.6}$$

This yields the discretised equation in equation (7.2.7).

$$a_P u_{k_P} = \sum_{nb} a_{nb} u_{k_{nb}} + b_{u_n} - \left(A_k^j \frac{\partial p}{\partial \xi^j}\right)_P + a_P^0 u_{k_P}^0 \tag{7.2.7}$$

with

$$b_{u_k} = b_{NO} + \bar{S}_{1P} + \int_{\delta V} \nabla \cdot \left(\mu \frac{\partial U}{\partial x^k}\right) dV \tag{7.2.8}$$

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.

$$\mathbf{r}(q^1, q^2) = \phi_1\left(\frac{q^1}{q_2^1}\right)\mathbf{r}\left(q_1^1, q^2\right) + \phi_2\left(\frac{q^1}{q_2^1}\right)\mathbf{r}\left(q_2^1, q^2\right) \tag{7.3.1}$$

$$\mathbf{r}(q^1, q^2) = \psi_1 \left(\frac{q^2}{q_2^2}\right) \mathbf{r}\left(q^1, q_1^2\right) + \psi_2 \left(\frac{q^2}{q_2^2}\right) \mathbf{r}\left(q^1, q_2^2\right) \tag{7.3.2}$$

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.

$$\mathbf{r} = x\mathbf{e}_x + y\mathbf{e}_y \tag{7.3.3}$$

Equation (7.3.1) is used for the line segments (B)(C) and (A)(D) in figure 7.2, and equation (7.3.2) is used for the line segments (A)(B) and (D)(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 (A)(D) goes from (A) to (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 (7.3.1) depending on the line segment as specified in the above paragraph.

Equation (7.3.4) applies to line segment (A)(B), where q^1 in equation (7.3.2) has been replaced with q_1^1.

$$\mathbf{r}\left(q_1^1, q^2\right) = \psi_1 \left(\frac{q^2}{q_2^2}\right) \mathbf{r}\left(q_1^1, q_1^2\right) + \psi_2 \left(\frac{q^2}{q_2^2}\right) \mathbf{r}\left(q_1^1, q_2^2\right) \tag{7.3.4}$$

Equation (7.3.5) applies to line segment (B)(C), where q^2 in equation (7.3.1) has been replaced with q_2^2.

$$\mathbf{r}\left(q^1, q_2^2\right) = \phi_1 \left(\frac{q^1}{q_2^1}\right) \mathbf{r}\left(q_1^1, q_2^2\right) + \phi_2 \left(\frac{q^1}{q_2^1}\right) \mathbf{r}\left(q_2^1, q_2^2\right) \tag{7.3.5}$$

Equation (7.3.6) applies to line segment (D)(C), where q^1 in equation (7.3.2) has been replaced with q_2^1.

$$\mathbf{r}\left(q_2^1, q^2\right) = \psi_1 \left(\frac{q^2}{q_2^2}\right) \mathbf{r}\left(q_2^1, q_1^2\right) + \psi_2 \left(\frac{q^2}{q_2^2}\right) \mathbf{r}\left(q_2^1, q_2^2\right) \tag{7.3.6}$$

Equation (7.3.7) applies to line segment (A)(D), where q^2 in equation (7.3.1) has been replaced with q_1^2.

$$\mathbf{r}\left(q^1, q_1^2\right) = \phi_1 \left(\frac{q^1}{q_2^1}\right) \mathbf{r}\left(q_1^1, q_1^2\right) + \phi_2 \left(\frac{q^1}{q_2^1}\right) \mathbf{r}\left(q_2^1, q_1^2\right) \tag{7.3.7}$$

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).

$$\text{(A)(B)}: \; x\left(q_1^1, q^2\right) = \psi_1\left(\frac{q^2}{q_2^2}\right) x\left(q_1^1, q_1^2\right) + \psi_2\left(\frac{q^2}{q_2^2}\right) x\left(q_1^1, q_2^2\right) \tag{7.3.8}$$

$$y\left(q_1^1, q^2\right) = \psi_1\left(\frac{q^2}{q_2^2}\right) y\left(q_1^1, q_1^2\right) + \psi_2\left(\frac{q^2}{q_2^2}\right) y\left(q_1^1, q_2^2\right) \tag{7.3.9}$$

$$\text{(B)(C)}: \; x\left(q^1, q_2^2\right) = \phi_1\left(\frac{q^1}{q_2^1}\right) x\left(q_1^1, q_2^2\right) + \phi_2\left(\frac{q^1}{q_2^1}\right) x\left(q_2^1, q_2^2\right) \tag{7.3.10}$$

$$y\left(q^1, q_2^2\right) = \phi_1\left(\frac{q^1}{q_2^1}\right) y\left(q_1^1, q_2^2\right) + \phi_2\left(\frac{q^1}{q_2^1}\right) y\left(q_2^1, q_2^2\right) \tag{7.3.11}$$

$$\text{(D)(C)}: \; x\left(q_2^1, q^2\right) = \psi_1\left(\frac{q^2}{q_2^2}\right) x\left(q_2^1, q_1^2\right) + \psi_2\left(\frac{q^2}{q_2^2}\right) x\left(q_2^1, q_2^2\right) \tag{7.3.12}$$

$$y\left(q_2^1, q^2\right) = \psi_1\left(\frac{q^2}{q_2^2}\right) y\left(q_2^1, q_1^2\right) + \psi_2\left(\frac{q^2}{q_2^2}\right) y\left(q_2^1, q_2^2\right) \tag{7.3.13}$$

$$\text{(A)(D)}: \; x\left(q^1, q_1^2\right) = \phi_1\left(\frac{q^1}{q_2^1}\right) x\left(q_1^1, q_1^2\right) + \phi_2\left(\frac{q^1}{q_2^1}\right) x\left(q_2^1, q_1^2\right) \tag{7.3.14}$$

$$y\left(q^1, q_1^2\right) = \phi_1\left(\frac{q^1}{q_2^1}\right) y\left(q_1^1, q_1^2\right) + \phi_2\left(\frac{q^1}{q_2^1}\right) y\left(q_2^1, q_1^2\right) \tag{7.3.15}$$

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].

$$\phi_n\left(\frac{q^i}{q^i_{max}}\right) = \prod_{k=1}^{N} \frac{q^i - q^i_k}{q^i_n - q^i_k} \quad (k \neq n) \tag{7.3.16}$$

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.

$$\phi_1\left(\frac{q^1}{q_2^1}\right) = 1 - \frac{q^1}{q_2^1} \tag{7.3.17}$$

$$\phi_2\left(\frac{q^1}{q_2^1}\right) = \frac{q^1}{q_2^1} \tag{7.3.18}$$

$$\psi_1\left(\frac{q^2}{q_2^2}\right) = 1 - \frac{q^2}{q_2^2} \tag{7.3.19}$$

$$\psi_2\left(\frac{q^2}{q_2^2}\right) = \frac{q^2}{q_2^2} \tag{7.3.20}$$

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).

$$\textcircled{A}\textcircled{B}:\ x\left(q_1^1, q^2\right) = \left(1 - \frac{q^2}{q_2^2}\right) x\left(q_1^1, q_1^2\right) + \frac{q^2}{q_2^2} x\left(q_1^1, q_2^2\right) \tag{7.3.21}$$

$$y\left(q_1^1, q^2\right) = \left(1 - \frac{q^2}{q_2^2}\right) y\left(q_1^1, q_1^2\right) + \frac{q^2}{q_2^2} y\left(q_1^1, q_2^2\right) \tag{7.3.22}$$

$$\textcircled{B}\textcircled{C}:\ x\left(q^1, q_2^2\right) = \left(1 - \frac{q^1}{q_2^1}\right) x\left(q_1^1, q_2^2\right) + \frac{q^1}{q_2^1} x\left(q_2^1, q_2^2\right) \tag{7.3.23}$$

$$y\left(q^1, q_2^2\right) = \left(1 - \frac{q^1}{q_2^1}\right) y\left(q_1^1, q_2^2\right) + \frac{q^1}{q_2^1} y\left(q_2^1, q_2^2\right) \tag{7.3.24}$$

$$\textcircled{D}\textcircled{C}:\ x\left(q_2^1, q^2\right) = \left(1 - \frac{q^2}{q_2^2}\right) x\left(q_2^1, q_1^2\right) + \frac{q^2}{q_2^2} x\left(q_2^1, q_2^2\right) \tag{7.3.25}$$

$$y\left(q_2^1, q^2\right) = \left(1 - \frac{q^2}{q_2^2}\right) y\left(q_2^1, q_1^2\right) + \frac{q^2}{q_2^2} y\left(q_2^1, q_2^2\right) \tag{7.3.26}$$

$$\textcircled{A}\textcircled{D}:\ x\left(q^1, q_1^2\right) = \left(1 - \frac{q^1}{q_2^1}\right) x\left(q_1^1, q_1^2\right) + \frac{q^1}{q_2^1} x\left(q_2^1, q_1^2\right) \tag{7.3.27}$$

$$y\left(q^1, q_1^2\right) = \left(1 - \frac{q^1}{q_2^1}\right) y\left(q_1^1, q_1^2\right) + \frac{q^1}{q_2^1} y\left(q_2^1, q_1^2\right) \tag{7.3.28}$$

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).

$$\textcircled{A}\textcircled{B}:\ x_{AB} = \left(1 - \frac{q^2}{q_2^2}\right) x_A + \frac{q^2}{q_2^2} x_B \tag{7.3.29}$$

$$y_{AB} = \left(1 - \frac{q^2}{q_2^2}\right) y_A + \frac{q^2}{q_2^2} y_B \tag{7.3.30}$$

$$\textcircled{B}\textcircled{C}:\ x_{BC} = \left(1 - \frac{q^1}{q_2^1}\right) x_B + \frac{q^1}{q_2^1} x_C \tag{7.3.31}$$

$$y_{BC} = \left(1 - \frac{q^1}{q_2^1}\right) y_B + \frac{q^1}{q_2^1} y_C \tag{7.3.32}$$

$$\textcircled{D}\textcircled{C}:\ x_{DC} = \left(1 - \frac{q^2}{q_2^2}\right) x_D + \frac{q^2}{q_2^2} x_C \tag{7.3.33}$$

$$y_{DC} = \left(1 - \frac{q^2}{q_2^2}\right) y_D + \frac{q^2}{q_2^2} y_C \tag{7.3.34}$$

$$\textcircled{A}\textcircled{D}:\ x_{AD} = \left(1 - \frac{q^1}{q_2^1}\right) x_A + \frac{q^1}{q_2^1} x_D \tag{7.3.35}$$

$$y_{AD} = \left(1 - \frac{q^1}{q_2^1}\right) y_A + \frac{q^1}{q_2^1} y_D \tag{7.3.36}$$

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.

$$
\begin{aligned}
\mathbf{r}(q^1,q^2) = {}& \phi_1\left(\frac{q^1}{q^1_2}\right)\mathbf{r}\left(q^1_1,q^2\right)+\phi_2\left(\frac{q^1}{q^1_2}\right)\mathbf{r}\left(q^1_2,q^2\right)+\psi_1\left(\frac{q^2}{q^2_2}\right)\mathbf{r}\left(q^1,q^2_1\right)+\psi_2\left(\frac{q^2}{q^2_2}\right)\mathbf{r}\left(q^1,q^2_2\right)\\
&+\phi_1\left(\frac{q^1}{q^1_2}\right)\psi_1\left(\frac{q^2}{q^2_2}\right)\mathbf{r}\left(q^1_1,q^2_1\right)+\phi_1\left(\frac{q^1}{q^1_2}\right)\psi_2\left(\frac{q^2}{q^2_2}\right)\mathbf{r}\left(q^1_1,q^2_2\right)\\
&+\phi_2\left(\frac{q^1}{q^1_2}\right)\psi_1\left(\frac{q^2}{q^2_2}\right)\mathbf{r}\left(q^1_2,q^2_1\right)+\phi_2\left(\frac{q^1}{q^1_2}\right)\psi_2\left(\frac{q^2}{q^2_2}\right)\mathbf{r}\left(q^1_2,q^2_2\right)
\end{aligned}
\tag{7.3.37}
$$

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).

$$
\begin{aligned}
x(q^1,q^2) = {}& \phi_1\left(\frac{q^1}{q^1_2}\right)x\left(q^1_1,q^2\right)+\phi_2\left(\frac{q^1}{q^1_2}\right)x\left(q^1_2,q^2\right)+\psi_1\left(\frac{q^2}{q^2_2}\right)x\left(q^1,q^2_1\right)+\psi_2\left(\frac{q^2}{q^2_2}\right)x\left(q^1,q^2_2\right)\\
&+\phi_1\left(\frac{q^1}{q^1_2}\right)\psi_1\left(\frac{q^2}{q^2_2}\right)x\left(q^1_1,q^2_1\right)+\phi_1\left(\frac{q^1}{q^1_2}\right)\psi_2\left(\frac{q^2}{q^2_2}\right)x\left(q^1_1,q^2_2\right)\\
&+\phi_2\left(\frac{q^1}{q^1_2}\right)\psi_1\left(\frac{q^2}{q^2_2}\right)x\left(q^1_2,q^2_1\right)+\phi_2\left(\frac{q^1}{q^1_2}\right)\psi_2\left(\frac{q^2}{q^2_2}\right)x\left(q^1_2,q^2_2\right)
\end{aligned}
\tag{7.3.38}
$$

$$
\begin{aligned}
y(q^1,q^2) = {}& \phi_1\left(\frac{q^1}{q^1_2}\right)y\left(q^1_1,q^2\right)+\phi_2\left(\frac{q^1}{q^1_2}\right)y\left(q^1_2,q^2\right)+\psi_1\left(\frac{q^2}{q^2_2}\right)y\left(q^1,q^2_1\right)+\psi_2\left(\frac{q^2}{q^2_2}\right)y\left(q^1,q^2_2\right)\\
&+\phi_1\left(\frac{q^1}{q^1_2}\right)\psi_1\left(\frac{q^2}{q^2_2}\right)y\left(q^1_1,q^2_1\right)+\phi_1\left(\frac{q^1}{q^1_2}\right)\psi_2\left(\frac{q^2}{q^2_2}\right)y\left(q^1_1,q^2_2\right)\\
&+\phi_2\left(\frac{q^1}{q^1_2}\right)\psi_1\left(\frac{q^2}{q^2_2}\right)y\left(q^1_2,q^2_1\right)+\phi_2\left(\frac{q^1}{q^1_2}\right)\psi_2\left(\frac{q^2}{q^2_2}\right)y\left(q^1_2,q^2_2\right)
\end{aligned}
\tag{7.3.39}
$$

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).

$$
\begin{aligned}
x(q^1,q^2) = {}& \left(1-\frac{q^1}{q^1_2}\right)x\left(q^1_1,q^2\right)+\frac{q^1}{q^1_2}x\left(q^1_2,q^2\right)+\left(1-\frac{q^2}{q^2_2}\right)x\left(q^1,q^2_1\right)+\frac{q^2}{q^2_2}x\left(q^1,q^2_2\right)\\
&+\left(1-\frac{q^1}{q^1_2}\right)\left(1-\frac{q^2}{q^2_2}\right)x\left(q^1_1,q^2_1\right)+\left(1-\frac{q^1}{q^1_2}\right)\frac{q^2}{q^2_2}x\left(q^1_1,q^2_2\right)\\
&+\frac{q^1}{q^1_2}\left(1-\frac{q^2}{q^2_2}\right)x\left(q^1_2,q^2_1\right)+\frac{q^1}{q^1_2}\frac{q^2}{q^2_2}x\left(q^1_2,q^2_2\right)
\end{aligned}
\tag{7.3.40}
$$

$$
\begin{aligned}
y(q^1,q^2) = {}& \left(1-\frac{q^1}{q^1_2}\right)y\left(q^1_1,q^2\right)+\frac{q^1}{q^1_2}y\left(q^1_2,q^2\right)+\left(1-\frac{q^2}{q^2_2}\right)y\left(q^1,q^2_1\right)+\frac{q^2}{q^2_2}y\left(q^1,q^2_2\right)\\
&+\left(1-\frac{q^1}{q^1_2}\right)\left(1-\frac{q^2}{q^2_2}\right)y\left(q^1_1,q^2_1\right)+\left(1-\frac{q^1}{q^1_2}\right)\frac{q^2}{q^2_2}y\left(q^1_1,q^2_2\right)\\
&+\frac{q^1}{q^1_2}\left(1-\frac{q^2}{q^2_2}\right)y\left(q^1_2,q^2_1\right)+\frac{q^1}{q^1_2}\frac{q^2}{q^2_2}y\left(q^1_2,q^2_2\right)
\end{aligned}
\tag{7.3.41}
$$

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).

$$
\begin{aligned}
x = & \left(1-\frac{q^1}{q^1_2}\right)x_{AB} + \frac{q^1}{q^1_2}x_{DC} + \left(1-\frac{q^2}{q^2_2}\right)x_{AD} + \frac{q^2}{q^2_2}x_{BC} \\
& + \left(1-\frac{q^1}{q^1_2}\right)\left(1-\frac{q^2}{q^2_2}\right)x_A + \left(1-\frac{q^1}{q^1_2}\right)\frac{q^2}{q^2_2}x_B + \frac{q^1}{q^1_2}\left(1-\frac{q^2}{q^2_2}\right)x_D + \frac{q^1}{q^1_2}\frac{q^2}{q^2_2}x_C
\end{aligned} \tag{7.3.42}
$$

$$
\begin{aligned}
y = & \left(1-\frac{q^1}{q^1_2}\right)y_{AB} + \frac{q^1}{q^1_2}y_{DC} + \left(1-\frac{q^2}{q^2_2}\right)y_{AD} + \frac{q^2}{q^2_2}y_{BC} \\
& + \left(1-\frac{q^1}{q^1_2}\right)\left(1-\frac{q^2}{q^2_2}\right)y_A + \left(1-\frac{q^1}{q^1_2}\right)\frac{q^2}{q^2_2}y_B + \frac{q^1}{q^1_2}\left(1-\frac{q^2}{q^2_2}\right)y_D + \frac{q^1}{q^1_2}\frac{q^2}{q^2_2}y_C
\end{aligned} \tag{7.3.43}
$$

7.3.2 Elliptic Generation System

The equation to be discretised to obtain the improved grid is equation (7.3.44) [3].

$$g^{ij}\frac{\partial^2 \mathbf{r}}{\partial q^i \partial q^j} + P^j\frac{\partial \mathbf{r}}{\partial \xi^j} = 0 \tag{7.3.44}$$

g^{ij} is the contravariant tensor components, $P^j = \nabla^2 q^j$ are the control functions. Einstein summation notation is used [16][38].

$$g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial \mathbf{r}}{\partial q^j}\right) + \nabla^2 q^j\frac{\partial \mathbf{r}}{\partial q^j} = \mathbf{0} \tag{7.3.45}$$

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.

$$g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial}{\partial q^j}(x\mathbf{e}_x + y\mathbf{e}_y)\right) + \nabla^2 q^j\frac{\partial}{\partial q^j}(x\mathbf{e}_x + y\mathbf{e}_y) = \mathbf{0}$$

$$g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial}{\partial q^j}(x\mathbf{e}_x)\right) + g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial}{\partial q^j}(y\mathbf{e}_y)\right) + \nabla^2 q^j\frac{\partial}{\partial q^j}(x\mathbf{e}_x) + \nabla^2 q^j\frac{\partial}{\partial q^j}(y\mathbf{e}_y) = \mathbf{0}$$

$$g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial x}{\partial q^j}\right)\mathbf{e}_x + g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial y}{\partial q^j}\right)\mathbf{e}_y + \nabla^2 q^j\frac{\partial x}{\partial q^j}\mathbf{e}_x + \nabla^2 q^j\frac{\partial y}{\partial q^j}\mathbf{e}_y = \mathbf{0} \tag{7.3.46}$$

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).

$$
\begin{aligned}
& g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial x}{\partial q^j}\right)\mathbf{e}_x\cdot\mathbf{e}_x + g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial y}{\partial q^j}\right)\mathbf{e}_y\cdot\mathbf{e}_x \\
& \qquad + \nabla^2 q^j\frac{\partial x}{\partial q^j}\mathbf{e}_x\cdot\mathbf{e}_x + \nabla^2 q^j\frac{\partial y}{\partial q^j}\mathbf{e}_y\cdot\mathbf{e}_x = \mathbf{0}\cdot\mathbf{e}_x
\end{aligned} \tag{7.3.47}
$$

$$g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial x}{\partial q^j}\right) + \nabla^2 q^j \frac{\partial x}{\partial q^j} = 0 \tag{7.3.48}$$

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).

$$g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial x}{\partial q^j}\right)\mathbf{e}_x \cdot \mathbf{e}_y + g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial y}{\partial q^j}\right)\mathbf{e}_y \cdot \mathbf{e}_y + \nabla^2 q^j \frac{\partial x}{\partial q^j}\mathbf{e}_x \cdot \mathbf{e}_y + \nabla^2 q^j \frac{\partial y}{\partial q^j}\mathbf{e}_y \cdot \mathbf{e}_y = \mathbf{0} \cdot \mathbf{e}_y \tag{7.3.49}$$

$$g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial y}{\partial q^j}\right) + \nabla^2 q^j \frac{\partial y}{\partial q^j} = 0 \tag{7.3.50}$$

The above equations are written using Einstein's summation notation, and these summations as shown in equations (7.3.51) and (7.3.52).

$$\sum_{i=1}^{2}\sum_{j=1}^{2}\left(g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial x}{\partial q^j}\right) + \nabla^2 q^j \frac{\partial x}{\partial q^j}\right) = 0 \tag{7.3.51}$$

$$\sum_{i=1}^{2}\sum_{j=1}^{2}\left(g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial y}{\partial q^j}\right) + \nabla^2 q^j \frac{\partial y}{\partial q^j}\right) = 0 \tag{7.3.52}$$

Taking the sums yields equations (7.3.53) and (7.3.54) for the x- and y-component respectively.

$$g^{11}\frac{\partial}{\partial q^1}\left(\frac{\partial x}{\partial q^1}\right) + g^{12}\frac{\partial}{\partial q^1}\left(\frac{\partial x}{\partial q^2}\right) + g^{21}\frac{\partial}{\partial q^2}\left(\frac{\partial x}{\partial q^1}\right) + g^{22}\frac{\partial}{\partial q^2}\left(\frac{\partial x}{\partial q^2}\right) + \nabla^2 q^1 \frac{\partial x}{\partial q^1} + \nabla^2 q^2 \frac{\partial x}{\partial q^2} = 0 \tag{7.3.53}$$

$$g^{11}\frac{\partial}{\partial q^1}\left(\frac{\partial y}{\partial q^1}\right) + g^{12}\frac{\partial}{\partial q^1}\left(\frac{\partial y}{\partial q^2}\right) + g^{21}\frac{\partial}{\partial q^2}\left(\frac{\partial y}{\partial q^1}\right) + g^{22}\frac{\partial}{\partial q^2}\left(\frac{\partial y}{\partial q^2}\right) + \nabla^2 q^1 \frac{\partial y}{\partial q^1} + \nabla^2 q^2 \frac{\partial y}{\partial q^2} = 0 \tag{7.3.54}$$

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).

$$\left.\frac{\partial \varphi}{\partial q^1}\right|_{i,j} = \frac{\varphi_{i+1,j} - \varphi_{i-1,j}}{2\delta q^1} \tag{7.3.55}$$

$$\left.\frac{\partial \varphi}{\partial q^2}\right|_{i,j} = \frac{\varphi_{i,j+1} - \varphi_{i,j-1}}{2\delta q^2} \tag{7.3.56}$$

$$\left.\frac{\partial^2 \varphi}{\left(\partial q^1\right)^2}\right|_{i,j} = \frac{\varphi_{i+1,j} + \varphi_{i-1,j} - 2\varphi_{i,j}}{\left(\delta q^1\right)^2} \tag{7.3.57}$$

$$\left.\frac{\partial^2 \varphi}{\left(\partial q^2\right)^2}\right|_{i,j} = \frac{\varphi_{i,j+1} + \varphi_{i,j-1} - 2\varphi_{i,j}}{\left(\delta q^2\right)^2} \tag{7.3.58}$$

$$\left.\frac{\partial^2 \varphi}{\partial q^1 \partial q^2}\right|_{i,j} = \frac{\varphi_{i+1,j+1} + \varphi_{i-1,j-1} - \varphi_{i+1,j-1} - \varphi_{i-1,j+1}}{4\delta q^1 \delta q^2} \tag{7.3.59}$$

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).

$$\left.\frac{\partial \varphi}{\partial q^1}\right|_{i,j} = \frac{\varphi_{i+1,j} - \varphi_{i-1,j}}{2} \tag{7.3.60}$$

$$\left.\frac{\partial \varphi}{\partial q^2}\right|_{i,j} = \frac{\varphi_{i,j+1} - \varphi_{i,j-1}}{2} \tag{7.3.61}$$

$$\left.\frac{\partial^2 \varphi}{\left(\partial q^1\right)^2}\right|_{i,j} = \left(\varphi_{i+1,j} + \varphi_{i-1,j} - 2\varphi_{i,j}\right) \tag{7.3.62}$$

$$\left.\frac{\partial^2 \varphi}{\left(\partial q^2\right)^2}\right|_{i,j} = \left(\varphi_{i,j+1} + \varphi_{i,j-1} - 2\varphi_{i,j}\right) \tag{7.3.63}$$

$$\left.\frac{\partial^2 \varphi}{\partial q^1 \partial q^2}\right|_{i,j} = \frac{\varphi_{i+1,j+1} + \varphi_{i-1,j-1} - \varphi_{i+1,j-1} - \varphi_{i-1,j+1}}{4} \tag{7.3.64}$$

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).

$$\begin{aligned} g^{11}\left(x_{i+1,j} + x_{i-1,j} - 2x_{i,j}\right) + g^{12}\frac{x_{i+1,j+1} + x_{i-1,j-1} - x_{i+1,j-1} - x_{i-1,j+1}}{4} \\ + g^{21}\frac{x_{i+1,j+1} + x_{i-1,j-1} - x_{i+1,j-1} - x_{i-1,j+1}}{4} + g^{22}\left(x_{i,j+1} + x_{i,j-1} - 2x_{i,j}\right) \\ + \nabla^2 q^1 \frac{y_{i+1,j} - y_{i-1,j}}{2} + \nabla^2 q^2 \frac{y_{i,j+1} - y_{i,j-1}}{2} = 0 \end{aligned} \tag{7.3.65}$$

$$\begin{aligned} g^{11}\left(y_{i+1,j} + y_{i-1,j} - 2y_{i,j}\right) + g^{12}\frac{y_{i+1,j+1} + y_{i-1,j-1} - y_{i+1,j-1} - y_{i-1,j+1}}{4} \\ + g^{21}\frac{y_{i+1,j+1} + y_{i-1,j-1} - y_{i+1,j-1} - y_{i-1,j+1}}{4} + g^{22}\left(y_{i,j+1} + y_{i,j-1} - 2y_{i,j}\right) \\ + \nabla^2 q^1 \frac{y_{i+1,j} - y_{i-1,j}}{2} + \nabla^2 q^2 \frac{y_{i,j+1} - y_{i,j-1}}{2} = 0 \end{aligned} \tag{7.3.66}$$

Rearranged to gather the same terms, equations (7.3.65) and (7.3.66) become equations (7.3.67) and (7.3.68).

$$\begin{aligned} x_{i,j}\left(-2g^{11}-2g^{22}\right) + x_{i+1,j}\left(g^{11}+\frac{\nabla^2 q^1}{2}\right) + x_{i-1,j}\left(g^{11}-\frac{\nabla^2 q^1}{2}\right) \\ + x_{i,j+1}\left(g^{22}+\frac{\nabla^2 q^2}{2}\right) + x_{i,j-1}\left(g^{22}-\frac{\nabla^2 q^2}{2}\right) \\ + x_{i+1,j+1}\left(\frac{g^{12}}{4}+\frac{g^{21}}{4}\right) + x_{i-1,j+1}\left(-\frac{g^{12}}{4}-\frac{g^{21}}{4}\right) \\ + x_{i+1,j-1}\left(-\frac{g^{12}}{4}-\frac{g^{21}}{4}\right) + x_{i-1,j-1}\left(\frac{g^{12}}{4}+\frac{g^{21}}{4}\right) = 0 \end{aligned} \quad (7.3.67)$$

$$\begin{aligned} y_{i,j}\left(-2g^{11}-2g^{22}\right) + y_{i+1,j}\left(g^{11}+\frac{\nabla^2 q^1}{2}\right) + y_{i-1,j}\left(g^{11}-\frac{\nabla^2 q^1}{2}\right) \\ + y_{i,j+1}\left(g^{22}+\frac{\nabla^2 q^2}{2}\right) + y_{i,j-1}\left(g^{22}-\frac{\nabla^2 q^2}{2}\right) \\ + y_{i+1,j+1}\left(\frac{g^{12}}{4}+\frac{g^{21}}{4}\right) + y_{i-1,j+1}\left(-\frac{g^{12}}{4}-\frac{g^{21}}{4}\right) \\ + y_{i+1,j-1}\left(-\frac{g^{12}}{4}-\frac{g^{21}}{4}\right) + y_{i-1,j-1}\left(\frac{g^{12}}{4}+\frac{g^{21}}{4}\right) = 0 \end{aligned} \quad (7.3.68)$$

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.

$$\begin{aligned} c^x_{i,j}x_{i,j} + c^x_{i+1,j}x_{i+1,j} + c^x_{i-1,j}x_{i-1,j} + c^x_{i,j+1}x_{i,j+1} + c^x_{i,j-1}x_{i,j-1} \\ + c^x_{i+1,j+1}x_{i+1,j+1} + c^x_{i-1,j+1}x_{i-1,j+1} + c^x_{i+1,j-1}x_{i+1,j-1} + c^x_{i-1,j-1}x_{i-1,j-1} = 0 \end{aligned} \quad (7.3.69)$$

with

$$c^x_{i,j} = -2g^{11} - 2g^{22} \quad (7.3.70)$$

$$c^x_{i+1,j} = g^{11} + \frac{\nabla^2 q^1}{2} \quad (7.3.71)$$

$$c^x_{i-1,j} = g^{11} - \frac{\nabla^2 q^1}{2} \quad (7.3.72)$$

$$c^x_{i,j+1} = g^{22} + \frac{\nabla^2 q^2}{2} \quad (7.3.73)$$

$$c^x_{i,j-1} = g^{22} - \frac{\nabla^2 q^2}{2} \quad (7.3.74)$$

$$c^x_{i+1,j+1} = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.75)$$

$$c^x_{i-1,j+1} = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.76)$$

$$c^x_{i+1,j-1} = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.77)$$

$$c^x_{i-1,j-1} = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.78)$$

Equation (7.3.79) shows the discretised elliptic grid generation for the y-component.

$$c^y_{i,j}y_{i,j} + c^y_{i+1,j}y_{i+1,j} + c^y_{i-1,j}y_{i-1,j} + c^y_{i,j+1}y_{i,j+1} + c^y_{i,j-1}y_{i,j-1} \\ + c^y_{i+1,j+1}y_{i+1,j+1} + c^y_{i-1,j+1}y_{i-1,j+1} + c^y_{i+1,j-1}y_{i+1,j-1} + c^y_{i-1,j-1}y_{i-1,j-1} = 0 \quad (7.3.79)$$

with

$$c^y_{i,j} = -2g^{11} - 2g^{22} \quad (7.3.80)$$

$$c^y_{i+1,j} = g^{11} + \frac{\nabla^2 q^1}{2} \quad (7.3.81)$$

$$c^y_{i-1,j} = g^{11} - \frac{\nabla^2 q^1}{2} \quad (7.3.82)$$

$$c^y_{i,j+1} = g^{22} + \frac{\nabla^2 q^2}{2} \quad (7.3.83)$$

$$c^y_{i,j-1} = g^{22} - \frac{\nabla^2 q^2}{2} \quad (7.3.84)$$

$$c^y_{i+1,j+1} = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.85)$$

$$c^y_{i-1,j+1} = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.86)$$

$$c^y_{i+1,j-1} = -\frac{g^{12}}{4} - \frac{g^{21}}{4} \quad (7.3.87)$$

$$c^y_{i-1,j-1} = \frac{g^{12}}{4} + \frac{g^{21}}{4} \quad (7.3.88)$$

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].

$$\mathbf{A}^{(k)} = A^k_j \mathbf{e}_j = \mathbf{g}_l \times \mathbf{g}_m \quad (7.3.90)$$

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].

$$\varepsilon_{klm} = \begin{cases} +1 \rightarrow klm = 123, 231 \text{ or } 312 \\ -1 \rightarrow klm = 321, 213 \text{ or } 132 \\ \ \ 0 \rightarrow \text{any indeces are the same} \end{cases} \quad (7.3.91)$$

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).

$$\mathbf{g}_i = \frac{\partial x^j}{\partial q^i}\mathbf{e}_j \tag{7.3.92}$$

where $\frac{\partial x^j}{\partial q^i}$ can also be noted J_i^j as defined by equation (7.3.93).

$$J_i^j = \frac{\partial x^j}{\partial q^i} \tag{7.3.93}$$

Equation (7.3.90) can then be rewritten to yield equation (7.3.94) by use of equation (7.3.92).

$$\mathbf{A}^{(k)} = \frac{\partial x^p}{\partial q^l}\mathbf{e}_p \times \frac{\partial x^q}{\partial q^m}\mathbf{e}_q \tag{7.3.94}$$

The indeces 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]

$$\begin{aligned}\mathbf{A}^{(k)} &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\mathbf{e}_p \times \mathbf{e}_q \\ &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\varepsilon_{pqr}\mathbf{e}_r\end{aligned} \tag{7.3.95}$$

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).

$$A_i^{(k)} = \mathbf{A}^{(k)} \cdot \mathbf{e}_i \tag{7.3.96}$$

This yields equations (7.3.97), (7.3.98) and (7.3.99) for the three components.

$$\begin{aligned}A_1^{(k)} &= \mathbf{A}^{(k)} \cdot \mathbf{e}_1 \\ &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\varepsilon_{pqr}\mathbf{e}_r \cdot \mathbf{e}_1 \\ &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\varepsilon_{pq1} \\ &= \frac{\partial x^2}{\partial q^l}\frac{\partial x^3}{\partial q^m} - \frac{\partial x^3}{\partial q^l}\frac{\partial x^2}{\partial q^m}\end{aligned} \tag{7.3.97}$$

$$\begin{aligned}A_2^{(k)} &= \mathbf{A}^{(k)} \cdot \mathbf{e}_2 \\ &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\varepsilon_{pqr}\mathbf{e}_r \cdot \mathbf{e}_2 \\ &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\varepsilon_{pq2} \\ &= \frac{\partial x^3}{\partial q^l}\frac{\partial x^1}{\partial q^m} - \frac{\partial x^1}{\partial q^l}\frac{\partial x^3}{\partial q^m}\end{aligned} \tag{7.3.98}$$

$$\begin{aligned} A_3^{(k)} &= \mathbf{A}^{(k)} \cdot \mathbf{e}_3 \\ &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\varepsilon_{pqr}\mathbf{e}_r \cdot \mathbf{e}_3 \\ &= \frac{\partial x^p}{\partial q^l}\frac{\partial x^q}{\partial q^m}\varepsilon_{pq3} \\ &= \frac{\partial x^1}{\partial q^l}\frac{\partial x^2}{\partial q^m} - \frac{\partial x^2}{\partial q^l}\frac{\partial x^1}{\partial q^m} \end{aligned} \tag{7.3.99}$$

Further, all the nine compontents 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).

$$J = \frac{1}{4}\left(x_{i+1,j} - x_{i-1,j}\right)\left(y_{i,j+1} - y_{i,j-1}\right) - \frac{1}{4}\left(y_{i+1,j} - y_{i-1,j}\right)\left(x_{i,j+1} - x_{i,j-1}\right) \quad (7.3.120)$$

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.

$$P^i = \nabla^2 q^i \qquad i = 1, 2 \quad (7.3.126)$$

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` (7.4.1)

`q2 = 0:M` (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` (7.4.3)

`y_max = 2` (7.4.4)

`step_h = 1` (7.4.5)

`step_w = 5` (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`	(7.4.7)	`yA = step_h`	(7.4.13)
`xB = 0`	(7.4.8)	`yB = y_max`	(7.4.14)
`xC = x_max`	(7.4.9)	`yC = y_max`	(7.4.15)
`xD = x_max`	(7.4.10)	`yD = 0`	(7.4.16)
`xE = step_w`	(7.4.11)	`yE = 0`	(7.4.17)
`xF = step_w`	(7.4.12)	`yF = step_h`	(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 (A)(B), (B)(C) and (D)(C) in figure 7.2.

$$\text{ⒶⒷ}: \; x_{AB} = \left(1 - \frac{q^2}{q_2^2}\right) x_A + \frac{q^2}{q_2^2} x_B \tag{7.4.19}$$

$$y_{AB} = \left(1 - \frac{q^2}{q_2^2}\right) y_A + \frac{q^2}{q_2^2} y_B \tag{7.4.20}$$

$$\text{ⒷⒸ}: \; x_{BC} = \left(1 - \frac{q^1}{q_2^1}\right) x_B + \frac{q^1}{q_2^1} x_C \tag{7.4.21}$$

$$y_{BC} = \left(1 - \frac{q^1}{q_2^1}\right) y_B + \frac{q^1}{q_2^1} y_C \tag{7.4.22}$$

$$\text{ⒸⒹ}: \; x_{DC} = \left(1 - \frac{q^2}{q_2^2}\right) x_D + \frac{q^2}{q_2^2} x_C \tag{7.4.23}$$

$$y_{DC} = \left(1 - \frac{q^2}{q_2^2}\right) y_D + \frac{q^2}{q_2^2} y_C \tag{7.4.24}$$

The line segment ⒶⒹ needs to be split into the three line segments ⒶⒻ, ⒻⒺ and ⒺⒹ for this calculation. The placement of the points Ⓔ and Ⓕ in the computational domain determines how the boundary points between Ⓐ and Ⓓ 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 ⒶⒹ. Vectors of coordinates q^1 for the line segments ⒶⒻ,ⒻⒺ and ⒺⒹ noted q^1_{AF}, q^1_{FE} and q^1_{ED} were then created as shown in equations (7.4.25)-(7.4.27).

```
q1AF = 0:AFpoints                    (7.4.25)
q1FE = 0:FEpoints                    (7.4.26)
q1ED = 0:(N-AFpoints-FEpoints)       (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^1_2 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 ⒶⒻ,ⒻⒺ and ⒺⒹ are then found by equations (7.4.28)-(7.4.33).

$$\text{ⒶⒻ}: \; x_{AF} = \left(1 - \frac{q^1_{AF}}{q^1_{AF,2}}\right) x_A + \frac{q^1_{AF}}{q^1_{AF,2}} x_F \tag{7.4.28}$$

$$y_{AF} = \left(1 - \frac{q^1_{AF}}{q^1_{AF,2}}\right) y_A + \frac{q^1_{AF}}{q^1_{AF,2}} y_F \tag{7.4.29}$$

$$\text{ⒻⒺ}: \; x_{FE} = \left(1 - \frac{q^1_{FE}}{q^1_{FE,2}}\right) x_F + \frac{q^1_{FE}}{q^1_{FE,2}} x_E \tag{7.4.30}$$

$$y_{FE} = \left(1 - \frac{q^1_{FE}}{q^1_{FE,2}}\right) y_F + \frac{q^1_{FE}}{q^1_{FE,2}} y_E \tag{7.4.31}$$

$$\text{ⒺⒹ}: \; x_{ED} = \left(1 - \frac{q^1_{ED}}{q^1_{ED,2}}\right) x_E + \frac{q^1_{ED}}{q^1_{ED,2}} x_D \tag{7.4.32}$$

$$y_{ED} = \left(1 - \frac{q^1_{ED}}{q^1_{ED,2}}\right) y_E + \frac{q^1_{ED}}{q^1_{ED,2}} y_D \tag{7.4.33}$$

For the calculation of the centre points, the line segment (A)(D) must be put back together. This is done as shown in equations (7.4.34) and (7.4.35).

```
xAD = [xAF xFE(2:end-1) xED]
```
(7.4.34)

```
yAD = [yAF yFE(2:end-1) yED]
```
(7.4.35)

Note that since each sub-line segment go from a corner point to another, points (E) and (F) are overlapped. This is solved by taking only the points from position 2 to `end-1` for the line segment (E)(F). `xAD` and `yAD` 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.

$$x = \left(1 - \frac{q^1}{q^1_2}\right) x_{AB} + \frac{q^1}{q^1_2} x_{DC} + \left(1 - \frac{q^2}{q^2_2}\right) x_{AD} + \frac{q^2}{q^2_2} x_{BC} + \left(1 - \frac{q^1}{q^1_2}\right)\left(1 - \frac{q^2}{q^2_2}\right) x_A + \left(1 - \frac{q^1}{q^1_2}\right)\frac{q^2}{q^2_2} x_B + \frac{q^1}{q^1_2}\left(1 - \frac{q^2}{q^2_2}\right) x_D + \frac{q^1}{q^1_2}\frac{q^2}{q^2_2} x_C \quad (7.4.36)$$

$$y = \left(1 - \frac{q^1}{q^1_2}\right) y_{AB} + \frac{q^1}{q^1_2} y_{DC} + \left(1 - \frac{q^2}{q^2_2}\right) y_{AD} + \frac{q^2}{q^2_2} y_{BC} + \left(1 - \frac{q^1}{q^1_2}\right)\left(1 - \frac{q^2}{q^2_2}\right) y_A + \left(1 - \frac{q^1}{q^1_2}\right)\frac{q^2}{q^2_2} y_B + \frac{q^1}{q^1_2}\left(1 - \frac{q^2}{q^2_2}\right) y_D + \frac{q^1}{q^1_2}\frac{q^2}{q^2_2} y_C \quad (7.4.37)$$

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.

```
for j =1:length(q2)
    for i = 1:length(q1)
        x(j,i) = (1-q1(i)/q1(end))*xAB(j) +(q1(i)/q1(end))*xDC(j)...
            +(1-q2(j)/q2(end))*xAD(i) +(q2(j)/q2(end))*xBC(i)...
            -(1-q1(i)/q1(end))*(1-q2(j)/q2(end))*xA...
            -(1-q1(i)/q1(end))*(q2(j)/q2(end))*xB...
            -(q1(i)/q1(end))*(1-q2(j)/q2(end))*xD...
            -(q1(i)/q1(end))*(q2(j)/q2(end))*xC;
        y(j,i) = (1-q1(i)/q1(end))*yAB(j) +(q1(i)/q1(end))*yDC(j)...
            +(1-q2(j)/q2(end))*yAD(i) +(q2(j)/q2(end))*yBC(i)...
            -(1-q1(i)/q1(end))*(1-q2(j)/q2(end))*yA...
            -(1-q1(i)/q1(end))*(q2(j)/q2(end))*yB...
            -(q1(i)/q1(end))*(1-q2(j)/q2(end))*yD...
            -(q1(i)/q1(end))*(q2(j)/q2(end))*yC;
    end %for
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 *devided 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).

$$c^x_{i,j}x_{i,j} + \sum c^x_{nb}x_{nb} = \sum -c^x_{edge}x_{edge} \tag{7.4.38}$$

$$c^y_{i,j}y_{i,j} + \sum c^y_{nb}y_{nb} = \sum -c^y_{edge}y_{edge} \tag{7.4.39}$$

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.

$$x^{new} = (1-\alpha)x + \alpha x^{\circ} \tag{7.4.40}$$

$$y^{new} = (1-\alpha)y + \alpha y^{\circ} \tag{7.4.41}$$

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).

$$C_x = \max\left(|x - x^{\circ}|\right) \tag{7.4.42}$$

$$C_y = \max\left(|y - y^{\circ}|\right) \tag{7.4.43}$$

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

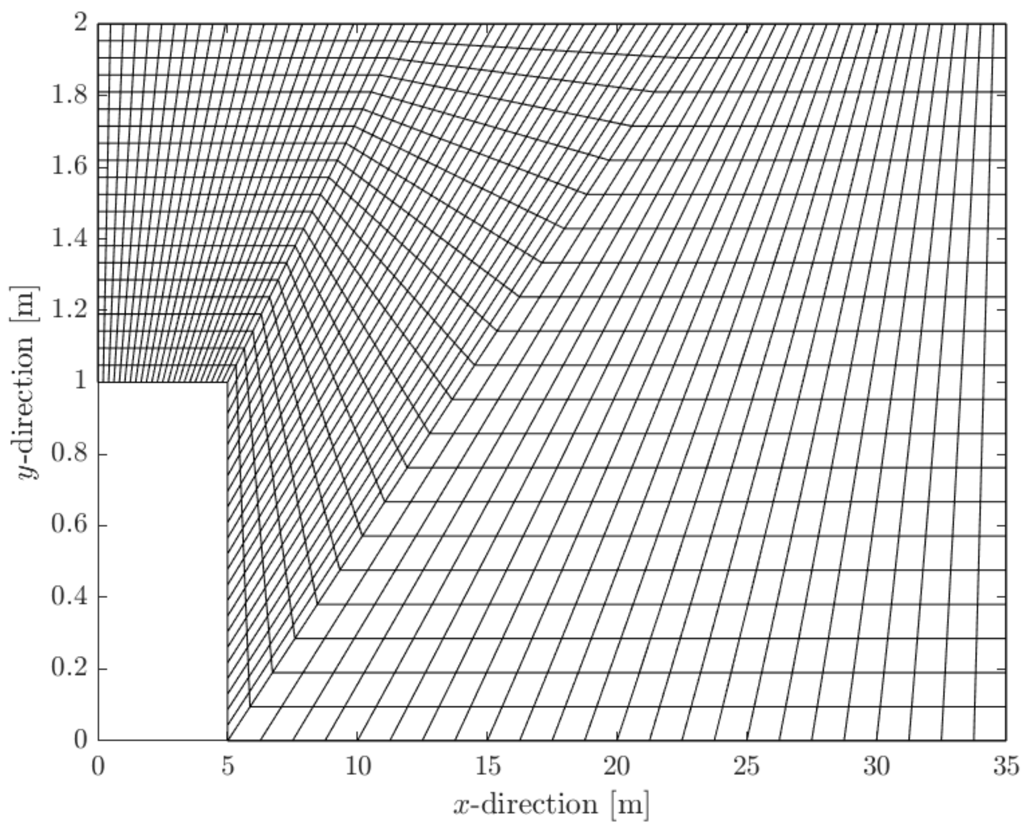


Figure 7.3: Grid obtained by transfinite interpolation.

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.

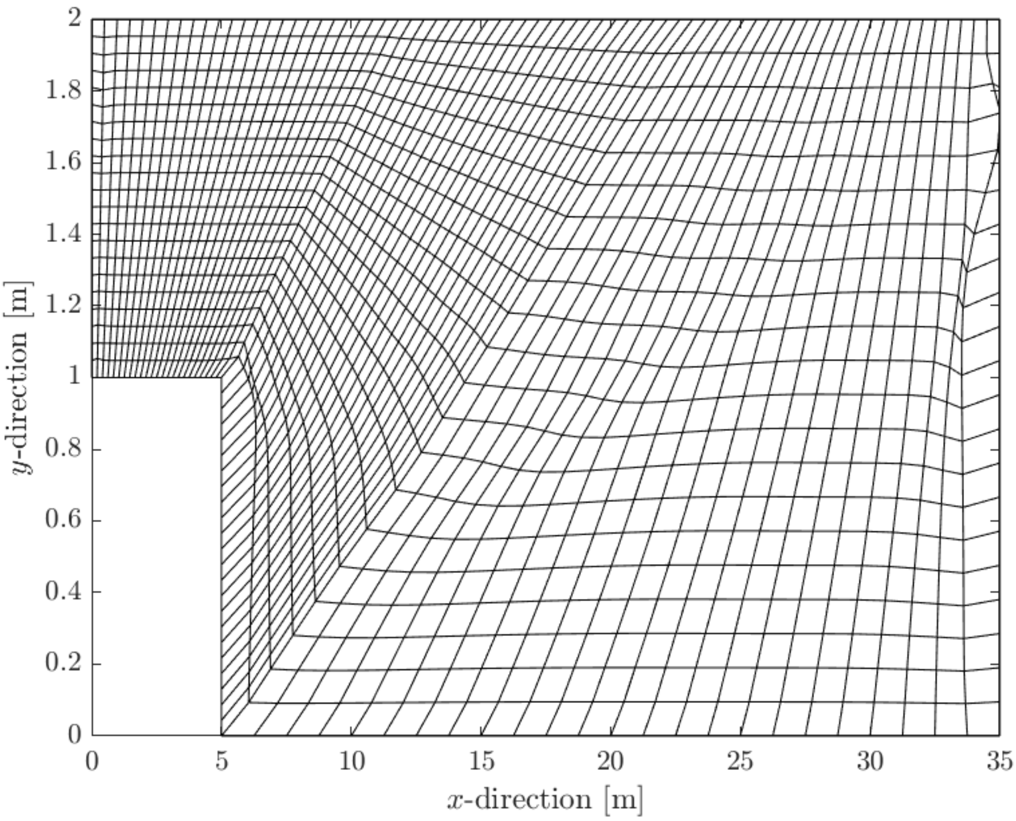


Figure 7.4: Elliptic grid after 100 iterations.

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 merhods 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.RShao, 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

A

Governing Equations

A.1 The Mass Based Equation of Continuity

The continuity equation in vector form is shown in equation A.1.1 [16].

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{A.1.1}$$

A.2 The Equation of Motion

The momentum equation in vector form is shown in equation A.2.1 [16].

$$\frac{\partial}{\partial t}(\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p - \nabla \cdot \boldsymbol{\sigma} + \rho \mathbf{g} \tag{A.2.1}$$

The x-component of the two dimensional momentum equation is shown in equation A.2.2.

$$\frac{\partial}{\partial t}(\rho u) + \frac{\partial}{\partial x}(\rho u u) + \frac{\partial}{\partial y}(\rho v u) = -\frac{\partial p}{\partial x} - \frac{\partial \sigma_{xx}}{\partial x} - \frac{\partial \sigma_{yx}}{\partial y} + \rho g_x \tag{A.2.2}$$

The y-component of the two dimensional momentum equation is shown in equation A.2.3.

$$\frac{\partial}{\partial t}(\rho v) + \frac{\partial}{\partial x}(\rho u v) + \frac{\partial}{\partial y}(\rho v v) = -\frac{\partial p}{\partial y} - \frac{\partial \sigma_{xy}}{\partial x} - \frac{\partial \sigma_{yy}}{\partial y} + \rho g_y \tag{A.2.3}$$

The stress tensors σ for two dimensional systems are shown in equations A.2.4, A.2.5 and A.2.6.

$$\sigma_{xx} = -\mu \left[2\frac{\partial u}{\partial x} - \frac{2}{3}\Big(\nabla \cdot \mathbf{v}\Big) \right] \tag{A.2.4}$$

$$\sigma_{yy} = -\mu \left[2\frac{\partial v}{\partial y} - \frac{2}{3}\Big(\nabla \cdot \mathbf{v}\Big) \right] \tag{A.2.5}$$

$$\sigma_{xy} = \sigma_{yx} = -\mu \left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right] \tag{A.2.6}$$

A.3 Other Equations and Theorems

Gauss' theorem

Gauss' theorem is shown in equation A.3.1 [2].

$$\int_{CV} \nabla \cdot \phi \; dV = \int_{A} \mathbf{n} \cdot \phi \; dA \tag{A.3.1}$$

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].

$$\int_{a}^{b} \frac{d}{dx} f(x) dx = f(b) - f(a) \tag{A.3.2}$$

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.

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

$$\int_{A} \mathbf{n} \cdot \Big(\rho \mathbf{u}\Big)\, dA = 0$$

$$\int_{A_e} \mathbf{n} \cdot \Big(\rho \mathbf{u}\Big)\, dA_e + \int_{A_w} \mathbf{n} \cdot \Big(\rho \mathbf{u}\Big)\, dA_w = 0$$

$$\Big(\rho u A\Big)_e - \Big(\rho u A\Big)_w = 0$$

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:

$$\Big(\max(F_w A_w, 0) + \max(0, -F_e A_e) + F_e A_e - F_w A_w\Big)\ u_P \\ - \max(0, -F_e A_e)\ u_E - \max(F_w A_w, 0)\ u_W = \mathbf{RHS}$$

$$\Big(\max(F_w A_w, 0) + \max(0, -F_e A_e) + F_e A_e - F_w A_w\Big)\ u_i \\ - \max(0, -F_e A_e)\ u_{i+1} - \max(F_w A_w, 0)\ u_{i-1} = \mathbf{RHS}$$

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.

$$\mathbf{LHS} = -\nabla p - \sum_i \frac{\partial \boldsymbol{\sigma}_i}{\partial x_i}$$

$$\mathbf{LHS} = -\nabla p - \frac{\partial \boldsymbol{\sigma}_x}{\partial x}$$

$$\mathbf{LHS} = \mathbf{e}_x \cdot \left(-\nabla p - \frac{\partial \boldsymbol{\sigma}_x}{\partial x} \right)$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} - \frac{\partial \sigma_{xx}}{\partial x}$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} - \frac{\partial}{\partial x} \left(-\mu \left[2\frac{\partial u}{\partial x} - \frac{2}{3} \left(\nabla \cdot \mathbf{u} \right) \right] \right)$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} - \frac{\partial}{\partial x} \left(-2\mu \frac{\partial u}{\partial x} \right)$$

$$\mathbf{LHS} = -\int_{CV} \frac{\partial p}{\partial x}\, dV - \int_A \int_{\delta x} \frac{\partial}{\partial x} \left(-2\mu \frac{\partial u}{\partial x} \right)\, dAdx$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} \Delta V - \int_{\delta x} \frac{\partial}{\partial x} \left(-2\mu \frac{\partial u}{\partial x} \right) A\, dx$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} \Delta V - \left(-2\mu \frac{\partial u}{\partial x} A \right)_e + \left(-2\mu \frac{\partial u}{\partial x} A \right)_w$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x} \delta x A + 2\mu \left(\frac{\partial u}{\partial x} A \right)_e - 2\mu \left(\frac{\partial u}{\partial x} A \right)_w$$

The derivatives are approximated using central differences:

$$\left. \frac{\partial p}{\partial x} \right|_i = \frac{p_I - p_{I-1}}{\delta x}$$

$$\left. \frac{\partial u}{\partial x} \right|_e = \frac{u_{i+1} - u_i}{\delta x}$$

$$\left. \frac{\partial u}{\partial x} \right|_w = \frac{u_i - u_{i-1}}{\delta x}$$

The convective mass flux per unit area F and the diffusion conductance D:

$$F = \rho u \qquad\qquad D = \frac{\mu}{\delta x}$$

F at the velocity cell faces is approximated with linear interpolation:

$$F_e = \rho \frac{u_i + u_{i+1}}{2} \qquad\qquad F_w = \rho \frac{u_{i-1} + u_i}{2}$$

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:

$$\textbf{LHS} = -\left(\frac{p_I - p_{I-1}}{\cancel{\delta x}}\right)\cancel{\delta x}A + 2\mu A\left(\frac{u_{i+1} - u_i}{\delta x}\right) - 2\mu A\left(\frac{u_i - u_{i-1}}{\delta x}\right)$$

$$\textbf{LHS} = -\left(p_I - p_{I-1}\right)A + \frac{2\mu A}{\delta x}\Big(u_{i+1} - u_i\Big) - \frac{2\mu A}{\delta x}\Big(u_i - u_{i-1}\Big)$$

$$\textbf{LHS} = -\left(p_I - p_{I-1}\right)A + 2D_eA\Big(u_{i+1} - u_i\Big) - 2D_wA\Big(u_i - u_{i-1}\Big)$$

$$\textbf{LHS} = 2D_eAu_{i+1} - 2D_eAu_i - 2D_wAu_i + 2D_wAu_{i-1} - \left(p_I - p_{I-1}\right)A$$

$$\textbf{LHS} = \left(-2D_eA - 2D_wA\right)u_i + 2D_eAu_{i+1} + 2D_wAu_{i-1} - \left(p_I - p_{I-1}\right)A$$

Final Discretised Momentum Equation

$$\begin{aligned}&\Big(4AD + \max(F_wA, 0) + \max(0, -F_eA) + F_eA - F_wA\Big)\ u_i\ + \\ &\qquad\Big(-2AD - \max(0, -F_eA)\Big)\ u_{i+1} + \Big(-2AD - \max(F_wA, 0)\ \Big)u_{i-1} \\ &\qquad\qquad = -A\left(p_I - p_{I-1}\right)_{cs}\end{aligned}$$

Coefficient form

$$a_iu_i + a_{i-1}u_{i-1} + a_{i+1}u_{i+1} = b_i$$

with

$$\begin{aligned}a_i &= -a_{i-1} - a_{i+1} + F_eA - F_wA \\ a_{i+1} &= -2AD\ - \max(0, -F_eA) \\ a_{i-1} &= -2AD - \max(F_wA, 0) \\ b_i &= -A\left(p_I - p_{I-1}\right)\end{aligned}$$

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.

$$\begin{aligned}&a_i(u_i - u_i^*) + a_{i-1}(u_{i-1} - u_{i-1}^*) + a_{i+1}(u_{i+1} - u_{i+1}^*) = \\ &\qquad\Big(-\left(p_I - p_{I-1}\right) + \Big(p_I^* - p_{I-1}^*\Big)\Big)A + \cancel{b_i - b_i}\end{aligned}$$

$$\begin{aligned}&a_i(u_i - u_i^*) + a_{i-1}(u_{i-1} - u_{i-1}^*) + a_{i+1}(u_{i+1} - u_{i+1}^*) = \\ &\qquad\Big(-p_I + p_{I-1} + p_I^* - p_{I-1}^*\Big)A\end{aligned}$$

omit

$$a_i(u_i - u_i^*) + \cancel{a_{i-1}u_{i-1}' + a_{i+1}u_{i+1}'} = -\Big(p_I' - p_{I-1}'\Big)A$$

$$u_i = u_i^* - \frac{A}{a_i^{centre}}\left(p'_I - p'_{I-1}\right)$$

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.

$$\rho A\left(u_{i+1}^* - \frac{A}{a_{i+1}^{centre}}\left(p'_{I+1} - p'_I\right)\right) - \rho A\left(u_i^* - \frac{A_i}{a_i^{centre}}\left(p'_I - p'_{I-1}\right)\right) = 0$$

$$\rho A u_{i+1}^* - \frac{\rho A^2}{a_{i+1}^{centre}}\left(p'_{I+1} - p'_I\right) - \rho A u_i^* + \frac{\rho A^2}{a_i^{centre}}\left(p'_I - p'_{I-1}\right) = 0$$

$$-\frac{\rho A^2}{a_{i+1}^{centre}}\left(p'_{I+1} - p'_I\right) + \frac{\rho A^2}{a_i^{centre}}\left(p'_I - p'_{I-1}\right) + \rho A u_{i+1}^* - \rho A u_i^* = 0$$

$$p'_I \frac{\rho A^2}{a_{i+1}^{centre}} - p'_{I+1}\frac{\rho A^2}{a_{i+1}^{centre}} + p'_I \frac{\rho A^2}{a_i^{centre}} - p'_{I-1}\frac{\rho A^2}{a_i^{centre}} + \rho A u_{i+1}^* - \rho A u_i^* = 0$$

Pressure correction equation:

$$p'_I \nu_I + p'_{I+1}\nu_{I+1} + p'_{I-1}\nu_{I-1} = \beta_I$$

Coefficients:

$$\nu_I = -\nu_{I+1} - \nu_{I-1} \tag{B.1.1}$$

$$\nu_{I+1} = -\frac{\rho A^2}{a_{i+1}^{centre}} \tag{B.1.2}$$

$$\nu_{I-1} = -\frac{\rho A^2}{a_i^{centre}} \tag{B.1.3}$$

$$\beta_I = -\rho A u_{i+1}^* + \rho A u_i^* \tag{B.1.4}$$

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

$$a_i = -a_{i-1} - a_{i+1} + F_e A - F_w A + 2AD + \max(F_w A, 0)$$

$$a_{i+1} = -2AD - \max(0, -F_e A)$$

$$b_i = -A\left(p_I - p_{I-1}\right) + (2AD + \max(F_w A, 0))u_{in}$$

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.

$$\rho A\left(u_{i+1}^* - \frac{A}{a_{i+1}^{centre}}\left(p'_{I+1} - p'_I\right)\right) - \rho A u_{in} = 0$$

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

$$\begin{aligned}
a_i &= -a_{i-1} + F_e A - F_w A \\
a_{i-1} &= -2AD - \max(F_w A, 0) \\
b_i &= -A\left(p_{out} - p_{I-1}\right)
\end{aligned}$$

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 be 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

$$u_{in} = 1 \cdot 10^{-3} \qquad p_{out} = 1 \cdot 10^5 \tag{B.3.1}$$

$$\alpha_u = 1 \qquad \alpha_p = 0.05 \tag{B.3.2}$$

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).

$$u^\circ = 1.5 \cdot 10^{-3} \left[\text{ m/s } \right] \text{ for all } u \qquad p^\circ = 1.5 \cdot 10^{5} \left[\text{ Pa } \right] \text{ for all } p \tag{B.3.3}$$

Convergence criteria

The convergence criteria used are

$$C_1 < 10^{-6} \tag{B.3.4}$$
$$C_3 < 10^{-6} \tag{B.3.5}$$
$$C_4 < 10^{-6} \tag{B.3.6}$$

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.

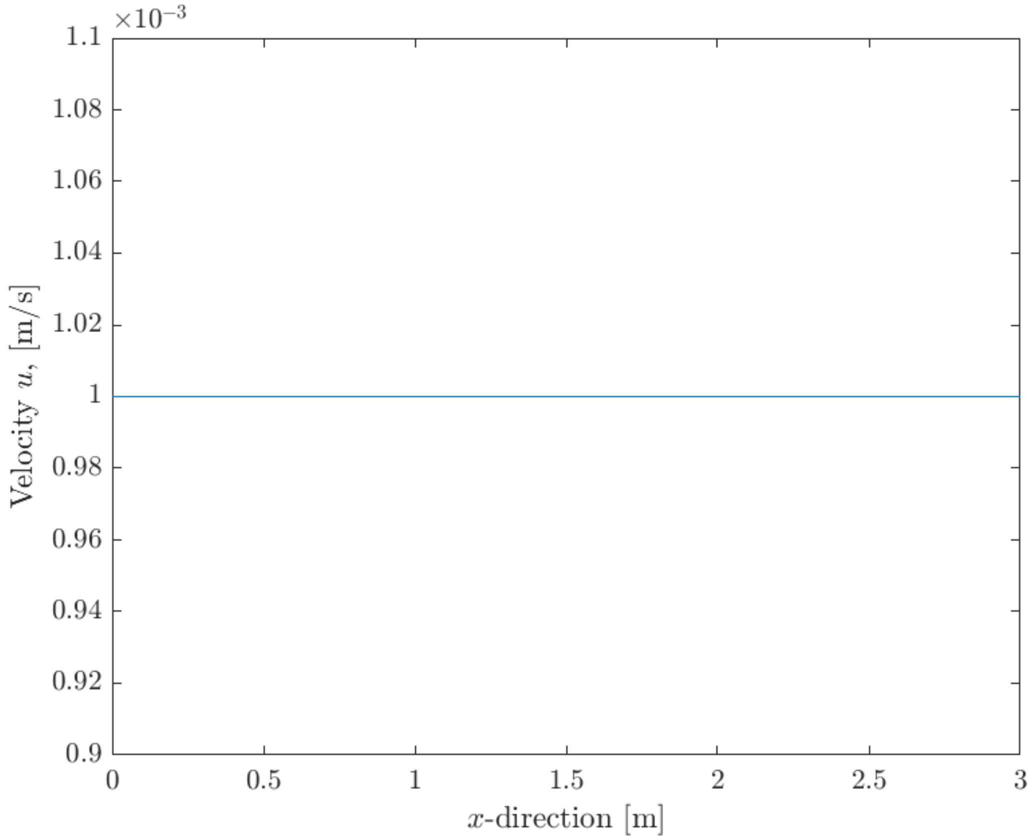


Figure B.1: Velocity profile for the one dimensional model.

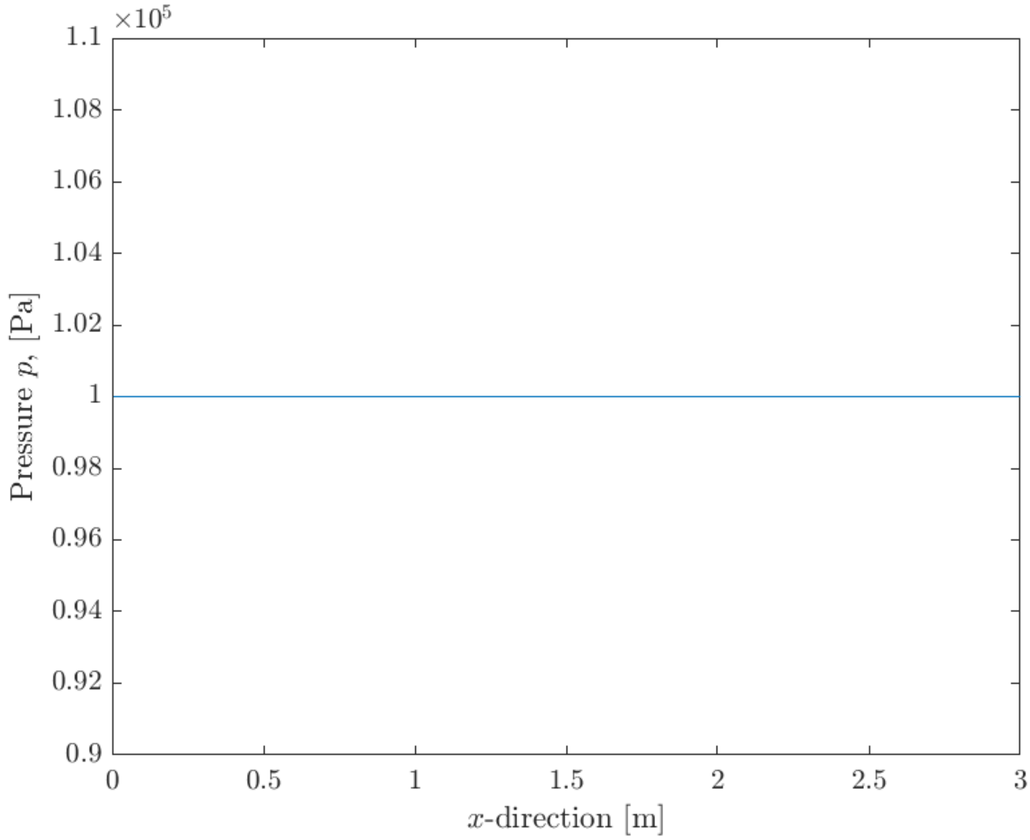


Figure B.2: Pressure profile for the one dimensional model.

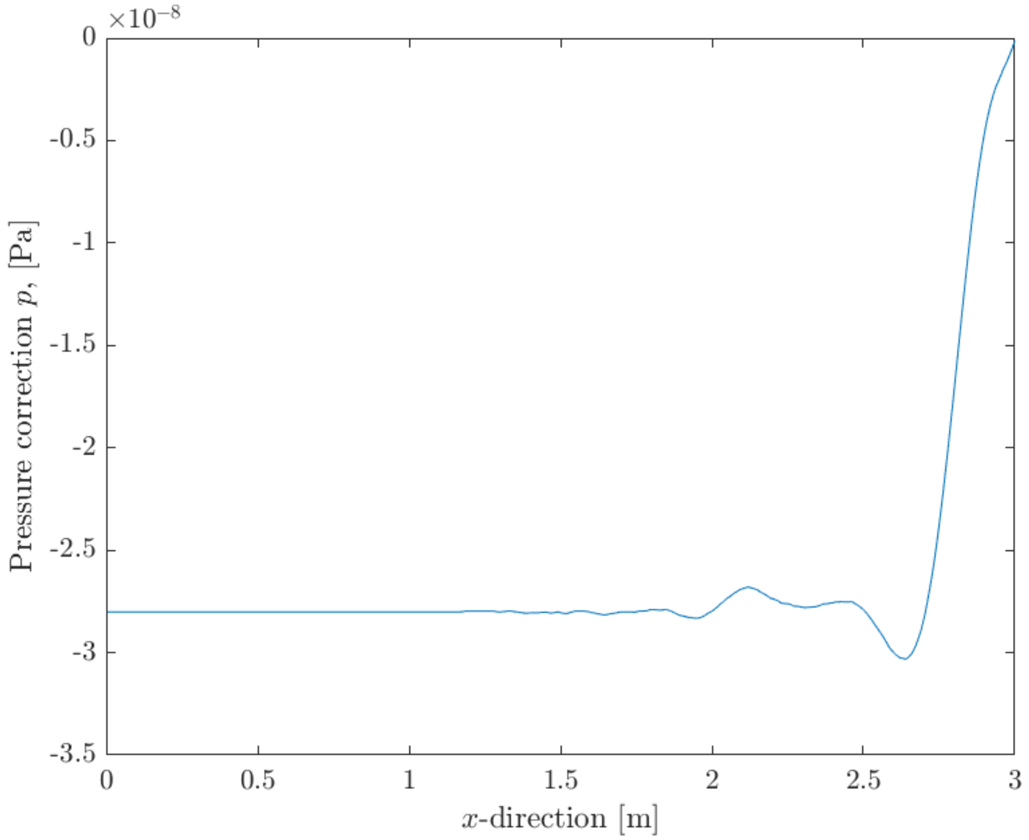


Figure B.3: Pressure correction for the one dimensional model.

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.

$$\nabla \cdot \left(\rho \mathbf{u}\right) = 0$$

$$\int_{CV} \nabla \cdot \left(\rho \mathbf{u}\right) \, dV = 0$$

$$\int_{A} \mathbf{n} \cdot \left(\rho \mathbf{u}\right) \, dA = 0$$

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

$$\int_{A_{x,e}} \rho u \; dA - \int_{A_{x,w}} \rho u \; dA + \int_{A_{y,n}} \rho v \; dA - \int_{A_{y,s}} \rho v \; dA = 0$$

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

C.2 Momentum equation

The transient term is neglected, and the vector form Momentum equation is then

$$\nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p - \nabla \cdot \sigma + \rho \mathbf{g}$$

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.

$$\nabla \cdot (\rho \mathbf{u}\mathbf{u}) = \mathbf{RHS}$$

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

$$\int_{A} \mathbf{n} \cdot (\rho \mathbf{u}\mathbf{u}) \, dA = \mathbf{RHS}$$

$$\int_{A_{x,e}} \mathbf{e}_x \cdot \rho \mathbf{u}\mathbf{u} \; dA + \int_{A_{x,w}} -\mathbf{e}_x \cdot \rho \mathbf{u}\mathbf{u} \; dA + \int_{A_{y,n}} \mathbf{e}_y \cdot \rho \mathbf{u}\mathbf{u} \; dA + \int_{A_{y,s}} -\mathbf{e}_y \cdot \rho \mathbf{u}\mathbf{u} \; dA = \mathbf{RHS}$$

$$\int_{A_{x,e}} \rho u \mathbf{u} \; dA - \int_{A_{x,w}} \rho u \mathbf{u} \; dA + \int_{A_{y,n}} \rho v \mathbf{u} \; dA - \int_{A_{y,s}} \rho v \mathbf{u} \; dA = \mathbf{RHS}$$

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

x-component

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

$$\rho \left(uu\right)_e A_{x,e} - \rho \left(uu\right)_w A_{x,w} + \rho \left(vu\right)_n A_{y,n} - \rho \left(vu\right)_s A_{y,s} = \mathbf{RHS}$$

$$F_{x,e} u_e A_{x,e} - F_{x,w} u_w A_{x,w} + F_{x,n} u_n A_{y,n} - F_{x,s} u_s A_{y,s} = \mathbf{RHS}$$

y-component

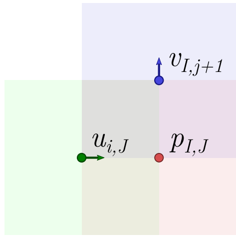
$$\mathbf{e}_y \cdot \left(\rho \left(u\mathbf{u}\right)_e A_{x,e} - \rho \left(u\mathbf{u}\right)_w A_{x,w} + \rho \left(v\mathbf{u}\right)_n A_{y,n} - \rho \left(v\mathbf{u}\right)_s A_{y,s}\right) = \mathbf{RHS}$$

$$\rho \left(uv\right)_e A_{x,e} - \rho \left(uv\right)_w A_{x,w} + \rho \left(vv\right)_n A_{y,n} - \rho \left(vv\right)_s A_{y,s} = \mathbf{RHS}$$

$$F_{y,e} v_e A_{x,e} - F_{y,w} v_w A_{x,w} + F_{y,n} v_n A_{y,n} - F_{y,s} v_s A_{y,s} = \mathbf{RHS}$$

Upwind Differencing

Positive x-flow, Positive y-flow



$$
\begin{aligned}
u_e &= u_P \text{ and } u_w = u_W \\
u_n &= u_P \text{ and } u_s = u_S \\
v_e &= v_P \text{ and } v_w = v_W \\
v_n &= v_P \text{ and } v_s = v_S
\end{aligned}
$$

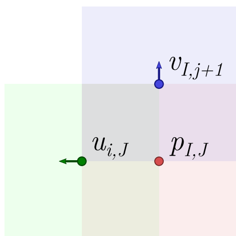
x-component is:

$$F_{x,e}u_PA_{x,e} - F_{x,w}u_WA_{x,w} + F_{y,n}u_PA_{y,n} - F_{y,s}u_SA_{y,s} = \mathbf{RHS}$$

y-component is:

$$F_{x,e}v_PA_{x,e} - F_{x,w}v_WA_{x,w} + F_{y,n}v_PA_{y,n} - F_{y,s}v_SA_{y,s} = \mathbf{RHS}$$

Negative x-flow, Positive y-flow



$$
\begin{aligned}
u_e &= u_E \text{ and } u_w = u_P \\
u_n &= u_P \text{ and } u_s = u_S \\
v_e &= v_E \text{ and } v_w = v_P \\
v_n &= v_P \text{ and } v_s = v_S
\end{aligned}
$$

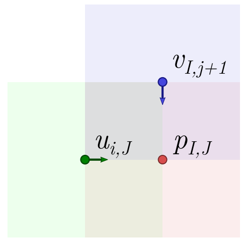
x-component is:

$$F_{x,e}u_EA_{x,e} - F_{x,w}u_PA_{x,w} + F_{y,n}u_PA_{y,n} - F_{y,s}u_SA_{y,s} = \mathbf{RHS}$$

y-component is:

$$F_{x,e}v_EA_{x,e} - F_{x,w}v_PA_{x,w} + F_{y,n}v_PA_{y,n} - F_{y,s}v_SA_{y,s} = \mathbf{RHS}$$

Positive x-flow, Negative y-flow



$$\begin{aligned} u_e &= u_P \text{ and } u_w = u_W \\ u_n &= u_N \text{ and } u_s = u_P \\ v_e &= v_P \text{ and } v_w = v_W \\ v_n &= v_N \text{ and } v_s = v_P \end{aligned}$$

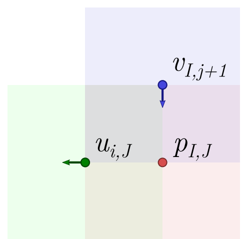
x-component is:

$$F_{x,e}u_P A_{x,e} - F_{x,w}u_W A_{x,w} + F_{y,n}u_N A_{y,n} - F_{y,s}u_P A_{y,s} = \mathbf{RHS}$$

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



$$\begin{aligned} u_e &= u_E \text{ and } u_w = u_P \\ u_n &= u_N \text{ and } u_s = u_P \\ v_e &= v_E \text{ and } v_w = v_P \\ v_n &= v_N \text{ and } v_s = v_P \end{aligned}$$

x-component is:

$$F_{x,e}u_E A_{x,e} - F_{x,w}u_P A_{x,w} + F_{y,n}u_N A_{y,n} - F_{y,s}u_P A_{y,s} = \mathbf{RHS}$$

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:

$$\Big(\max\big(0,-F_{x,e}A_{x,e}\big)+\max\big(F_{x,w}A_{x,w},0\big)+\max\big(0,-F_{y,n}A_{y,n}\big)+\max\big(F_{y,s}A_{y,s},0\big)$$
$$+F_{x,e}A_{x,e}-F_{x,w}A_{x,w}+F_{y,n}A_{y,n}-F_{y,s}A_{y,s}\Big)u_P$$
$$+\Big(-\max\big(0,-F_{x,e}A_{x,e}\big)\Big)u_E+\Big(-\max\big(F_{x,w}A_{x,w},0\big)\Big)u_W$$
$$+\Big(-\max\big(0,-F_{y,n}A_{y,n}\big)\Big)u_N+\Big(-\max\big(F_{y,s}A_{y,s},0\big)\Big)u_S=\mathbf{RHS}$$

y-component:

$$\Big(\max\big(0,-F_{x,e}A_{x,e}\big)+\max\big(F_{x,w}A_{x,w},0\big)+\max\big(0,-F_{y,n}A_{y,n}\big)+\max\big(F_{y,s}A_{y,s},0\big)$$
$$+F_{x,e}A_{x,e}-F_{x,w}A_{x,w}+F_{y,n}A_{y,n}-F_{y,s}A_{y,s}\Big)v_P$$
$$+\Big(-\max\big(0,-F_{x,e}A_{x,e}\big)\Big)v_E+\Big(-\max\big(F_{x,w}A_{x,w},0\big)\Big)v_W$$
$$+\Big(-\max\big(0,-F_{y,n}A_{y,n}\big)\Big)v_N+\Big(-\max\big(F_{y,s}A_{y,s},0\big)\Big)v_S=\mathbf{RHS}$$

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

$$\mathbf{LHS}=\quad\mathbf{e}_x\cdot\left(-\nabla p-\frac{\partial\boldsymbol{\sigma}_x}{\partial x}-\frac{\partial\boldsymbol{\sigma}_y}{\partial y}\right)$$

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

$$\mathbf{LHS}=-\frac{\partial p}{\partial x}-\left(-\frac{\partial}{\partial x}\mu\left[2\frac{\partial u}{\partial x}-\frac{2}{3}\left(\cancel{\nabla\cdot\mathbf{u}}\right)\right]\right)-\left(-\frac{\partial}{\partial y}\mu\left[\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right]\right)$$

$$\mathbf{LHS}=-\frac{\partial p}{\partial x}-\left(-\frac{\partial}{\partial x}\mu\left[2\frac{\partial u}{\partial x}\right]\right)-\left(-\frac{\partial}{\partial y}\mu\left[\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right]\right)$$

$$\mathbf{LHS}=-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}\left(2\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial x}\right)$$

$$\mathbf{LHS}=-\frac{\partial p}{\partial x}+\frac{\partial}{\partial x}\left(2\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)+\frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial y}\right)$$

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

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

$$\mathbf{LHS} = -\int_{CV}\frac{\partial p}{\partial x}\,dV + \int_{CV}\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)\,dV + \int_{CV}\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\,dV$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x}\Delta V + \int_{\delta x}\int_{A_x}\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)\,dAdx + \int_{\delta y}\int_{A_y}\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\,dAdy$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x}\Delta V + \int_{\delta x}\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)\,A_x dx + \int_{\delta x}\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)\,A_y dy$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial x}\Delta V + \left(\mu\frac{\partial u}{\partial x}A_x\right)_e - \left(\mu\frac{\partial u}{\partial x}A_x\right)_w + \left(\mu\frac{\partial u}{\partial y}A_y\right)_n - \left(\mu\frac{\partial u}{\partial y}A_y\right)_s$$

$$\mathbf{LHS} = -\left.\frac{\partial p}{\partial x}\right|_{i,J}\Delta V + \mu\left.\frac{\partial u}{\partial x}\right|_e A_{x,e} - \mu\left.\frac{\partial u}{\partial x}\right|_w A_{x,w} + \mu\left.\frac{\partial u}{\partial y}\right|_n A_{y,n} - \mu\left.\frac{\partial u}{\partial y}\right|_s A_{y,s}$$

$$\mathbf{LHS} = -\left.\frac{\partial p}{\partial x}\right|_{i,J}\delta x A_x + \mu\left.\frac{\partial u}{\partial x}\right|_e A_{x,e} - \mu\left.\frac{\partial u}{\partial x}\right|_w A_{x,w} + \mu\left.\frac{\partial u}{\partial y}\right|_n A_{y,n} - \mu\left.\frac{\partial u}{\partial y}\right|_s A_{y,s}$$

The derivative terms above are approximated with the following central differences:

$$\left.\frac{\partial p}{\partial x}\right|_{i,J} = \frac{p_{I,J} - p_{I-1,J}}{\delta x}$$

$$\left.\frac{\partial u}{\partial x}\right|_e = \frac{u_{i+1,J} - u_{i,J}}{\delta x}$$

$$\left.\frac{\partial u}{\partial x}\right|_w = \frac{u_{i,J} - u_{i-1,J}}{\delta x}$$

$$\left.\frac{\partial u}{\partial y}\right|_n = \frac{u_{i,J+1} - u_{i,J}}{\delta y}$$

$$\left.\frac{\partial u}{\partial y}\right|_s = \frac{u_{i,J} - u_{i,J-1}}{\delta y}$$

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:

$$\mathbf{LHS} = -\frac{p_{I,J} - p_{I-1,J}}{\cancel{\delta x}}\cancel{\delta x}A_x + \mu\frac{u_{i+1,J} - u_{i,J}}{\delta x}A_x - \mu\frac{u_{i,J} - u_{i-1,J}}{\delta x}A_x$$
$$+ \mu\frac{u_{i,J+1} - u_{i,J}}{\delta y}A_y - \mu\frac{u_{i,J} - u_{i,J-1}}{\delta y}A_y$$

$$\mathbf{LHS} = -\Big(p_{I,J} - p_{I-1,J}\Big)A_x + \frac{\mu A_x}{\delta x}\Big(u_{i+1,J} - u_{i,J}\Big) - \frac{\mu A_x}{\delta x}\Big(u_{i,J} - u_{i-1,J}\Big)$$
$$+ \frac{\mu A_y}{\delta y}\Big(u_{i,J+1} - u_{i,J}\Big) - \frac{\mu A_y}{\delta y}\Big(u_{i,J} - u_{i,J-1}\Big)$$

$$\mathbf{LHS} = -\Big(p_{I,J} - p_{I-1,J}\Big)A_x + D_xA_x\Big(u_{i+1,J} - u_{i,J}\Big) - D_xA_x\Big(u_{i,J} - u_{i-1,J}\Big)$$
$$+ D_yA_y\Big(u_{i,J+1} - u_{i,J}\Big) - D_yA_y\Big(u_{i,J} - u_{i,J-1}\Big)$$

$$\mathbf{LHS} = D_xA_xu_{i+1,J} - D_xA_xu_{i,J} - D_xA_xu_{i,J} + D_xA_xu_{i-1,J}$$
$$+ D_yA_yu_{i,J+1} - D_yA_yu_{i,J} - D_yA_yu_{i,J} + D_yA_yu_{i,J-1} - \Big(p_{I,J} - p_{I-1,J}\Big)A_x$$

$$\mathbf{LHS} = \Big(- D_xA_x - D_xA_x - D_yA_y - D_yA_y\Big)u_{i,J} + D_xA_xu_{i+1,J} + D_xA_xu_{i-1,J}$$
$$+ D_yA_yu_{i,J+1} + D_yA_yu_{i,J-1} - \Big(p_{I,J} - p_{I-1,J}\Big)A_x$$

y**-component**

$$\mathbf{LHS} = \quad \mathbf{e}_y \cdot \left(- \nabla p - \frac{\partial \boldsymbol{\sigma}_x}{\partial x} - \frac{\partial \boldsymbol{\sigma}_y}{\partial y} \right)$$

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

$$\mathbf{LHS} = -\frac{\partial p}{\partial y} - \left(- \frac{\partial}{\partial x}\mu\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right]\right) - \left(- \frac{\partial}{\partial y}\mu\left[2\frac{\partial v}{\partial y} - \frac{2}{3}\Big(\cancel{\nabla \cdot \mathbf{u}}\Big)\right]\right)$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial y} - \left(- \frac{\partial}{\partial x}\mu\left[\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right]\right) - \left(- \frac{\partial}{\partial y}\mu\left[2\frac{\partial v}{\partial y}\right]\right)$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(2\mu\frac{\partial v}{\partial y}\right)$$

$$\mathbf{LHS} = -\frac{\partial p}{\partial y} + \frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right) + \frac{\partial}{\partial y}\left(2\mu\frac{\partial v}{\partial y}\right)$$

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

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

$$\textbf{LHS} = -\int_{CV} \frac{\partial p}{\partial y}\, dV + \int_{CV} \frac{\partial}{\partial x}\left(\mu \frac{\partial v}{\partial x}\right)\, dV + \int_{CV} \frac{\partial}{\partial y}\left(\mu \frac{\partial v}{\partial y}\right)\, dV$$

$$\textbf{LHS} = -\frac{\partial p}{\partial y}\Delta V + \int_{CV} \frac{\partial}{\partial x}\left(\mu \frac{\partial v}{\partial x}\right)\, dx A_x + \int_{CV} \frac{\partial}{\partial y}\left(\mu \frac{\partial v}{\partial y}\right)\, dy A_y$$

$$\textbf{LHS} = -\frac{\partial p}{\partial y}\Delta V + \left(\mu \frac{\partial v}{\partial x} A_x\right)_e - \left(\mu \frac{\partial v}{\partial x} A_x\right)_w + \left(\mu \frac{\partial v}{\partial y} A_y\right)_n - \left(\mu \frac{\partial v}{\partial y} A_y\right)_s$$

$$\textbf{LHS} = -\left.\frac{\partial p}{\partial y}\right|_{I,j} \Delta V + \mu \left.\frac{\partial v}{\partial x}\right|_e A_{x,e} - \mu \left.\frac{\partial v}{\partial x}\right|_w A_{x,w} + \mu \left.\frac{\partial v}{\partial y}\right|_n A_{y,n} - \mu \left.\frac{\partial v}{\partial y}\right|_s A_{y,s}$$

$$\textbf{LHS} = -\left.\frac{\partial p}{\partial y}\right|_{I,j} \delta y A_y + \mu \left.\frac{\partial v}{\partial x}\right|_e A_{x,e} - \mu \left.\frac{\partial v}{\partial x}\right|_w A_{x,w} + \mu \left.\frac{\partial v}{\partial y}\right|_n A_{y,n} - \mu \left.\frac{\partial v}{\partial y}\right|_s A_{y,s}$$

The derivative terms above are approximated with the following central differences:

$$\left.\frac{\partial p}{\partial y}\right|_{I,j} = \frac{p_{I,J} - p_{I,J-1}}{\delta y}$$

$$\left.\frac{\partial v}{\partial x}\right|_{e} = \frac{v_{I+1,j} - v_{I,j}}{\delta x}$$

$$\left.\frac{\partial v}{\partial x}\right|_{w} = \frac{v_{I,j} - v_{I-1,j}}{\delta x}$$

$$\left.\frac{\partial v}{\partial y}\right|_{n} = \frac{v_{I,j+1} - v_{I,j}}{\delta y}$$

$$\left.\frac{\partial v}{\partial y}\right|_{s} = \frac{v_{I,j} - v_{I,j-1}}{\delta y}$$

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:

$$\textbf{LHS} = -\frac{p_{I,J} - p_{I,J-1}}{\cancel{\delta y}}\cancel{\delta y} A_y + \mu \frac{v_{I+1,j} - v_{I,j}}{\delta x} A_{x,e} - \mu \frac{v_{I,j} - v_{I-1,j}}{\delta x} A_{x,w}$$
$$+ \mu \frac{v_{I,j+1} - v_{I,j}}{\delta y} A_{y,n} - \mu \frac{v_{I,j} - v_{I,j-1}}{\delta y} A_{y,s}$$

$$\textbf{LHS} = -\big(p_{I,J} - p_{I,J-1}\big) A_y + \frac{\mu A_{x,e}}{\delta x}\big(v_{I+1,j} - v_{I,j}\big) - \frac{\mu A_{x,w}}{\delta x}\big(v_{I,j} - v_{I-1,j}\big)$$
$$+ \frac{\mu A_{y,n}}{\delta y}\big(v_{I,j+1} - v_{I,j}\big) - \frac{\mu A_{y,s}}{\delta y}\big(v_{I,j} - v_{I,j-1}\big)$$

$$\textbf{LHS} = -\Big(p_{I,J} - p_{I,J-1}\Big)A_y + D_x A_{x,e}\Big(v_{I+1,j} - v_{I,j}\Big) - D_x A_{x,w}\Big(v_{I,j} - v_{I-1,j}\Big)$$
$$+ D_y A_{y,n}\Big(v_{I,j+1} - v_{I,j}\Big) - D_y A_{y,s}\Big(v_{I,j} - v_{I,j-1}\Big)$$

$$\textbf{LHS} = D_x A_{x,e} v_{I+1,j} - D_x A_{x,e} v_{I,j} - D_x A_{x,w} v_{I,j} + D_x A_{x,w} v_{I-1,j}$$
$$+ D_y A_{y,n} v_{I,j+1} - D_y A_{y,n} v_{I,j} - D_y A_{y,s} v_{I,j} + D_y A_{y,s} v_{I,j-1} - \Big(p_{I,J} - p_{I,J-1}\Big)A_y$$

$$\textbf{LHS} = \Big(- D_x A_{x,e} - D_x A_{x,w} - D_y A_{y,n} - D_y A_{y,s}\Big) v_{I,j} D_x A_{x,e} v_{I+1,j} + D_x A_{x,w} v_{I-1,j}$$
$$+ D_y A_{y,n} v_{I,j+1} + D_y A_{y,s} v_{I,j-1} - \Big(p_{I,J} - p_{I,J-1}\Big)A_y$$

Both Sides Combined

x-component

$$\begin{aligned}\Big(&\max\Big(0, -F_{x,e}A_x\Big) + \max\Big(F_{x,w}A_y, 0\Big) + \max\Big(0, -F_{y,n}A_y\Big) + \max\Big(F_{y,s}A_y, 0\Big)\\ &+ F_{x,e}A_x - F_{x,w}A_y + F_{y,n}A_y - F_{y,s}A_y + D_xA_x + D_xA_y\\ &+ D_yA_y + D_yA_y\Big)u_{i,J} + \Big(- \max\Big(0, -F_{x,e}A_x\Big) - D_xA_x\Big)u_{i+1,J}\\ &+ \Big(- \max\Big(F_{x,w}A_y, 0\Big) - D_xA_y\Big)u_{i-1,J} + \Big(- \max\Big(0, -F_{y,n}A_y\Big) - D_yA_y\Big)u_{i,J+1}\\ &+ \Big(- \max\Big(F_{y,s}A_y, 0\Big) - D_yA_y\Big)u_{i,J-1} = -\Big(p_{I,J} - p_{I-1,J}\Big)A_x\end{aligned}$$

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

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

y-component

$$\Big(\max\Big(0,-F_{x,e}A_x\Big)+\max\Big(F_{x,w}A_y,0\Big)+\max\Big(0,-F_{y,n}A_y\Big)+\max\Big(F_{y,s}A_y,0\Big)$$
$$+F_{x,e}A_x-F_{x,w}A_y+F_{y,n}A_y-F_{y,s}A_y+D_xA_x+D_xA_y$$
$$+D_yA_y+D_yA_y\Big)v_{I,j}+\Big(-\max\Big(0,-F_{x,e}A_x\Big)-D_xA_x\Big)v_{I+1,j}$$
$$+\Big(-\max\Big(F_{x,w}A_y,0\Big)-D_xA_y\Big)v_{I-1,j}+\Big(-\max\Big(0,-F_{y,n}A_y\Big)-D_yA_y\Big)v_{I,j+1}$$
$$+\Big(-\max\Big(F_{y,s}A_y,0\Big)-D_yA_y\Big)v_{I,j-1}=-\Big(p_{I,J}-p_{I,J-1}\Big)A_x$$

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

$$a_{I,j}=-a_{I+1,j}-a_{I-1,j}-a_{I,j+1}-a_{I,j-1}+F_{x,e}A_x-F_{x,w}A_y+F_{y,n}A_y-F_{y,s}A_y$$

$$a_{I+1,j}=-\max\Big(0,-F_{x,e}A_x\Big)-D_xA_x$$

$$a_{I-1,j}=-\max\Big(F_{x,w}A_y,0\Big)-D_xA_y$$

$$a_{I,j+1}=-\max\Big(0,-F_{y,n}A_y\Big)-D_yA_y$$

$$a_{I,j-1}=-\max\Big(F_{y,s}A_y,0\Big)-D_yA_y$$

$$b_{I,j}=-\Big(p_{I,J}-p_{I,J-1}\Big)A_y$$

C.3 SIMPLE-Equations

Velocity Correction Equation

x-component

The Momentum equation for the correct properties:

$$a_{i,J}u_{i,J}+a_{i+1,J}u_{i+1,J}+a_{i-1,J}u_{i-1,J}+a_{i,J+1}u_{i,J+1}+a_{i,J-1}u_{i,J-1}$$
$$=-\Big(p_{I,J}-p_{I-1,J}\Big)A_{x,i,J}+b_{i,J}$$

The Momentum equation for the intermediate / guessed properties:

$$a_{i,J}u^*_{i,J}+a_{i+1,J}u^*_{i+1,J}+a_{i-1,J}u^*_{i-1,J}+a_{i,J+1}u^*_{i,J+1}+a_{i,J-1}u^*_{i,J-1}$$
$$=-\Big(p^*_{I,J}-p^*_{I-1,J}\Big)A_{x,i,J}+b_{i,J}$$

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:

$$\begin{aligned} a_{i,J}(u_{i,J} - u^*_{i,J}) + a_{i+1,J}(u_{i+1,J} - u^*_{i+1,J}) + a_{i-1,J}(u_{i-1,J} - u^*_{i-1,J}) \\ + a_{i,J+1}(u_{i,J+1} - u^*_{i,J+1}) + a_{i,J-1}(u_{i,J-1} - u^*_{i,J-1}) \\ = \Big(- p_{I,J} + p_{I-1,J} + p^*_{I,J} - p^*_{I-1,J}\Big) A_{x,i,J} + \cancel{b_{i,J} - b_{i,J}} \end{aligned}$$

$$\begin{aligned} a^{centre}_{i,J}(u_{i,J} - u^*_{i,J}) + \cancel{a_{i+1,J}u'_{i+1,J}} + \cancel{a_{i-1,J}u'_{i-1,J}} + \cancel{a_{i,J+1}u'_{i,J+1}} + \cancel{a_{i,J-1}u'_{i,J-1}} \\ = -\left(p'_{I,J} - p'_{I-1,J}\right) A_{x,i,J} \end{aligned}$$

The velocity correction equation is then:

$$u_{i,J} = u^*_{i,J} - \frac{A_{x,i,J}}{a^{centre}_{i,J}} \left(p'_{I,J} - p'_{I-1,J}\right)$$

y-component

The Momentum equation for the correct properties:

$$\begin{aligned} a_{I,j}v_{I,j} + a_{I+1,j}v_{I+1,j} + a_{I-1,j}v_{I-1,j} + a_{I,j+1}v_{I,j+1} + a_{I,j-1}v_{I,j-1} \\ = -\Big(p_{I,J} - p_{I,J-1}\Big) A_{y,I,j} + b_{i,J} \end{aligned}$$

The Momentum equation for the intermediate / guessed properties:

$$\begin{aligned} a_{I,j}v^*_{I,j} + a_{I+1,j}v^*_{I+1,j} + a_{I-1,j}v^*_{I-1,j} + a_{I,j+1}v^*_{I,j+1} + a_{I,j-1}v^*_{I,j-1} \\ = -\Big(p^*_{I,J} - p^*_{I,J-1}\Big) A_{y,I,j} + b_{i,J} \end{aligned}$$

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:

$$\begin{aligned} a_{I,j}(v_{I,j} - v^*_{I,j}) + a_{I+1,j}(v_{I+1,j} - v^*_{I+1,j}) + a_{I-1,j}(v_{I-1,j} - v^*_{I-1,j}) \\ + a_{I,j+1}(v_{I,j+1} - v^*_{I,j+1}) + a_{I,j-1}(v_{I,j-1} - v^*_{I,j-1}) \\ = \Big(- p_{I,J} + p_{I,J-1} + p^*_{I,J} - p^*_{I,J-1}\Big) A_{y,I,j} + \cancel{b_{I,j} - b_{I,j}} \end{aligned}$$

$$a^{centre}_{I,j}(v_{I,j} - v^*_{I,j}) + \cancel{a_{I+1,j}v'_{I+1,j}} + \cancel{a_{I-1,j}v'_{I-1,j}} + \cancel{a_{I,j+1}v'_{I,j+1}} + \cancel{a_{I,j-1}v'_{I,j-1}} = -\left(p'_{I,J} - p'_{I,J-1}\right) A_{y,I,j}$$

The velocity correction equation is then:

$$v_{I,j} = v^*_{I,j} - \frac{A_{y,I,j}}{a^{centre}_{I,j}} \left(p'_{I,J} - p'_{I,J-1}\right)$$

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:

$$\rho A_{x,i+1,J}\left(u^*_{i+1,J} - \frac{A_{x,i+1,J}}{a^{centre}_{i+1,J}}\left(p'_{I+1,J} - p'_{I,J}\right)\right) - \rho A_{x,i,J}\left(u^*_{i,J} - \frac{A_{x,i,J}}{a^{centre}_{i,J}}\left(p'_{I,J} - p'_{I-1,J}\right)\right)$$
$$+\rho A_{y,I,j+1}\left(v^*_{I,j+1} - \frac{A_{y,I,j+1}}{a^{centre}_{I,j+1}}\left(p'_{I,J+1} - p'_{I,J}\right)\right) - \rho A_{y,I,j}\left(v^*_{I,j} - \frac{A_{y,I,j}}{a^{centre}_{I,j}}\left(p'_{I,J} - p'_{I,J-1}\right)\right) = 0$$

$$\rho A_{x,i+1,J} u^*_{i+1,J} - \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}}\left(p'_{I+1,J} - p'_{I,J}\right) - \rho A_{x,i,J} u^*_{i,J} + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}}\left(p'_{I,J} - p'_{I-1,J}\right)$$
$$+ \rho A_{y,I,j+1} v^*_{I,j+1} - \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}}\left(p'_{I,J+1} - p'_{I,J}\right) - \rho A_{y,I,j} v^*_{I,j} + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}}\left(p'_{I,J} - p'_{I,J-1}\right) = 0$$

$$- \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}}\left(p'_{I+1,J} - p'_{I,J}\right) + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}}\left(p'_{I,J} - p'_{I-1,J}\right)$$
$$- \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}}\left(p'_{I,J+1} - p'_{I,J}\right) + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}}\left(p'_{I,J} - p'_{I,J-1}\right)$$
$$= -\rho A_{x,i+1,J} u^*_{i+1,J} + \rho A_{x,i,J} u^*_{i,J} - \rho A_{y,I,j+1} v^*_{I,j+1} + \rho A_{y,I,j} v^*_{I,j}$$

$$- \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} p'_{I+1,J} + \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} p'_{I,J} + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} p'_{I,J} - \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} p'_{I-1,J}$$
$$- \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} p'_{I,J+1} + \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} p'_{I,J} + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} p'_{I,J} - \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} p'_{I,J-1}$$
$$= -\rho A_{x,i+1,J} u^*_{i+1,J} + \rho A_{x,i,J} u^*_{i,J} - \rho A_{y,I,j+1} v^*_{I,j+1} + \rho A_{y,I,j} v^*_{I,j}$$

$$\left(\frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} + \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}}\right) p'_{I,J}$$
$$- \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} p'_{I+1,J} - \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} p'_{I-1,J} - \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} p'_{I,J+1} - \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} p'_{I,J-1}$$
$$= -\rho A_{x,i+1,J} u^*_{i+1,J} + \rho A_{x,i,J} u^*_{i,J} - \rho A_{y,I,j+1} v^*_{I,j+1} + \rho A_{y,I,j} v^*_{I,j}$$

$$\left(\frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} + \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}}\right) p'_{I,J}$$
$$- \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} p'_{I+1,J} - \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} p'_{I-1,J} - \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} p'_{I,J+1} - \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} p'_{I,J-1}$$
$$= -A_{x,i+1,J} F^*_{i+1,J} + A_{x,i,J} F^*_{i,J} - A_{y,I,j+1} F^*_{I,j+1} + A_{y,I,j} F^*_{I,j}$$

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

$$\begin{aligned}
\nu_{I,J} &= \frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} + \frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} + \frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} + \frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} \\
\nu_{I+1,J} &= -\frac{\rho A^2_{x,i+1,J}}{a^{centre}_{i+1,J}} \\
\nu_{I-1,J} &= -\frac{\rho A^2_{x,i,J}}{a^{centre}_{i,J}} \\
\nu_{I,J+1} &= -\frac{\rho A^2_{y,I,j+1}}{a^{centre}_{I,j+1}} \\
\nu_{I,J-1} &= -\frac{\rho A^2_{y,I,j}}{a^{centre}_{I,j}} \\
\beta_{I,J} &= -A_{x,i+1,J}F^*_{i+1,J} + A_{x,i,J}F^*_{i,J} - A_{y,I,j+1}F^*_{I,j+1} + A_{y,I,j}F^*_{I,j}
\end{aligned}$$

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).

$$\mathbf{r} = x\mathbf{e}_x + y\mathbf{e}_y + z\mathbf{e}_z \tag{D.1.1}$$

$$\sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3}\left(g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial x}{\partial q^j}\right) + \nabla^2 q^j \frac{\partial x}{\partial q^j}\right) = 0 \tag{D.1.2}$$

$$\sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3}\left(g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial y}{\partial q^j}\right) + \nabla^2 q^j \frac{\partial y}{\partial q^j}\right) = 0 \tag{D.1.3}$$

$$\sum_{i=1}^{3}\sum_{j=1}^{3}\sum_{k=1}^{3}\left(g^{ij}\frac{\partial}{\partial q^i}\left(\frac{\partial z}{\partial q^j}\right) + \nabla^2 q^j \frac{\partial z}{\partial q^j}\right) = 0 \tag{D.1.4}$$

Taking the sums yields equations (D.1.5) (D.1.6) and (D.1.7) for the x-, y- and z components respectively.

$$\begin{aligned} g^{11}\frac{\partial}{\partial q^1}\left(\frac{\partial x}{\partial q^1}\right) + g^{12}\frac{\partial}{\partial q^1}\left(\frac{\partial x}{\partial q^2}\right) + g^{13}\frac{\partial}{\partial q^1}\left(\frac{\partial x}{\partial q^3}\right) \\ + g^{21}\frac{\partial}{\partial q^2}\left(\frac{\partial x}{\partial q^1}\right) + g^{22}\frac{\partial}{\partial q^2}\left(\frac{\partial x}{\partial q^2}\right) + g^{23}\frac{\partial}{\partial q^2}\left(\frac{\partial x}{\partial q^3}\right) \\ + g^{31}\frac{\partial}{\partial q^3}\left(\frac{\partial x}{\partial q^1}\right) + g^{32}\frac{\partial}{\partial q^3}\left(\frac{\partial x}{\partial q^2}\right) + g^{33}\frac{\partial}{\partial q^3}\left(\frac{\partial x}{\partial q^3}\right) \\ + \nabla^2 q^1 \frac{\partial x}{\partial q^1} + \nabla^2 q^2 \frac{\partial x}{\partial q^2} + \nabla^2 q^3 \frac{\partial x}{\partial q^3} = 0 \end{aligned} \tag{D.1.5}$$

$$\begin{aligned} g^{11}\frac{\partial}{\partial q^1}\left(\frac{\partial y}{\partial q^1}\right) + g^{12}\frac{\partial}{\partial q^1}\left(\frac{\partial y}{\partial q^2}\right) + g^{13}\frac{\partial}{\partial q^1}\left(\frac{\partial y}{\partial q^3}\right) \\ + g^{21}\frac{\partial}{\partial q^2}\left(\frac{\partial y}{\partial q^1}\right) + g^{22}\frac{\partial}{\partial q^2}\left(\frac{\partial y}{\partial q^2}\right) + g^{23}\frac{\partial}{\partial q^2}\left(\frac{\partial y}{\partial q^3}\right) \\ + g^{31}\frac{\partial}{\partial q^3}\left(\frac{\partial y}{\partial q^1}\right) + g^{32}\frac{\partial}{\partial q^3}\left(\frac{\partial y}{\partial q^2}\right) + g^{33}\frac{\partial}{\partial q^3}\left(\frac{\partial y}{\partial q^3}\right) \\ + \nabla^2 q^1\frac{\partial y}{\partial q^1} + \nabla^2 q^2\frac{\partial y}{\partial q^2} + \nabla^2 q^3\frac{\partial y}{\partial q^3} = 0 \end{aligned} \quad \text{(D.1.6)}$$

$$\begin{aligned} g^{11}\frac{\partial}{\partial q^1}\left(\frac{\partial z}{\partial q^1}\right) + g^{12}\frac{\partial}{\partial q^1}\left(\frac{\partial z}{\partial q^2}\right) + g^{13}\frac{\partial}{\partial q^1}\left(\frac{\partial z}{\partial q^3}\right) \\ + g^{21}\frac{\partial}{\partial q^2}\left(\frac{\partial z}{\partial q^1}\right) + g^{22}\frac{\partial}{\partial q^2}\left(\frac{\partial z}{\partial q^2}\right) + g^{23}\frac{\partial}{\partial q^2}\left(\frac{\partial z}{\partial q^3}\right) \\ + g^{31}\frac{\partial}{\partial q^3}\left(\frac{\partial z}{\partial q^1}\right) + g^{32}\frac{\partial}{\partial q^3}\left(\frac{\partial z}{\partial q^2}\right) + g^{33}\frac{\partial}{\partial q^3}\left(\frac{\partial z}{\partial q^3}\right) \\ + \nabla^2 q^1\frac{\partial z}{\partial q^1} + \nabla^2 q^2\frac{\partial z}{\partial q^2} + \nabla^2 q^3\frac{\partial z}{\partial q^3} = 0 \end{aligned} \quad \text{(D.1.7)}$$

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:

$$J = \det\left(J^i_j\right) \tag{D.2.10}$$

$$= \begin{vmatrix} \frac{\partial x^1}{\partial q^1} & \frac{\partial x^1}{\partial q^2} & \frac{\partial x^1}{\partial q^3} \\ \frac{\partial x^2}{\partial q^1} & \frac{\partial x^2}{\partial q^2} & \frac{\partial x^2}{\partial q^3} \\ \frac{\partial x^3}{\partial q^1} & \frac{\partial x^3}{\partial q^2} & \frac{\partial x^3}{\partial q^3} \end{vmatrix} \tag{D.2.11}$$

$$= \frac{\partial x^1}{\partial q^1}\begin{vmatrix} \frac{\partial x^2}{\partial q^2} & \frac{\partial x^2}{\partial q^3} \\ \frac{\partial x^3}{\partial q^2} & \frac{\partial x^3}{\partial q^3} \end{vmatrix} - \frac{\partial x^1}{\partial q^2}\begin{vmatrix} \frac{\partial x^2}{\partial q^1} & \frac{\partial x^2}{\partial q^3} \\ \frac{\partial x^3}{\partial q^1} & \frac{\partial x^3}{\partial q^3} \end{vmatrix} + \frac{\partial x^1}{\partial q^3}\begin{vmatrix} \frac{\partial x^2}{\partial q^1} & \frac{\partial x^2}{\partial q^2} \\ \frac{\partial x^3}{\partial q^1} & \frac{\partial x^3}{\partial q^2} \end{vmatrix} \tag{D.2.12}$$

$$\begin{aligned} &= \frac{\partial x^1}{\partial q^1}\left(\frac{\partial x^2}{\partial q^2}\frac{\partial x^3}{\partial q^3} - \frac{\partial x^3}{\partial q^2}\frac{\partial x^2}{\partial q^3}\right) - \frac{\partial x^1}{\partial q^2}\left(\frac{\partial x^2}{\partial q^1}\frac{\partial x^3}{\partial q^3} - \frac{\partial x^3}{\partial q^1}\frac{\partial x^2}{\partial q^3}\right) \\ &\quad + \frac{\partial x^1}{\partial q^3}\left(\frac{\partial x^2}{\partial q^1}\frac{\partial x^3}{\partial q^2} - \frac{\partial x^3}{\partial q^1}\frac{\partial x^2}{\partial q^2}\right) \end{aligned} \tag{D.2.13}$$

$$\begin{aligned} &= \frac{\partial x^1}{\partial q^1}\frac{\partial x^2}{\partial q^2}\frac{\partial x^3}{\partial q^3} - \frac{\partial x^1}{\partial q^1}\frac{\partial x^3}{\partial q^2}\frac{\partial x^2}{\partial q^3} - \frac{\partial x^1}{\partial q^2}\frac{\partial x^2}{\partial q^1}\frac{\partial x^3}{\partial q^3} + \frac{\partial x^1}{\partial q^2}\frac{\partial x^3}{\partial q^1}\frac{\partial x^2}{\partial q^3} \\ &\quad + \frac{\partial x^1}{\partial q^3}\frac{\partial x^2}{\partial q^1}\frac{\partial x^3}{\partial q^2} - \frac{\partial x^1}{\partial q^3}\frac{\partial x^3}{\partial q^1}\frac{\partial x^2}{\partial q^2} \end{aligned} \tag{D.2.14}$$

Contravariant tensor components summed over k:

$$g^{ij} = \frac{\mathbf{A}^i \cdot \mathbf{A}^j}{J^2} = \frac{A^i_k \mathbf{e}_k \cdot A^j_l \mathbf{e}_L}{j^2} = \frac{A^i_k A^j_l \delta_{kl}}{J^2} = \frac{A^i_k A^j_k}{J^2} \tag{D.2.15}$$

Components of g^{ij}:

$$g^{11} = \frac{A^1_k A^1_k}{J^2} = \frac{A^1_1 A^1_1 + A^1_2 A^1_2 + A^1_3 A^1_3}{J^2} \tag{D.2.16}$$

$$g^{21} = \frac{A^2_k A^1_k}{J^2} = \frac{A^2_1 A^1_1 + A^2_2 A^1_2 + A^2_3 A^1_3}{J^2} \tag{D.2.17}$$

$$g^{31} = \frac{A^3_k A^1_k}{J^2} = \frac{A^3_1 A^1_1 + A^3_2 A^1_2 + A^3_3 A^1_3}{J^2} \tag{D.2.18}$$

$$g^{12} = \frac{A^1_k A^2_k}{J^2} = \frac{A^1_1 A^2_1 + A^1_2 A^2_2 + A^1_3 A^2_3}{J^2} \tag{D.2.19}$$

$$g^{22} = \frac{A^2_k A^2_k}{J^2} = \frac{A^2_1 A^2_1 + A^2_2 A^2_2 + A^2_3 A^2_3}{J^2} \tag{D.2.20}$$

$$g^{32} = \frac{A^3_k A^2_k}{J^2} = \frac{A^3_1 A^2_1 + A^3_2 A^2_2 + A^3_3 A^2_3}{J^2} \tag{D.2.21}$$

$$g^{13} = \frac{A^1_k A^3_k}{J^2} = \frac{A^1_1 A^3_1 + A^1_2 A^3_2 + A^1_3 A^3_3}{J^2} \tag{D.2.22}$$

$$g^{23} = \frac{A^2_k A^3_k}{J^2} = \frac{A^2_1 A^3_1 + A^2_2 A^3_2 + A^2_3 A^3_3}{J^2} \tag{D.2.23}$$

$$g^{33} = \frac{A^3_k A^3_k}{J^2} = \frac{A^3_1 A^3_1 + A^3_2 A^3_2 + A^3_3 A^3_3}{J^2} \tag{D.2.24}$$

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			✓	✓	
`ntest`	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	$\hat{p}^{\circ}$		✓	✓	
`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	$\tilde{\hat{p}}_{out}$		✓	✓	
`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	$\hat{u}^{\circ}$		✓	✓	
`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	$\hat{v}^{\circ}$		✓	✓	
`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.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            One dimensional fluid flow in a straight channel              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc
clear
close all
tic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Solver specifications
maxits = 20000;
N = 100;                                      % Number of scalar nodal points
runiterationwise = 0;               % Plots the profiles after each iteration
plotinitialprofiles = 0;                           % Plot the initial guesses

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initial guesses

p_out = 1e5;          % Pa
u_in = 1e-3;          % m/s
p_guess = 1.5e5;      % Pa
u_guess = 1.5e-3;     % m/s

% Creating a linear profile for the initial guess of the pressure
pprofile = [linspace(p_guess,p_out,N+1), p_out];
p_circ = pprofile(2:end-1);

% Creating a linear profile for the initial guess of the velocity
uprofile = linspace(u_in,u_guess,N+1);
% Placing the velocities in the staggered grid
u_circ = 0.5*(uprofile(2:end)+uprofile(1:end-1));

alpha_u = 1;          % Under-relaxation of the velocity
alpha_p = 0.05;       % Under-relaxation of the pressure
% 1 corresponds to no under-relaxation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Parameters and system specifications
L = 3;                % Channel length [m]

x_0 = 0;              % Channel start [m]
x_N = L;              % Channel end [m]
mu = 8.90 * 10^-4;    % Viscosity [Pa s]

sys_width = 1;        % Height of the channel is set to unity
                      % for the one dimensional model

del_x = x_N/N;        % Width of the control volume [m]
A = del_x*sys_width;% Cross-sectional area [m^2]


rho = 1e3;            % Density [kg/m^3]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialisation

p_corr = zeros(1, N);
p_new = zeros(1, N);

u_star = zeros(1, N);
u_corr = zeros(1, N);
u_new = zeros(1, N);

F_e = zeros(1, N);
F_w = zeros(1, N);
D = mu/del_x;

U = zeros(N, N);
bu = zeros(1, N);
```

```

T = zeros(N, N);
beta = zeros(1, N);
a_u = zeros(1, N);

xu_plot = linspace(x_0, x_N, N+1); % staggered grid
xp_plot = linspace(x_0+del_x/2, x_N+del_x/2, N+1);

if plotinitialprofiles == 1
    figure
    plot(xu_plot, [u_in, u_circ])
    hold on
    plot(xu_plot(1:2), [u_in, u_circ(1)],'r')
    title('Initial guess $u$', 'interpreter', 'latex')
    figure
    plot(xp_plot, [p_circ,p_out])
    hold on
    plot(xp_plot(end-1:end), [p_circ(end),p_out],'r')
    title('Initial guess $p$', 'interpreter', 'latex')
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% While loop
conv = 0;
it = 1;
while conv == 0

    %% Solve momentum equation
    % Calculation of coefficients F_e:
    for i = 1:length(u_circ)-1
        F_e(i) = rho*1/2*(u_circ(i+1)+u_circ(i));
    end %for
    F_e(end) = rho*1/2*(u_circ(end)+u_circ(end-1)); % F_e = F_w

    % Calculation of coefficients F_w:
    F_w(1) = rho*1/2*(u_circ(1)+u_in);
    for i = 2:length(u_circ)
        F_w(i) = rho*1/2*(u_circ(i)+u_circ(i-1));
    end %for

    % u_2 (u(1))
    U(1,1) = 4*D*A +  max(0, -F_e(1)*A) + max(F_w(1)*A,0)...
        + F_e(1)*A - F_w(1)*A;
    U(1,2) = -2*D*A - max(0, -F_e(1)*A);
    bu(1) = ( 2*D*A +  max(F_w(1)*A, 0))*u_in ...
        - A*(p_circ(2) - p_circ(1));


    % u_centers
    for j = 2:length(u_circ)-1
        U(j,j) =  4*D*A +  max(0, -F_e(j)*A) + max(F_w(j)*A, 0) ...
            + F_e(j)*A - F_w(j)*A ;
        U(j,j+1) = -2*D*A- max(0, -F_e(j)*A);
        U(j,j-1) =  -2*D*A- max(F_w(j)*A, 0);
        bu(j) =  - (p_circ(j+1) - p_circ(j))*A;


    end %for

    % u_n+1 (u(end))
    U(end,end) =  4*D*A + max(F_w(end)*A, 0) ...
        + F_e(end)*A - F_w(end)*A ;
    U(end,end-1) = -2*D*A -max(F_w(end)*A, 0);
    bu(end) = - (p_out - p_circ(end))*A;

    % Matrix inversion
    u_star = U\bu';

    %% Solve pressure correction equation
    % a^center-coefficients in the momentum equation
    for i = 1:length(U)
        a_u(i) = U(i,i);
    end %for

    % p_1
    T(1,1) = rho*A/a_u(1);
```

```
    T(1,2) = - rho*A/a_u(1);
    beta(1) =  rho*(-u_star(1) + u_in);

    % p_centers
    for j = 2:length(p_corr)-1
        T(j,j) = rho*A*(1/a_u(j) + 1/a_u(j-1));
        T(j,j+1) = - rho*A/a_u(j);
        T(j,j-1) = - rho*A/a_u(j-1);
        beta(j) =  rho*(-u_star(j) + u_star(j-1));
    end %for

    % p_N
    T(end,end) = rho*A*(1/a_u(end)+1/a_u(end-1));
    T(end,end-1) =  -  rho*A/a_u(end);
    beta(end) = rho*(-u_star(end) + u_star(end-1));

    % Matrix inversion
    p_corr = T\beta';

    %% Velocity correction
    for j = 1:length(p_corr)-1
        u_corr(j) = - A/a_u(j)*(p_corr(j+1)-p_corr(j));
    end %for
    u_corr(end) = - A/a_u(end)*(-p_corr(end));
    % pressure correction is zero for the known outlet pressure

    %% Under-relaxation
    % Pressure
    p_new = p_circ + alpha_p* p_corr';

    % Under-relaxation of u
    u_new = alpha_u*(u_star' + u_corr) + (1-alpha_u)*u_circ;

    %% Check convergence
    if isnan(rcond(U)) || isnan(rcond(T))
        fprintf('Stopped due to singularity in matrix\n')
        fprintf('RCOND velocity: %e \nRCOND pressure: %e\n',...
            rcond(U), rcond(T))
        fprintf('Problem occured after %d iterations\n', it-1)
        return
    end %if

    c1 = 1/u_in*sqrt((U*u_star-bu')'*(U*u_star-bu')); % coefficient summed
    c3 = abs(sum(beta)); % continuity
    c4 =  1/u_in*max(abs(u_circ - u_star')); % change from last iteration

    c1_lim = 10^-6;
    c3_lim = 10^-6;
    c4_lim = 10^-6;

    c1_diff = c1-c1_lim;
    c3_diff = c3-c3_lim;
    c4_diff = c4-c4_lim;

    if  (c1 < c1_lim) && (c3 < c3_lim) && (c4 < c4_lim) || (it == maxits)
        conv = 1; % While loop is stopped
        if (it == maxits)
            fprintf('Stopped at max iterations (%d)\n',it);
        else
            fprintf('Solution converged after %d iterations\n',it);
        end %if

        fprintf('c1\tMomentum residual\t\t%.2e\tLimit: %.2e\n',c1,c1_lim);
        fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',c3,c3_lim);
        fprintf('c4\tDiff. last iteration\t%.2e\tLimit: %.2e\n',c4,c4_lim);

        if max([c1_diff c3_diff c4_diff])== c1_diff
            fprintf('Limiting criteria is c1\tMomentum residual\n')
        elseif max([c1_diff c3_diff c4_diff])== c3_diff
            fprintf('Limiting criteria is c3\tPressure correction\n')
        elseif max([c1_diff c3_diff c4_diff])== c4_diff
            fprintf('Limiting criteria is c4\tDiff. last iteration\n')
        end %if


    else
```

```
        u_circ = u_new; % Not converged, updated variables.
        p_circ = p_new;
        it = it + 1;


    end %if
    if runiterationwise == 1 || conv == 1
        % For iterationwise plotting and for when the model is stopped
        % Plot after each iteration and close before proceding to the next

        u_new_plot = [u_in u_new];

        % Discretied x-node points
        p_plot = [p_new p_out];
        p_corr_plot = [p_corr' 0];

        fu = figure;
        plot(xu_plot, u_new_plot)
        s = sprintf('Plot of $u^{new}$ after %d iterations', it-1 );
%         f = title(s);
%         set(f, 'interpreter', 'latex', 'fontsize', 16)
        set(gca,'TickLabelInterpreter','latex')
        xlabel('$x$-direction [m]', 'interpreter', 'latex')
        xlim([0,3])
        ylabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
%         set(fu, 'Position', [5,217,414.6667,420]);
        %[left bottom width height]
        saveas(gcf,'unew1D.png')


        fp = figure;
        plot(xp_plot, p_plot)
        s = sprintf('Plot of $p^{new}$ after %d iterations', it-1 );
%         f = title(s);
%         set(f, 'interpreter', 'latex', 'fontsize', 16)
        set(gca,'TickLabelInterpreter','latex')
        xlabel('$x$-direction [m]', 'interpreter', 'latex')
        xlim([0,3])
        ylabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
%         set(fp, 'Position', [419.6667,217,434.6667,420]);
        % [left bottom width height]
        saveas(gcf,'pnew1D.png')

        fpcorr = figure;
        plot(xp_plot, p_corr_plot)
        s = sprintf('Plot of $p^{corr}$ after %d iterations', it-1 );
%         f = title(s);
%         set(f, 'interpreter', 'latex', 'fontsize', 16)
        set(gca,'TickLabelInterpreter','latex')
        xlabel('$x$-direction [m]', 'interpreter', 'latex')
        xlim([0,3])
        ylabel('Pressure correction $p$, [Pa]', 'interpreter', 'latex')
%         set(fpcorr, 'Position', [855,217.6667,424,422.6667]);
        % [left bottom width height]
        saveas(gcf,'pcorr1D.png')

        if conv ~= 1 % if not converged
           pause
           close all
        end %if
    end % if

end %while

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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Two dimensional fluid flow in a straight channel, dimensionless     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear
clc
close all
tic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Solver specifications
maxits = 100000;  % Maximum number of iterations, stop if iterations exceed
N = 88;                      % Number of scalar nodal points in x-direction
M = 18;                      % Number of scalar nodal points in y-direction
runiterationwise = 0;             % Plots the profiles after each iteration
plotinitialprofiles = 0;                        % Plot the initial guesses
solvvel = true;                                     % Solve for v-velocity
contplots = false;          % Show plots of continuity + cont_x and cont_y
v_out_zero = false;               % Use v_out = zero as boundary condition

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System specifications
m = M - 1;                      % Number of y-velocity nodes in y-direction
L = 22;                                                   % Channel length
h = 1;                                                    % Channel height
D_hyd = 4*h*1/(1+1+h+h);           % Hydraulic diameter for Reynolds number

x_0 = 0;                               % Defining the domain using x and y
x_N = L;
y_0 = 0;
y_M = h;

mu_true = 8.90 * 10^-4;                               % Viscosity of water

del_z_true = 1;                                             % System depth
del_x_true = x_N/N;                                % Control volume width
del_y_true = y_M/M;                               % Control volume height
A_x_true = del_y_true*del_z_true;      % Cross-sectional area in x-direction
A_y_true = del_x_true*del_z_true;      % Cross-sectional area in y-direction

rho_true = 997;                                         % Density of water
u_in_true = 0.0005;                                     % Inlet u-velocity
g_x = 0;                                                   % No gravitation
g_y = 0;                                                   % No gravitation


Re = rho_true*D_hyd*u_in_true/mu_true;                    % Reynolds number

p_atm = 101325;                       % Atmospheric presssure at outlet
p_out_tilde = 0;                                       % Adjusted pressure
p_out = ones(1,M)*p_out_tilde;                    % Outlet pressure profile

alpha_u = 0.01;                             % Under-relaxation factor for u
alpha_v = 0.01;                             % Under-relaxation factor for v
alpha_p = 0.02;                             % Under-relaxation factor for p

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Dimensionless parameters
mu = 1;                                           % Dimensionless viscosity
rho = 1;                                            % Dimensionless density
del_x = del_x_true/D_hyd;            % Dimensionless control volume width
del_y = del_y_true/D_hyd;           % Dimensionless control volume height
A_x = A_x_true/D_hyd^2; % Dimensionless cross-sectional area in x-direction
A_y = A_y_true/D_hyd^2; % Dimensionless cross-sectional area in y-direction
```

```
D_x = 1/Re*mu/del_x;   % Dimensionless diffusion conductance in x-direction
D_y = 1/Re*mu/del_y;   % Dimensionless diffusion conductance in y-direction
u_in = 1;                                                  % Inlet u-velocity
v_in = 0;                                                  % Inlet u-velocity
u_guess = 1.0; %                              % Initial guess for u-velocity
v_guess = 0.0; %                              % Initial guess for v-velocity
u_circ = ones(1,M*N)*u_guess;          % Initial guess vector for u-velocity
v_circ = ones(1,m*N)*v_guess;          % Initial guess vector for v-velocity
p_guess = 0/(rho_true*u_in_true^2);             % Initial guess for pressure
p_circ_vector = linspace(p_guess,p_out_tilde,N)';      % Linear profile from
                                              % guess to known outlet pressure
p_circ = zeros(M*N,1);
for j = 1:M    % Filling in initial pressure vector with the linear profile
    p_circ((j-1)*N+1:j*N) = p_circ_vector;
end %for

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialization of solution vectors
p_corr = zeros(1, M*N);                                 % Pressure correction
p_new = zeros(1, M*N);                                         % New pressure

u_star = zeros(1, M*N);                  % u-velocity after matrix inversion
u_corr = zeros(1, M*N);                              % u-velocity correction
u_new = zeros(1, M*N);                                      % New u-velocity
U = zeros(M*N, M*N);                         % u-velocity coefficient matrix
bu = zeros(1, M*N);                          % u-velocity source term vector

v_star = zeros(1, m*N);                  % v-velocity after matrix inversion
v_corr = zeros(1, m*N);                              % v-velocity correction
v_new = zeros(1, m*N);                                      % New v-velocity
V = zeros(m*N, m*N);                         % v-velocity coefficient matrix
bv = zeros(1, m*N);                          % v-velocity source term vector

F_xe = zeros(1, M*N);                     % Convective mass flux per unit area
F_xw = zeros(1, M*N);
F_xn = zeros(1, M*N);
F_xs = zeros(1, M*N);
F_ye = zeros(1, m*N);
F_yw = zeros(1, m*N);
F_yn = zeros(1, m*N);
F_ys = zeros(1, m*N);

T = zeros(M*N, M*N);                   % Pressure correction coefficient matrix
                                          % for pressure
beta = zeros(1, M*N);                  % Pressure correction source term vector

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plots of initial guesses
if plotinitialprofiles == true
    f111 = figure;
    surf(linspace(x_0+del_x/2, x_N+del_x/2, N+1),...
             linspace(y_0+del_y/2, y_M-del_y/2, M),...
             [global2matrix(p_circ,N,M) p_out']);             % surf(x,y,z)
    s = sprintf('Initial guess $p_{circ}$');
    f = title(s);
    set(f, 'interpreter', 'latex', 'fontsize', 16)
    set(gca,'TickLabelInterpreter','latex')
    xlabel('$x$-direction [m]', 'interpreter', 'latex')
    ylabel('$y$-direction [m]', 'interpreter', 'latex')
    zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')

    u_circ_carthesian = ones(M,N+1)*u_guess;
    u_circ_carthesian(:,1) = u_in;

    f122 = figure;
    surf(linspace(x_0, x_N, N+1),...                                 % surf(x,y,z)
        linspace(y_0+del_y/2, y_M-del_y/2, M),u_circ_carthesian);
    s = sprintf('Initial guess $u_{circ}$');
    f = title(s);
    set(f, 'interpreter', 'latex', 'fontsize', 16)
    set(gca,'TickLabelInterpreter','latex')
    xlabel('$x$-direction [m]', 'interpreter', 'latex')
    ylabel('$y$-direction [m]', 'interpreter', 'latex')
    zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')

    v_circ_carthesian = ones(M+1,N+1)*v_guess;
```

```
    v_circ_carthesian(1,:) = 0;
    v_circ_carthesian(M+1,:) = 0;
    v_circ_carthesian(:,1) = 0;

    f133 = figure;
    surf(linspace(-x_0-del_x/2, x_N+del_x/2, N+1),...
        linspace(y_0, y_M, M+1),v_circ_carthesian);           % surf(x,y,z)
    % set(f,'edgecolor','none')
    s = sprintf('Initial guess $v_{circ}$');
    f = title(s);
    set(f, 'interpreter', 'latex', 'fontsize', 16)
    set(gca,'TickLabelInterpreter','latex')
    xlabel('$x$-direction [m]', 'interpreter', 'latex')
    ylabel('$y$-direction [m]', 'interpreter', 'latex')
    zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
pause
close all
end % if

% Defining x and y points for the staggered grid for plotting
xu_plot = linspace(x_0, x_N, N+1);
yu_plot = [0,linspace(y_0+del_y/2, y_M-del_y/2, M),h];
xv_plot = [0,linspace(x_0+del_x/2, x_N-del_x/2, N)];
yv_plot = linspace(y_0, y_M, M+1);
xp_plot = linspace(x_0+del_x/2, x_N+del_x/2, N+1) + del_x_true/2;
yp_plot = linspace(y_0+del_y/2, y_M-del_y/2, M) + del_y_true/2;

%% Specifications before iteration
if solvvel == false  % Not solve for v-velocity
    v_out_zero = false; % Turn off outlet boundary condition for v
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% While loop
conv = 0;                             % 0 is not converged, 1 when converged
it = 1;                                                % The current iteration

% Coefficients in matrix, example:
% sP_coeff is part of the a_P-coefficient at the diagonal position in the
% matrix, while S_coeff is the coefficient in the matrix for the south node
while  conv == 0 % it<=maxits %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Generation of F
    %Generation of F_x:
    for i = 1:M*N

        etest = mod(i, N) == 0;
        wtest = mod(i-1, N) == 0;
        ntest = M*N - (N - 1) <= i && i <= N*M  ;
        stest = 1 <= i && i <= N  ;

        % Northeastern corner
        if etest == true && ntest == true
            F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));   % F_xe = F_xw
            F_xn(i) = 0;                               % v_NorthWall = 0;

            F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
            F_xs(i) = rho/2*v_circ(i-N);

        % Southeastern corner
        elseif etest == true && stest == true
            F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));   % F_xe = F_xw
            F_xs(i) = 0;                               % v_SouthWall = 0;

            F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
            F_xn(i) = rho/2*v_circ(i);

        % Northwestern corner
        elseif wtest == true && ntest == true
            F_xw(i) = rho/2*(u_in+u_circ(i));          % Inlet
            F_xn(i) = 0;                               % v_NorthWall = 0;

            F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
            F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));

        % Southwestern corner
```

```
        elseif wtest == true && stest == true
            F_xw(i) = rho/2*(u_in+u_circ(i));           % Inlet
            F_xs(i) = 0;                                % v_SouthWall = 0;

            F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
            F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

        % At eastern boundary (x = L)
        elseif etest == true && ntest == false && stest == false
            F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));    % F_xe = F_xw

            F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
            F_xn(i) = rho/2*v_circ(i);
            F_xs(i) = rho/2*v_circ(i-N);

        % At western boundary (x = 0)
        elseif wtest == true && ntest == false && stest == false
            F_xw(i) = rho/2*(u_in+u_circ(i));           % Inlet

            F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
            F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
            F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));

        % At northern boundary (y = h)
        elseif ntest == true && etest == false && wtest == false
           F_xn(i) = 0;                                 % v_NorthWall = 0;

           F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
           F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
           F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));

        % At southern boundary (y = 0)
        elseif stest == true && etest == false && wtest == false
            F_xs(i) = 0;                                % v_SouthWall = 0;

            F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
            F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
            F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

        %Not at any boundary
        else
            F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
            F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
            F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
            F_xs(i) = rho/2*(v_circ(i-N) + v_circ(i-N+1));

        end % if

        etest = false;
        wtest = false;
        ntest = false;
        stest = false;

    end %for

    %Generation of F_y:
    for i = 1:m*N % Global indexing system

        % Eastern boundary requires no special treatment (x = L)
        wtest = mod(i-1, N) == 0;
        ntest = m*N - (N - 1) <= i && i <= m*N   ;
        stest = 1 <= i && i <= N   ;

        % Northwestern corner
        if wtest == true && ntest == true
            F_yw(i) = rho*u_in;                         % inlet
            F_yn(i) = rho/2*v_circ(i);                  % v_NorthWall = 0;

            F_ye(i) = rho/2*(u_circ(i) + u_circ(i+N));
            F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));

        % Southwestern corner
        elseif wtest == true && stest == true
            F_yw(i) = rho*u_in;                         % inlet
            F_ys(i) = rho/2*v_circ(i);                  % v_SouthWall = 0;
```

```
            F_ye(i) = rho/2*(u_circ(i) + u_circ(i+N));
            F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));

        % At western boundary (x = 0)
        elseif wtest == true && ntest == false && stest == false
            F_yw(i) = rho*u_in;                              % inlet

            F_ye(i) = rho/2*(u_circ(i) + u_circ(i+N));
            F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));
            F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));

        % At northern boundary (y = h)
        elseif ntest == true && wtest == false
            F_yn(i) = rho/2*v_circ(i);                       % v_NorthWall = 0;

            F_ye(i) = rho/2*(u_circ(i) + u_circ(i+N));
            F_yw(i) = rho/2*(u_circ(i-1) + u_circ(i-1+N));
            F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));

        % At southern boundary (y = 0)
        elseif stest == true && wtest == false
            F_ys(i) = rho/2*v_circ(i);                       % v_SouthWall = 0;

            F_ye(i) = rho/2*(u_circ(i) + u_circ(i+N));
            F_yw(i) = rho/2*(u_circ(i-1) + u_circ(i-1+N));
            F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));

        %Not at any boundary
        else
            F_ye(i) = rho/2*(u_circ(i) + u_circ(i+N));
            F_yw(i) = rho/2*(u_circ(i-1) + u_circ(i-1+N));
            F_yn(i) = rho/2*(v_circ(i) + v_circ(i+N));
            F_ys(i) = rho/2*(v_circ(i) + v_circ(i-N));

        end % if

        etest = false;
        wtest = false;
        ntest = false;
        stest = false;

    end % for

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% u-velocity
    for i = 1:M*N % Global indexing system

        etest = mod(i, N) == 0;
        wtest = mod(i-1, N) == 0;
        ntest = M*N - (N - 1) <= i && i <= N*M  ;
        stest = 1 <= i && i <= N  ;

        % Northeastern corner
        if etest == true && ntest == true
            % At eastern boundary (x = L)
            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff =  F_xe(i)*A_x;

            bu(i) =  -(p_out(end)-p_circ(i))*A_x;

            % At northern boundary
            nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y) + 2*D_y*A_y;
            %wall shear stress

            W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_xw(i)*A_x;
            U(i, i-1) = W_coeff;

            S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_xs(i)*A_y;
            U(i, i-N) = S_coeff;

        % Southeastern corner
        elseif etest == true && stest == true
            % At eastern boundary (x = L)
            E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
```

```
            eP_coeff =  F_xe(i)*A_x;

            bu(i) =  -(p_out(1)-p_circ(i))*A_x;

            % At southern boundary (y = 0)
            sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
            %wall shear stress

            W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_xw(i)*A_x;
            U(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_xn(i)*A_y;
            U(i, i+N) = N_coeff;

        % Northwestern corner
        elseif wtest == true && ntest == true
            % At western boundary (x = 0)
            wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

            % At northern boundary
            nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;
            %wall shear stress

            bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
                +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;

            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_xe(i)*A_x;
            U(i, i+1) = E_coeff;

            S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_xs(i)*A_y;
            U(i, i-N) = S_coeff;

        % Southwestern corner
        elseif wtest == true && stest == true
            % At western boundary (x = 0)
            wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

            % At southern boundary (y = 0)
            sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
            %wall shear stress

            bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
                +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;

            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_xe(i)*A_x;
            U(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_xn(i)*A_y;
            U(i, i+N) = N_coeff;

        % At eastern boundary (x = L)
        elseif etest == true && ntest == false && stest == false
            % At eastern boundary (x = L)
            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff =  F_xe(i)*A_x;

            bu(i) =  -(p_out(floor((i-1)/N)+1)-p_circ(i))*A_x;

            W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_xw(i)*A_x;
            U(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_xn(i)*A_y;
            U(i, i+N) = N_coeff;

            S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_xs(i)*A_y;
            U(i, i-N) = S_coeff;

```

```
        % At western boundary (x = 0)
        elseif wtest == true && ntest == false && stest == false
            % At western boundary (x = 0)
            wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

            bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
                +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;

            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_xe(i)*A_x;
            U(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_xn(i)*A_y;
            U(i, i+N) = N_coeff;

            S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_xs(i)*A_y;
            U(i, i-N) = S_coeff;

        % At northern boundary (y = h)
        elseif ntest == true && etest == false && wtest == false
            % At northern boundary
            nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;
            %wall shear stress

            bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_xe(i)*A_x;
            U(i, i+1) = E_coeff;

            W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_xw(i)*A_x;
            U(i, i-1) = W_coeff;

            S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_xs(i)*A_y;
            U(i, i-N) = S_coeff;

        % At southern boundary (y = 0)
        elseif stest == true && etest == false && wtest == false
            % At southern boundary (y = 0)
            sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;
            %wall shear stress

            bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;

            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_xe(i)*A_x;
            U(i, i+1) = E_coeff;

            W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_xw(i)*A_x;
            U(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_xn(i)*A_y;
            U(i, i+N) = N_coeff;

        %Not at any boundary
        else
            bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
            E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_xe(i)*A_x;
            U(i, i+1) = E_coeff;

            W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_xw(i)*A_x;
            U(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_xn(i)*A_y;
            U(i, i+N) = N_coeff;

            S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_xs(i)*A_y;
```

```
            U(i, i-N) = S_coeff;

        end % if

        % Filling in the rest of the matrix, adding all point coefficients
        U(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;

        etest = false;
        wtest = false;
        ntest = false;
        stest = false;

    end %for

    u_star = U\bu';

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% v-velocity
    for i = 1:m*N % Global indexing system
        bv(i) = -(p_circ(i+N)-p_circ(i))*A_y + rho*g_y*del_y*A_y;

        etest = mod(i, N) == 0;
        wtest = mod(i-1, N) == 0;
        ntest = m*N - (N - 1) <= i && i <= m*N  ;
        stest = 1 <= i && i <= N  ;

        % Northeastern corner
        if etest == true && ntest == true

            % At eastern boundary (x = L)
            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff =  F_ye(i)*A_x;

            if v_out_zero == true
                eP_coeff = eP_coeff + 1e+30;
            end %if

            % At northern boundary
            nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y;

            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, i-N) = S_coeff;

        % Southeastern corner
        elseif etest == true && stest == true

            % At eastern boundary (x = L)
            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff =  F_ye(i)*A_x;
            if v_out_zero == true
                eP_coeff = eP_coeff + 1e+30;
            end %if
            % At southern boundary (y = 0),
            sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, i+N) = N_coeff;

        % Northwestern corner
        elseif wtest == true && ntest == true

            % At western boundary (x = 0)
            wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

            % At northern boundary
            nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;
```

```
            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, i-N) = S_coeff;

        % Southwestern corner
        elseif wtest == true && stest == true

            % At western boundary (x = 0)
            wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

            % At southern boundary (y = 0),
            sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, i+N) = N_coeff;

        % At eastern boundary (x = L)
        elseif etest == true && ntest == false && stest == false

            % At eastern boundary (x = L)
            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff =  F_ye(i)*A_x;
            if v_out_zero == true
                eP_coeff = eP_coeff + 1e+30;
            end %if

            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, i+N) = N_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, i-N) = S_coeff;

        % At western boundary (x = 0)
        elseif wtest == true && ntest == false && stest == false

            % At western boundary (x = 0)
            wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, i+N) = N_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, i-N) = S_coeff;

        % At northern boundary (y = h)
        elseif ntest == true && etest == false && wtest == false

            % At northern boundary
            nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;
```

```
            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, i-N) = S_coeff;

        % At southern boundary (y = 0)
        elseif stest == true && etest == false && wtest == false

            % At southern boundary (y = 0),
            sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, i+N) = N_coeff;

        %Not at any boundary
        else

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, i+N) = N_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, i-N) = S_coeff;

        end % if

        % Filling in the rest of the matrix, adding all point coefficients
        V(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;

        etest = false;
        wtest = false;
        ntest = false;
        stest = false;

    end % for
    v_star = V\bv';
    if solvvel == false
        v_star = zeros(length(v_star),1);
    end %if

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Pressure correction
    au = diag(U);                    % a^center-coefficients for u-velocity
    av = diag(V);                    % a^center-coefficients for v-velocity

    for i = 1:M*N % Global indexing system

        etest = mod(i, N) == 0;
        wtest = mod(i-1, N) == 0;
        ntest = M*N - (N - 1) <= i && i <= N*M  ;
        stest = 1 <= i && i <= N  ;

        % Northeastern corner
        if etest == true && ntest == true

```

```
            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
                + A_y*v_star(i-N));

            % At eastern boundary (x = L)
            eP_coeff = rho*A_x^2/au(i);

            % At northern boundary
            nP_coeff = 0 ;

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            S_coeff = -rho*A_y^2/av(i-N);
            sP_coeff =  -S_coeff;
            T(i, i-N) = S_coeff;

        % Southeastern corner
        elseif etest == true && stest == true

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
                -A_y*v_star(i));

            % At eastern boundary (x = L)
            eP_coeff = rho*A_x^2/au(i);

            % At southern boundary (y = 0)
            sP_coeff = 0;

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, i+N) = N_coeff;

        % Northwestern corner
        elseif wtest == true && ntest == true

            beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in  ...
                + A_y*v_star(i-N));

            % At western boundary (x = 0)
            wP_coeff = 0;

            % At northern boundary
            nP_coeff = 0 ;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            S_coeff = -rho*A_y^2/av(i-N);
            sP_coeff =  -S_coeff;
            T(i, i-N) = S_coeff;

        % Southwestern corner
        elseif wtest == true && stest == true
            beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in ...
                -A_y*v_star(i));

            % At western boundary (x = 0)
            wP_coeff = 0;

            % At southern boundary (y = 0)
            sP_coeff = 0;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, i+N) = N_coeff;

```

```
        % At eastern boundary (x = L)
        elseif etest == true && ntest == false && stest == false

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                -A_y*v_star(i) + A_y*v_star(i-N));

            % At eastern boundary (x = L)
            eP_coeff = rho*A_x^2/au(i);

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, i+N) = N_coeff;

            S_coeff = -rho*A_y^2/av(i-N);
            sP_coeff =  -S_coeff;
            T(i, i-N) = S_coeff;

        % At western boundary (x = 0)
        elseif wtest == true && ntest == false && stest == false

            beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in ...
                -A_y*v_star(i) + A_y*v_star(i-N));

            % At western boundary (x = 0)
            wP_coeff = 0;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, i+N) = N_coeff;

            S_coeff =- rho*A_y^2/av(i-N);
            sP_coeff =  -S_coeff;
            T(i, i-N) = S_coeff;

        % At northern boundary (y = h)
        elseif ntest == true && etest == false && wtest == false

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                + A_y*v_star(i-N));

            % At northern boundary
            nP_coeff = 0 ;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            S_coeff = -rho*A_y^2/av(i-N);
            sP_coeff =  -S_coeff;
            T(i, i-N) = S_coeff;

        % At southern boundary (y = 0)
        elseif stest == true && etest == false && wtest == false

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                -A_y*v_star(i));

            % At southern boundary (y = 0)
            sP_coeff = 0;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

```

```
            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, i+N) = N_coeff;

        %Not at any boundary
        else

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
                -A_y*v_star(i) + A_y*v_star(i-N));

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, i+N) = N_coeff;

            S_coeff = -rho*A_y^2/av(i-N);
            sP_coeff =  -S_coeff;
            T(i, i-N) = S_coeff;

        end % if

        % Filling in the rest of the matrix, adding all point coefficients
        T(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;

        etest = false;
        wtest = false;
        ntest = false;
        stest = false;
    end % for
    p_corr = T\beta';

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Velocity correction

    for j = 1:length(p_corr)
        if mod(j, N) == 0                                % eastern boundary
            u_corr(j) =  - A_x/au(j)*(-p_corr(j));
                    % pressure correction is zero for known outlet pressure
        else
            u_corr(j) =  - A_x/au(j)*(p_corr(j+1)-p_corr(j));
        end % if
    end %for

    for k = 1:length(p_corr)-N
            v_corr(k) =  - A_y/av(k)*(p_corr(k+N)-p_corr(k));
    end %for

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Under-relaxation

    p_new = p_circ + alpha_p* p_corr;
    u_new = alpha_u*(u_star' + u_corr) + (1-alpha_u)*u_circ;
    v_new = alpha_v*(v_star' + v_corr) + (1-alpha_v)*v_circ;

    if solvvel == false
        v_new = zeros(1,length(v_new));
        annoSolvvel = '$v$-velocity: Not solved';
    else
        annoSolvvel = '$v$-velocity: Solved';
    end %if

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Check convergence
    % Make sure there are no mistakes in the matrix operations above
    if ~isvector(u_new) || ~isvector(p_new) || ~isvector(p_new)
```

```
        fprintf('u_new - %dx%d\n',size(u_new,1),size(u_new,2))
        fprintf('v_new - %dx%d\n',size(v_new,1),size(v_new,2))
        fprintf('p_new - %dx%d\n',size(p_new,1),size(p_new,2))
        error('Matrix addition gone wrong')
    end

    if isnan(rcond(U)) || isnan(rcond(V)) || isnan(rcond(T))
%         clc                               % Remove if warnings are desired
        fprintf('Stopped due to singularity in matrix\n')
        fprintf('RCOND u-velocity: %e \nRCOND v-velocity: %e \n',...
            rcond(U), rcond(V))
        fprintf('RCOND pressure: %e\n',rcond(T))
        fprintf('Problem occured after %d iterations\n', it)
        return
    end %if

    c1 = 1/u_in*sqrt((U*u_star-bu')'*(U*u_star-bu'));           % residuals
    c2 = 1/u_in*sqrt((V*v_star-bv')'*(V*v_star-bv'));           % residuals
    c3 = abs(sum(beta));                                % continuity fulfulled
    c4 = 1/u_in*max(abs(u_circ - u_star')) ;   % change from last iteration
    c5 = 1/u_in*max(abs(v_circ - v_star'))  ;  % change from last iteration


    c1_lim = 10^-8;                                                 % Limits
    c2_lim = 10^-8;
    c3_lim = 10^-10;
    c4_lim = 10^-8;
    c5_lim = 10^-8;


    if solvvel == false         % Overwrite if v-velocity is not solved for
        c2 = 0;
        c5 = 0;
    end %if

    c1_diff = c1-c1_lim;                      % How far away from convergence
    c2_diff = c2-c2_lim;
    c3_diff = c3-c3_lim;
    c4_diff = c4-c4_lim;
    c5_diff = c5-c5_lim;

    if  (c1 < c1_lim) && (c2 < c2_lim) && (c3 < c3_lim) && (c4 < c4_lim)...
            && (c5 < c5_lim) || (it == maxits)
        conv = 1;                                               % Converged
        if (it == maxits)
            fprintf('Stopped at max iterations (%d)\n',it);
        else
            fprintf('Solution converged after %d iterations\n',it);
        end %if

        fprintf('c1\tMomentum residual u\t\t%.2e\tLimit: %.2e\n',...
            c1,c1_lim);
        fprintf('c2\tMomentum residual v\t\t%.2e\tLimit: %.2e\n',...
            c2,c2_lim);
        fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',...
            c3,c3_lim);
        fprintf('c4\tDiff. last iteration u\t%.2e\tLimit: %.2e\n',...
            c4,c4_lim);
        fprintf('c5\tDiff. last iteration v\t%.2e\tLimit: %.2e\n',...
            c5,c5_lim);

        if max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c1_diff
            fprintf('Limiting criteria is c1\tMomentum residual u\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c2_diff
            fprintf('Limiting criteria is c2\tMomentum residual v\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c3_diff
            fprintf('Limiting criteria is c3\tPressure correction\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c4_diff
            fprintf('Limiting criteria is c4\tDiff. last iteration u\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c5_diff
            fprintf('Limiting criteria is c5\tDiff. last iteration u\n')
        end %if

    else

        u_circ = u_new  ;                 % Not converged, updated variables
```

```
        v_circ = v_new ;                  % Not converged, updated variables
        p_circ = p_new ;                  % Not converged, updated variables

    end % if

    if runiterationwise == 1 || conv == 1
        plot_2D
        if conv == 0 % if not converged
           pause
           close all
        end %if
    end %if

    it = it + 1;                               % Update number of iterations

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end %while
toc
```

E.4.1.2 plot_2D.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%              Plotting of the two dimensional fluid flow                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Velocities and pressure back to matrices

u_new_plot = zeros(M+2,N+1);

u_new_plot(:,1) = u_in_true;
u_star_plot(:,1) = u_in_true;

u_new_plot(1,1) = Inf;              % The walls at the inlet are blocked out
u_new_plot(end,1) = Inf;

for j = 1:M
    for i = 1:N
        u_new_plot(j+1,i+1) = u_new((j-1)*N + i)*u_in_true;
    end % for
end % for


v_new_plot = zeros(M+1,N+1);

for j = 1:m                               % The rest of the points are zero
    for i = 1:N
        v_new_plot(j+1,i+1) = v_new((j-1)*N + i)*u_in_true;
    end % for
end % for


p_plot = zeros(M,N+1);
p_corrplot = zeros(M,N+1);

p_plot(:,N+1) = p_atm;

for j = 1:M                               % The rest of the points are zero
    for i = 1:N
        p_plot(j,i) = p_new((j-1)*N + i)*rho_true*u_in_true + p_atm;
%            ;
        p_corrplot(j,i) = p_corr((j-1)*N + i)*rho_true*u_in_true;
    end % for
end % for

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot
az_outlet = 45;             % Azimuth angle for setting viewpoint in figures
el_outlet = 30;          % Elevation height for setting viewpoint in figures

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f1 = figure;
f = surf(xu_plot,yu_plot,u_new_plot);          % surf(x,y,z)
s = sprintf('Plot of $u_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
```

```
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
saveas(gcf,'unew2D.png')

f1_outlet = figure;
f = surf(xu_plot,yu_plot,u_new_plot);          % surf(x,y,z)
view(az_outlet, el_outlet)
s = sprintf('Plot of $u_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
saveas(gcf,'unewoutlet2D.png')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f2 = figure;
f = surf(xv_plot,yv_plot,v_new_plot);          % surf(x,y,z)
s = sprintf('Plot of $v_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
saveas(gcf,'vnew2D.png')

f2_outlet = figure;
f = surf(xv_plot,yv_plot,v_new_plot);          % surf(x,y,z)
view(az_outlet, el_outlet)
s = sprintf('Plot of $v_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
saveas(gcf,'vnewoutlet2D.png')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f3 = figure;
surf(xp_plot,yp_plot,p_corrplot);          % surf(x,y,z)
s = sprintf('Plot of $p^{corr}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure correction $p''$, [Pa]', 'interpreter', 'latex')
ztickformat('%.2f')
% zlim([-0.1 0.1])
saveas(gcf,'pcorr2D.png')

f3_outlet = figure;
surf(xp_plot,yp_plot,p_corrplot);          % surf(x,y,z)
view(az_outlet, el_outlet)
s = sprintf('Plot of $p^{corr}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure correction $p''$, [Pa]', 'interpreter', 'latex')
% zlim([-0.1 0.1])
ztickformat('%.2f')
saveas(gcf,'pcorroutlet2D.png')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
f4 = figure;
f = surf(xp_plot,yp_plot,p_plot);          % surf(x,y,z)
```

```
s = sprintf('Plot of $p_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
ztickformat('%.7f')
saveas(gcf,'pnew2D.png')

f4_outlet = figure;
f = surf(xp_plot,yp_plot,p_plot);          % surf(x,y,z)
view(az_outlet, el_outlet)
s = sprintf('Plot of $p_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
ztickformat('%.7f')
saveas(gcf,'pnewoutlet2D.png')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plotting the continuity

 for i = 1:M*N % Global indexing system

        etest = mod(i, N) == 0;
        wtest = mod(i-1, N) == 0;
        ntest = M*N - (N - 1) <= i && i <= N*M  ;
        stest = 1 <= i && i <= N  ;

        % Northeastern corner
        if etest == true && ntest == true
            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
                + A_y*v_star(i-N));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1))  ;
            cont_y(i) = rho*( A_y*v_star(i-N)) ;

        % Southeastern corner
        elseif etest == true && stest == true
            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
                -A_y*v_star(i));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)) ;
            cont_y(i) = rho*(-A_y*v_star(i)) ;

        % Northwestern corner
        elseif wtest == true && ntest == true
            beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in  ...
                + A_y*v_star(i-N));
            cont_x(i) =  rho*(-A_x*u_star(i) +A_x*u_in);
            cont_y(i) =  rho*(A_y*v_star(i-N));

        % Southwestern corner
        elseif wtest == true && stest == true
            beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in ...
                -A_y*v_star(i));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_in) ;
            cont_y(i) = rho*(-A_y*v_star(i)) ;

        % At eastern boundary (x = L)
        elseif etest == true && ntest == false && stest == false
            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                -A_y*v_star(i) + A_y*v_star(i-N));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)) ;
            cont_y(i) = rho*(-A_y*v_star(i) + A_y*v_star(i-N)) ;

        % At western boundary (x = 0)
        elseif wtest == true && ntest == false && stest == false
            beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in ...
                -A_y*v_star(i) + A_y*v_star(i-N));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_in) ;
            cont_y(i) = rho*(-A_y*v_star(i) + A_y*v_star(i-N)) ;

        % At northern boundary (y = h)
```

```
        elseif ntest == true && etest == false && wtest == false
            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                + A_y*v_star(i-N));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)) ;
            cont_y(i) = rho*(A_y*v_star(i-N)) ;

        % At southern boundary (y = 0)
        elseif stest == true && etest == false && wtest == false
            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                -A_y*v_star(i));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1));
            cont_y(i) = rho*(-A_y*v_star(i));

        %Not at any boundary
        else
            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
                -A_y*v_star(i) + A_y*v_star(i-N));
            cont_x(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1))  ;
            cont_y(i) = rho*(-A_y*v_star(i) + A_y*v_star(i-N)) ;


        end % if
 end %for

beta_plot = zeros(M,N);
cont_x_plot = zeros(M,N);
cont_y_plot = zeros(M,N);

for j = 1:M                               % the rest of the points are zero
    for i = 1:N
        cont_x_plot(j,i) = cont_x((j-1)*N + i);
        cont_y_plot(j,i) = cont_y((j-1)*N + i);
        beta_plot(j,i) = beta((j-1)*N + i);
    end % for
end % for


if contplots
    f5 = figure;
    f = surf(xp_plot(1:end-1),yp_plot, cont_x_plot );          % surf(x,y,z)
    s = sprintf(...
        'Plot of $x-$component of continuity after %d iterations', it );
    f = title(s);
    set(f, 'interpreter', 'latex', 'fontsize', 16)
    set(gca,'TickLabelInterpreter','latex')
    xlabel('$x$-direction [m]', 'interpreter', 'latex')
    ylabel('$y$-direction [m]', 'interpreter', 'latex')
    zlabel('Mass flow rate [kg/s]', 'interpreter', 'latex')
    ztickformat('%.2f')
    saveas(gcf,'cont_x.png')


    f6 = figure;
    f = surf(xp_plot(1:end-1),yp_plot, cont_y_plot );          % surf(x,y,z)
    s = sprintf(...
        'Plot of $y-$component of continuity after %d iterations', it );
    f = title(s);
    set(f, 'interpreter', 'latex', 'fontsize', 16)
    set(gca,'TickLabelInterpreter','latex')
    xlabel('$x$-direction [m]', 'interpreter', 'latex')
    ylabel('$y$-direction [m]', 'interpreter', 'latex')
    zlabel('Mass flow rate [kg/s]', 'interpreter', 'latex')
    ztickformat('%.2f')
    saveas(gcf,'cont_y.png')

    f7 = figure;
    f = surf(xp_plot(1:end-1),yp_plot, beta_plot );          % surf(x,y,z)
    s = sprintf(...
        'Plot of $\\beta$ (continuity) after %d iterations', it );
    f = title(s);
    set(f, 'interpreter', 'latex', 'fontsize', 16)
    set(gca,'TickLabelInterpreter','latex')
    xlabel('$x$-direction [m]', 'interpreter', 'latex')
    ylabel('$y$-direction [m]', 'interpreter', 'latex')
    zlabel('Mass flow rate [kg/s]', 'interpreter', 'latex')
    ztickformat('%.2f')
```

```
    saveas(gcf,'beta.png')
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, pressurre 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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Two dimensional fluid flow over a backwards facing step, dimensionless  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear
clc
tic
warning on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Solver specs
maxits = 25000;   % Maximum number of iterations, stop if iterations exceed

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Options
plotiterationwise = false;        % Plots the profiles after each iteration
solvvel = true;                                        % Solve for v-velocity
plotCircVels = false;                               % Plot u_circ and v_circ
plotCorrVels = false;                               % Plot u_corr and v_corr
```

```
showVelociyQuiver = true;                          % Plot velocity quiver plots
plotInitialProfiles = false;                         % Plot the initial guesses
onlyChannel = false;% Turn off the BFS, transform model to straight channel

% Make .gif file of the profiles before convergence is reached:
printSetPlotIt = false;
% Also create a .gif of the u-and v-velocities with their intermediates:
gifIntermediates = false;
% Vector of the iterations for which to save the plots to the .gif files:
itSaves = [1 2 3 4 5 10:10:100 100:100:maxits];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System specifications
% Specify number of narrow points, leave the rest
N_narrow = 12;  % Number of scalar nodal points in narrow section in x-dir.
M_narrow = 12;  % Number of scalar nodal points in narrow section in y-dir.

l = 3;                                                  % Narrow channel length
h = 1;                                                  % Narrow channel height
L = 19;                                                   % Wide channel length
H = 0.5;                                                  % Wide channel height

L_total = l + L;                                         % Total channel length
H_total = h + H;                                         % Total channel height

x_0 = 0;                                    % Defining the domain using x and y
x_N = L_total;
y_0 = 0;
y_M = H_total;


if mod(N_narrow,3)~=0 || mod(M_narrow,2)~=0
    msg = 'Points don''t match dimensions';
    error(msg)
end %if

N_wide = N_narrow*19/3;    % # scalar nodal points in wide section in x-dir.
M_wide = M_narrow*1/2;     % # scalar nodal points in wide section in y-dir.

N_total = N_narrow + N_wide;% Total # of scalar nodal points in x-direction
M_total = M_narrow + M_wide;% Total # of scalar nodal points in y-direction

m_total = M_total - 1;     % Total number of y-velocity nodes in y-direction
m_wide = M_wide;% Number of y-velocity nodes in y-direction in wide section
m_narrow = M_narrow - 1;% # of y-velocity nodes in y-dir. in narrow section

% Total number of computational points in the domain ...
totalpoints = N_narrow*M_narrow + N_wide*M_total;        %   ... for u and P
totalpoints_v = N_narrow*m_narrow + N_wide*m_total;             % ... for v

D_hyd = 4*h*1/(1+1+h+h);                                  % Hydraulic diameter
mu_true = 8.90 * 10^-4;                                   % Viscosity of water

del_z_true = 1;                                                 % System depth
del_x_true = L_total/N_total;                           % Control volume width
del_y_true = H_total/M_total;                          % Control volume height
A_x_true = del_y_true*del_z_true;     % Cross-sectional area in x-direction
A_y_true = del_x_true*del_z_true;     % Cross-sectional area in y-direction

rho_true = 997;                                             % Density of water
u_in_true = 0.0005;                                         % Inlet u-velocity

g_x = 0;                                                      % No gravitation
g_y = 0;                                                      % No gravitation

Re = rho_true*D_hyd*u_in_true/mu_true;                       % Reynolds number

p_atm = 101325;                           % Atmospheric presssure at outlet
p_out_tilde = 0;                                           % Adjusted pressure
p_out = ones(1,M_total)*p_out_tilde;                 % Outlet pressure profile

alpha_u = 0.005;                             % Under-relaxation factor for u
alpha_v = 0.005;                             % Under-relaxation factor for v
alpha_p = 0.01;                              % Under-relaxation factor for p

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% Dimensionless parameters
mu = 1;                                              % Dimensionless viscosity
rho = 1;                                               % Dimensionless density
del_x = del_x_true/D_hyd;              % Dimensionless control volume width
del_y = del_y_true/D_hyd;             % Dimensionless control volume height
A_x = A_x_true/D_hyd^2; % Dimensionless cross-sectional area in x-direction
A_y = A_y_true/D_hyd^2; % Dimensionless cross-sectional area in y-direction
D_x = 1/Re*mu/del_x;    % Dimensionless diffusion conductance in x-direction
D_y = 1/Re*mu/del_y;    % Dimensionless diffusion conductance in y-direction
u_in = 1;                                                   % Inlet u-velocity
v_in = 0;                                                   % Inlet u-velocity
u_guess = 1.0;                                  % Initial guess for u-velocity
v_guess = 0.0;                                  % Initial guess for v-velocity
p_guess = 0/(rho_true*u_in_true^2);               % Initial guess for pressure

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialisation of p
% Filling in initial pressure vector with the linear profile.
% This section is set up for if gravity is added, but could be more compact
% if the option to add gravity was not there.

p_circ_y_wide = linspace(p_guess, p_guess+rho*g_y*H_total,M_total);
p_circ_carthesian_wide = zeros(M_total,N_wide);
for j = 1:M_total
    for i = 1:N_wide
        p_circ_carthesian_wide(j,i) = p_circ_y_wide(j);
    end %for
end %for

p_circ_y_narrow = p_circ_y_wide(M_wide+1:end);
p_circ_carthesian_narrow = zeros(M_narrow,N_narrow);
for j = 1:M_narrow
    for i = 1:N_narrow
        p_circ_carthesian_narrow(j,i) = p_circ_y_narrow(j);
    end %for
end %for

filler = zeros(M_wide, N_narrow);
p_circ_carthesian = [[filler; p_circ_carthesian_narrow] ...
    p_circ_carthesian_wide ];
p_circ_carthesian = flip(p_circ_carthesian,1);

p_circ = p_circ_carthesian(1,:);                       % Take the first vector

for i = 2:M_total
    row = p_circ_carthesian(i);
    if i <= M_narrow                                          % Take whole row
        p_circ = [p_circ, p_circ_carthesian(i,:)];
    else                                                % Take part of the row
        p_circ = [p_circ, p_circ_carthesian(i,N_narrow+1:N_total)];
    end %if
end %for

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialisation of u and v

u_circ = ones(totalpoints,1)*u_guess; % Fill in guess in the initial vector
if ~onlyChannel             % Only for the normal mode with the BFS enabled
    for i = 1:totalpoints
        if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
            u_circ(i) = u_guess*(M_narrow/M_total);
        end %if
    end %for
end %if


v_circ = ones(totalpoints_v,1)*v_guess; % Fill in guess in the initial vec.
if ~onlyChannel             % Only for the normal mode with the BFS enabled
    for i = 1:totalpoints_v
        if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
            v_circ(i) = v_guess*(m_narrow/m_total);
        end %if
    end %for
end %if

if plotInitialProfiles == true       % Plot the initial profiles if desired
```

```
    it = 0;
    u_new = u_circ;
    v_new = v_circ;
    p_new = p_circ;
    if printSetPlotIt == true
         plotProfilesITSAVE_subplots;
    else
        plotProfiles_dimensionless
        pause
        close all
    end % if
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialisation of solution vectors
p_new = zeros(1, totalpoints);                                      % New pressure

u_corr = zeros(1, totalpoints);                            % u-velocity correction
u_new = zeros(1, totalpoints);                                     % New u-velocity

v_corr = zeros(1, totalpoints_v);                          % v-velocity correction
v_new = zeros(1, totalpoints_v);                                   % New v-velocity

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% While loop
conv = 0;                              % 0 is not converged, 1 when converged
it = 1;                                                   % The current iteration

while  conv == 0
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%  Calculate velocities and pressure correction
    % Run the scripts:
    % Velocities
    BFS_u_velocity
    BFS_v_velocity
    if solvvel == false
        v_star = zeros(totalpoints_v,1);
    end %if

    % Pressure correction
    BFS_pressurecorrection

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Velocity correction

    startCorr = 1;
    for j = startCorr:totalpoints
        if ( i <= N_wide*M_wide && mod(i, N_wide) == 0 ) ...    % Below step
             || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0)
            % Eastern boundary : eastern pressure is known, no press. corr.
            u_corr(j) =  - A_x/au(j)*(-p_corr(j));
        else
            u_corr(j) =  - A_x/au(j)*(p_corr(j+1)-p_corr(j));
        end % if
    end %for

    for k = startCorr:totalpoints_v
            v_corr(k) = - A_y/av(k)*...
                (p_corr(getRowOver(k, N_wide, M_wide, N_total))-p_corr(k));
    end %for

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Under-relaxation

    u_new = alpha_u*(u_star + u_corr') + (1-alpha_u)*u_circ;

    if solvvel == false
        v_new = zeros(totalpoints_v,1);
    else
        v_new = alpha_v*(v_star + v_corr') + (1-alpha_v)*v_circ;
    end %if

    p_new = p_circ + alpha_p* p_corr';

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Check convergence
```

```
    % Make sure there are no mistakes in the matrix operations above
    if ~isvector(u_new) || ~isvector(p_new) || ~isvector(p_new)
        fprintf('u_new - %dx%d\n',size(u_new,1),size(u_new,2))
        fprintf('v_new - %dx%d\n',size(v_new,1),size(v_new,2))
        fprintf('p_new - %dx%d\n',size(p_new,1),size(p_new,2))
        error('Matrix addition gone wrong')
    end

    if isnan(rcond(U)) || isnan(rcond(V)) || isnan(rcond(T))
%           clc                                 % Remove if warnings are desired
        fprintf('Stopped due to singularity in matrix\n')
        fprintf('RCOND u-velocity: %e \nRCOND v-velocity: %e \n',...
            rcond(U), rcond(V))
        fprintf('RCOND pressure correction: %e\n',rcond(T))
        fprintf('Problem occured after %d iterations\n', it)
        toc
        return
    end %if

    c1 = 1/u_in*sqrt((U*u_star-bu')'*(U*u_star-bu'));           % residuals
    c2 = 1/u_in*sqrt((V*v_star-bv')'*(V*v_star-bv'));           % residuals
    c3 = abs(sum(beta));                                % continuity fulfulled
    c4 = 1/u_in*max(abs(u_circ - u_star)) ;     % change from last iteration
    c5 = 1/u_in*max(abs(v_circ - v_star))  ;    % change from last iteration

    c1_lim = 10^-8;                                                 % Limits
    c2_lim = 10^-8;
    c3_lim = 10^-10;
    c4_lim = 10^-8;
    c5_lim = 10^-8;

    if solvvel == false          % Overwrite if v-velocity is not solved for
        c2 = 0;
        c5 = 0;
    end %if

    c1_diff = c1-c1_lim;                        % How far away from convergence
    c2_diff = c2-c2_lim;
    c3_diff = c3-c3_lim;
    c4_diff = c4-c4_lim;
    c5_diff = c5-c5_lim;


   if  (c1 < c1_lim) && (c2 < c2_lim) && (c3 < c3_lim) && (c4 < c4_lim) ...
            && (c5 < c5_lim) || (it == maxits)
        conv = 1;                                                % Converged
        if (it == maxits)
            fprintf('Stopped at max iterations (%d)\n',it);
        else
            fprintf('Solution converged after %d iterations\n',it);
        end %if

        fprintf('c1\tMomentum residual u\t\t%.2e\tLimit: %.2e\n',...
            c1,c1_lim);
        fprintf('c2\tMomentum residual v\t\t%.2e\tLimit: %.2e\n',...
            c2,c2_lim);
        fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',...
            c3,c3_lim);
        fprintf('c4\tDiff. last iteration u\t%.2e\tLimit: %.2e\n',...
            c4,c4_lim);
        fprintf('c5\tDiff. last iteration v\t%.2e\tLimit: %.2e\n',...
            c5,c5_lim);

        if max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c1_diff
            fprintf('Limiting criteria is c1\tMomentum residual u\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c2_diff
            fprintf('Limiting criteria is c2\tMomentum residual v\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c3_diff
            fprintf('Limiting criteria is c3\tPressure correction\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c4_diff
            fprintf('Limiting criteria is c4\tDiff. last iteration u\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c5_diff
            fprintf('Limiting criteria is c5\tDiff. last iteration u\n')
        end %if

        showStep = false;
```

```
        if plotCircVels == true
            plotIntermediates
        end
        if plotCorrVels == true
            plotVelocityCorrection
        end
        plot_BFS
        if showVelociyQuiver == true
            plotVelocityQuiver
            plotColoredQuiver
        end %if

    else
        if plotiterationwise == true
            showStep = false;
            if plotCircVels == true
                plotIntermediates
            end
            if plotCorrVels == true
                plotVelocityCorrection
            end
            plot_BFS
            if showVelociyQuiver == true
                plotVelocityQuiver
                plotColoredQuiver
            end %if
            pause
            close all
        end %if
        if printSetPlotIt && ismember(it,itSaves)
            plot_BFS_iterations
            if gifIntermediates == true
                plotVelInts_BFS_iterations;
            end %if
        end

        u_circ = u_new;                   % Not converged, updated variables
        v_circ = v_new;                   % Not converged, updated variables
        p_circ = p_new;                   % Not converged, updated variables

        it = it + 1;  % Update number of iterations
    end % if
end %while
toc
```

E.5.1.2 BFS_u_velocity.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    u-velocity script for the BFS model                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

U = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
bu = zeros(1, totalpoints);          % Initialisation of source term vector

F_xe = zeros(1, totalpoints);    % Initialisation of convective mass fluxes
F_xw = zeros(1, totalpoints);
F_xn = zeros(1, totalpoints);
F_xs = zeros(1, totalpoints);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generation of F_x, Convective mass fluxes


for i = 1:totalpoints

    etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )...    % below step
          || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
    wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
    ntest = totalpoints - N_total < i && i <= totalpoints   ;
    if ~onlyChannel                                             % Normal mode
        wwall = i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
        stest = (1 <= i && i <= N_wide) ...     % Excluding the corner value
            || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
        scorner = i == N_wide*M_wide + N_narrow;    % Only the corner value
    else                                                       % No step mode
        wwall =  i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
```

```
        stest = i <= N_wide*M_wide + N_total;  % Excluding the corner value
        scorner = false;                            % Only the corner value
    end %if


    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
        F_xe(i) = rho/2*(u_circ(i)+ u_circ(i-1));
        F_xn(i) = 0;

        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xs(i) = rho/2*v_circ(i-N_total);

    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
        F_xe(i) = rho/2*(u_circ(i)+u_circ(i-1));
        F_xs(i) = 0;

        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*v_circ(i);

    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner
        F_xw(i) = rho/2*(u_in+u_circ(i));
        F_xn(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xs(i) = rho/2*(v_circ(i-N_total) + v_circ(i-N_total+1));

    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner
        F_xw(i) = rho/2*(u_in+u_circ(i));
        F_xs(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % Southwestern corner at step
    elseif ~etest && ~wtest && ~ntest && stest && wwall && ~scorner
        F_xw(i) = rho/2*(0 + u_circ(i));
        F_xs(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % At corner
    elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner
        F_xs(i) = rho/2*(0 + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)+1));
        F_xs(i)= 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % At eastern boundary (x = L)
    elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner
        F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));

        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*v_circ(i);
        F_xs(i) = rho/2*v_circ(getRowUnder(i, N_wide, M_wide, N_total));


    % At western boundary (x = 0)
    elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner
        F_xw(i) = rho/2*(u_in+u_circ(i));

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
        F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );

     % At western wall at step
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner
```

```
        F_xw(i) = rho/2*(0+u_circ(i));

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
        F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );



    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
       F_xn(i) = 0;

       F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
       F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
       F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );


    % At southern boundary (y = 0)
    elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
        F_xs(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % Not at any boundary
    else
        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
        F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );


    end % if
    etest = false;
    wtest = false;
    wwall = false;
    ntest = false;
    stest = false;
    scorner = false;
end %for



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% u-velocity


for i = 1:totalpoints                              % Global indexing system

    etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )...    % below step
        || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
    wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
    ntest = totalpoints - N_total < i && i <= totalpoints  ;
    if ~onlyChannel                                           % Normal mode
        wwall = i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
        stest = (1 <= i && i <= N_wide) ...     % Excluding the corner value
            || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
        scorner = i == N_wide*M_wide + N_narrow;    % Only the corner value
    else                                                     % No step mode
        wwall =  i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
        stest = i <= N_wide*M_wide + N_total;  % Excluding the corner value
        scorner = false;                            % Only the corner value
    end %if


    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner

        bu(i) =  -(p_out(end)-p_circ(i))*A_x;
```

```

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
                eP_coeff =  F_xe(i)*A_x;

        % At northern boundary
        nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y) + 2*D_y*A_y;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner

        bu(i) =  -(p_out(1)-p_circ(i))*A_x;

        % At eastern boundary (x = L)
        E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff =  F_xe(i)*A_x;

        % At southern boundary (y = 0)
        sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
            +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;

        % At western boundary (x = 0)
        wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

        % At northern boundary
        nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
            +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;

        % At western boundary (x = 0)
        wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

        % At southern boundary (y = 0)
        sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
```

```
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Southwestern corner at step
    elseif  ~etest && ~wtest && ~ntest && stest && wwall  && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
            +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;

        % At western boundary (x = 0)
        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;

        % At southern boundary (y = 0)
        S_coeff = -max(F_xs(i)*A_y,0) - 2*D_y*A_y;
        sP_coeff = -S_coeff -F_xs(i)*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % At corner
    elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;

        % At southern boundary (y = 0)
        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;


        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % At eastern boundary (x = L)
    elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner

        bu(i) =  -(p_out(1)-p_circ(i))*A_x;

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff =  F_xe(i)*A_x;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western boundary (x = 0)
    elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
```

```
            +(max(F_xw(i)*A_x,0) + D_x*A_x)*u_in;

        % At western boundary (x = 0)
        wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western wall
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
            +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;

        % At western boundary (x = 0)
        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest && ~wwall  && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;

        % At northern boundary
        nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At southern boundary (y = 0)
    elseif ~etest && ~wtest && ~ntest && stest && ~wwall  && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;

        % At southern boundary (y = 0)
        sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
```

```
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    %Not at any boundary
    else

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    end % if

    % Filling in the rest of the matrix, adding all point coefficients
    U(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;

    % If the step is disabled the points below the step are blocked out
    if onlyChannel && i <= N_wide*M_wide
        U(i,i) = U(i,i) + 10e+30;
    end %if

    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;

end %for
u_star = U\bu';                                       % Matrix inversion
```

E.5.1.3 BFS_v_velocity.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    v-velocity script for the BFS model                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V = zeros(totalpoints_v, totalpoints_v);  % Initialisation of coeff. matrix
bv = zeros(1, totalpoints_v);        % Initialisation of source term vector

F_ye = zeros(1, totalpoints_v);  % Initialisation of convective mass fluxes
F_yw = zeros(1, totalpoints_v);
F_yn = zeros(1, totalpoints_v);
F_ys = zeros(1, totalpoints_v);




%% Generation of F_y, Convective mass fluxes

for i = 1:totalpoints_v % Global indexing system

    % Eastern boundary requires no special treatment (x = L)
    etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ...  % below step
          || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0);
    wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
    ntest = totalpoints_v - N_total < i && i <= totalpoints_v  ;
    if ~onlyChannel                                            % Normal mode
        wwall = i <= N_wide*m_wide  && mod(i-1, N_wide) == 0; %
        stest = (1 <= i && i <= N_wide) ...     % Excluding the corner value
                || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow) ;
```

```
        wcorner = i == N_wide*(m_wide-1) + 1;       % Only the corner value
    else                                                 % No step mode
        wwall = i <= N_wide*m_wide  && mod(i-1, N_wide) == 0; %
        stest = i <= N_wide*m_wide + N_total;  % Excluding the corner value
        wcorner = false;                           % Only the corner value
    end %if



    % Northwestern corner
    if  wtest && ntest && ~stest && ~wwall && ~wcorner
        F_yw(i) = rho/2*(u_in+u_in);
        F_yn(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

    % Southwestern corner at inlet
    elseif  wtest && ~ntest && stest && ~wwall && ~wcorner
        F_yw(i) = rho*u_in;
        F_ys(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));


    % Southwestern corner at step
    elseif  ~wtest && ~ntest && stest && wwall && ~wcorner
        F_yw(i) = rho*0;
        F_ys(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));


    % At western boundary (x = 0)
    elseif  wtest && ~ntest && ~stest && ~wwall && ~wcorner
        F_yw(i) = rho*u_in;

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));


    % At western wall
    elseif  ~wtest && ~ntest && ~stest && wwall && ~wcorner
        F_yw(i) = rho*0;

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));


    % At corner, right point from the corner
    elseif  ~wtest && ~ntest && ~stest && wwall && wcorner
        F_yw(i)= 0;

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
```

```
    % At northern boundary (y = h)
    elseif  ~wtest && ntest && ~stest && ~wwall && ~wcorner
        F_yn(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yw(i) = rho/2*(u_circ(i-1) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));


    % At southern boundary (y = 0)
    elseif ~wtest && ~ntest && stest && ~wwall && ~wcorner
        F_ys(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yw(i) = rho/2*(u_circ(i-1) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));


    %Not at any boundary, including eastern boundary
    else
        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yw(i) = rho/2*(u_circ(i-1) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));

        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

    end % if
    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;
    wcorner = false;

end % for

%% v-velocity


for i = 1:totalpoints_v                           % Global indexing system

    etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ...  % below step
          || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0);
    wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
    ntest = totalpoints_v - N_total < i && i <= totalpoints_v  ;
    if ~onlyChannel                                         % Normal mode
        wwall = i <= N_wide*m_wide  && mod(i-1, N_wide) == 0; %
        stest = (1 <= i && i <= N_wide) ...    % Excluding the corner value
                || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow) ;
        wcorner = i == N_wide*(m_wide-1) + 1;       % Only the corner value
    else                                                   % No step mode
        wwall = i <= N_wide*m_wide  && mod(i-1, N_wide) == 0; %
        stest = i <= N_wide*m_wide + N_total;  % Excluding the corner value
        wcorner = false;                            % Only the corner value
    end %if



    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
```

```
        eP_coeff = F_ye(i)*A_x;

        % At northern boundary
        nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall    && ~wcorner
        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = F_ye(i)*A_x;

        % At southern boundary (y = 0),
        sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall && ~wcorner
        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))-...
                p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At western boundary (x = 0)
        wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

        % At northern boundary
        nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At western boundary (x = 0)
        wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

        % At southern boundary (y = 0),
        sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

    % Southwestern corner at step
    elseif ~etest && ~wtest && ~ntest && stest && wwall    && ~wcorner

```

```
            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
                0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);

            % At western boundary (x = 0)
            W_coeff =  -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;

            % At southern boundary (y = 0),
            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        % At eastern boundary (x = L)
        elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~wcorner

            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y;

            % At eastern boundary (x = L)
            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = F_ye(i)*A_x;

            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

        % At western boundary (x = 0)
        elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~wcorner

            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y;

            % At western boundary (x = 0)
            wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

        % At west wall (x = 0) [EXCLUDED CORNER]
        elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~wcorner

            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
                0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);

            % At western boundary (x = 0)
            W_coeff =  -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
```

```
        V(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    % At corner
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
            0*(-max(F_yw(i)*A_x,0) - D_x*A_x);

        % At western boundary (x = 0)
        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

       % At northern boundary
        nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At southern boundary (y = 0)
    elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At southern boundary (y = 0),
        sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
```

```
    %Not at any boundary
    else

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    end % if

    % Filling in the rest of the matrix, adding all point coefficients
    V(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;

    % If the step is disabled the points below the step are blocked out
    if onlyChannel && i <= N_wide*m_wide
        V(i,i) = V(i,i) + 10e+30;
    end %if

    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;

end % for
v_star = V\bv';                                      % Matrix inversion
```

E.5.1.4 BFS_pressurecorrection.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               Pressure correction script for the BFS model               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

T = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
beta = zeros(1, totalpoints);        % Initialisation of source term vector

au = diag(U);             % a^center-coefficients from the momentum equations
av = diag(V);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculation
for i = 1:totalpoints                             % Global indexing system

    etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 ) ...  % below step
        || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
    ntest = totalpoints - N_total < i && i <= totalpoints   ;
    wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;

    if ~onlyChannel                                          % Normal Mode
        wwall = i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
        stest = (1 <= i && i <= N_wide) ...     % Excluding the corner value
            || (N_wide*M_wide < i && i <= N_wide*M_wide + N_narrow) ;
    else                                                    % No step mode
        wwall = i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
        stest = i <= N_wide*M_wide + N_total;  % Excluding the corner value
    end


    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall
```

```

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
            + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At eastern boundary (x = L)
        eP_coeff = rho*A_x^2/au(i);

        % At northern boundary (y = h) (y = H)
        nP_coeff = 0 ;

        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
            -A_y*v_star(i));

        % At eastern boundary (x = L)
        eP_coeff = rho*A_x^2/au(i);

        % At southern boundary (y = 0)
        sP_coeff = 0;

        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall

        beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in  ...
            + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At western boundary (x = 0)
        wP_coeff = 0;

        % At northern boundary (y = h) (y = H)
        nP_coeff = 0 ;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall

        beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in ...
            -A_y*v_star(i));

        % At western boundary (x = 0)
        wP_coeff = 0;

        % At southern boundary (y = 0)
        sP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;
```

```
        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Southwestern corner at step
    elseif  ~etest && ~wtest && ~ntest && stest && wwall

        beta(i) =  rho*(-A_x*u_circ(i)...
                    +A_x*0 -A_y*v_circ(i)); % wall/"inlet" velocity is zero

        % At western boundary (x = 0)
        wP_coeff = 0;

        % At southern boundary (y = 0)
        sP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % At eastern boundary (x = L)
    elseif etest && ~wtest && ~ntest && ~stest && ~wwall

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
            -A_y*v_star(i) + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At eastern boundary (x = L)
        eP_coeff = rho*A_x^2/au(i);


        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western boundary at inlet (x = 0)
    elseif ~etest && wtest && ~ntest && ~stest && ~wwall

        beta(i) =  rho*(-A_x*u_star(i) +A_x*u_in -A_y*v_star(i) ...
            + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At western boundary (x = 0)
        wP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff =- rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western wall
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall

        beta(i) =  rho*(-A_x*u_circ(i)...  % West wall / inlet velocity is zero
```

```
                        +A_x*0 -A_y*v_circ(i) +...
                        A_y*v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

            % At western boundary (x = 0)
            wP_coeff = 0;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

            S_coeff =- rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
            sP_coeff =  -S_coeff;
            T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


        % At northern boundary (y = h)
        elseif ~etest && ~wtest && ntest && ~stest && ~wwall

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

            % At northern boundary (y = h)
            nP_coeff = 0 ;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
            sP_coeff =  -S_coeff;
            T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


        % At southern boundary (y = 0)
        elseif ~etest && ~wtest && ~ntest && stest && ~wwall

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
                -A_y*v_star(i));

            % At southern boundary (y = 0)
            sP_coeff = 0;

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;

            N_coeff = -rho*A_y^2/av(i);
            nP_coeff = -N_coeff;
            T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


        %Not at any boundary
        else

            beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) -A_y*v_star(i) + ...
                A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

            E_coeff = -rho*A_x^2/au(i);
            eP_coeff = -E_coeff ;
            T(i, i+1) = E_coeff;

            W_coeff =  -rho*A_x^2/au(i-1);
            wP_coeff = -W_coeff;
            T(i, i-1) = W_coeff;
```

```

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    end % if

    % Filling in the rest of the matrix, adding all point coefficients
    T(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;

    % If the step is disabled the points below the step are blocked out
    if onlyChannel && i <= N_wide*M_wide
        T(i,i) = T(i,i) + 10e+30;
    end %if

    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;

end %for
p_corr = T\beta';                                     % Matrix inversion
```

E.5.1.5 plot_BFS.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Surface plots for velocities, pressure and pressure correction      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
showOutletView = true;% Save profiles seen from outlet in addition to inlet
az = 37.5; % Viewpoints when seen from outlet
el = 30;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Settings
% filler is a value that is filled in where the step is. showStep can be
% adjusted if it is desirable to plot the profiles with zero at the step.
if ~exist('showStep','var')
    filler = Inf;
else
    if showStep == true
        filler = 0;
    else
        filler = Inf;
    end %if
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Velocities to matrices
% u-velocity

u_fullplot = zeros(M_total+2, N_total+1);
u_fullplot(M_wide+2:end-1,1) = u_in;
u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
u_fullplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
u_fullplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_fullplot = u_fullplot*u_in_true;

% Create a mesh for the plotting
[xu_plot,yu_plot] = meshgrid(x_0:del_x_true:x_N,...
    [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);

% y-points are adjusted at the inlet because the southern wall of the
% narrow channel does not align with the u-velocity nodes in the wide sec.
jj = [linspace(0, H_total-h, M_total-M_narrow+1),...
    linspace(H_total-h+del_y_true/2, y_M-del_y_true/2, M_narrow), H_total];
for i = 1:N_narrow
    for j = 1:M_total
        % alter the points of the plot for the wall of the narrow section
```

```
        yu_plot(j,i) = jj(j);
    end %for
end %for

% v-velocity
v_fullplot = zeros(m_total+2, N_total+1);                    % Inlet is zero
v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
v_fullplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
v_fullplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_fullplot = v_fullplot*u_in_true;

% Create a mesh for the plotting
[xv_plot,yv_plot] = meshgrid(x_0:del_x_true:x_N, y_0:del_y_true:y_M);


f1 = figure;
f = surf(xu_plot,yu_plot,u_fullplot);
% set(f,'edgecolor','none')
s = sprintf('Plot of $u_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
set(f1, 'Position', [3.6667   40.3333  555.3333  284.6667]);
%[left bottom width height]
saveas(gcf,'unewBFS.png')
if showOutletView
    view(az,el)
    saveas(gcf,'unewoutletBFS.png')
    view(37.5,30) % back to normal
end % if


f2 = figure;
f = surf(xv_plot,yv_plot,v_fullplot);         % surf(x,y,z)
% set(f,'edgecolor','none')
s = sprintf('Plot of $v_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
set(f2, 'Position', [3.6667  327.0000  554.0000  314.0000]);
%[left bottom width height]
saveas(gcf,'vnewBFS.png')
if showOutletView
    view(az,el)
    saveas(gcf,'vnewoutletBFS.png')
    view(37.5,30) % back to normal
end % if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pressure to matrix

p_fullplot = zeros(M_total, N_total+1);
p_fullplot(1:M_wide,N_narrow+1:end-1) = ...
    global2matrix(p_new(1:N_wide*M_wide), N_wide, M_wide);
p_fullplot(M_wide+1:end,1:end-1) = ...
    global2matrix(p_new(N_wide*M_wide+1:end), N_total, M_narrow);
p_fullplot(1:M_wide, 1:N_narrow) = filler;
p_fullplot(:, end) = p_out;

% Transformation from dimensionless to regular
p_fullplot = p_fullplot*rho_true*u_in_true + p_atm;

% Create a mesh for the plotting
[xp_plot,yp_plot] = meshgrid(...
```

```
    x_0:del_x_true:x_N, ...
    y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);
xp_plot = xp_plot + del_x_true/2;
yp_plot = yp_plot + del_y_true/2;

f3 = figure;
f = surf(xp_plot,yp_plot,p_fullplot);
% set(f,'edgecolor','none')
s = sprintf('Plot of $p_{new}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
ztickformat('%.7f')
set(f3, 'Position', [ 721.6667   40.3333  560.0000  287.3333]);
%[left bottom width height]
saveas(gcf,'pnewBFS.png')
if showOutletView
    view(az,el)
    saveas(gcf,'pnewoutletBFS.png')
    view(37.5,30) % back to normal
end % if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pressure correction to matrix
if it > 0 % Don't plot in case of initial profiles (plotinitialprofiles)

    p_corrplot = zeros(M_total, N_total+1);
    p_corrplot(1:M_wide,N_narrow+1:end-1) = ...
        global2matrix(p_corr(1:N_wide*M_wide), N_wide, M_wide);
    p_corrplot(M_wide+1:end,1:end-1) = ...
        global2matrix(p_corr(N_wide*M_wide+1:end), N_total, M_narrow);
    p_corrplot(1:M_wide, 1:N_narrow) = filler;

    % Transformation from dimensionless to regular
    p_corrplot = p_corrplot*rho_true*u_in_true;

    % Create a mesh for the plotting
    [xp_plot,yp_plot] = meshgrid(...
        x_0:del_x_true:x_N, ...
        y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);

    f4 = figure;
    f = surf(xp_plot,yp_plot,p_corrplot);          % surf(x,y,z)
    % set(f,'edgecolor','none')
    s = sprintf('Plot of $p_{corr}$ after %d iterations', it );
%     f = title(s);
%     set(f, 'interpreter', 'latex', 'fontsize', 16)
    set(gca,'TickLabelInterpreter','latex')
    xlabel('$x$-direction [m]', 'interpreter', 'latex')
    ylabel('$y$-direction [m]', 'interpreter', 'latex')
        ztickformat('%.2f')
    zlabel('Pressure correction $p''$, [Pa]', 'interpreter', 'latex')
    set(f4, 'Position', [719.6667  329.6667  560.0000  311.3333]);
    saveas(gcf,'pcorrBFS.png')


    if showOutletView
        view(az,el)
        saveas(gcf,'pcorroutletBFS.png')
        view(37.5,30) % back to normal
    end % if
end %if
```

E.5.1.6 plotVelocityQuiver.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Velocity quiver plots                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filler = 0;    % For the quiver plots, the velocities at the step are set to
               % zero and not Inf, rectangles are therefore used to block
               % out the step from the plots afterwards.

% u-velocity
```

```
u_fullplot = zeros(M_total+2, N_total+1);
u_fullplot(M_wide+2:end-1,1) = u_in;
u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
u_fullplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
u_fullplot(1:M_wide, 1:N_narrow) = 0;

% Transformation from dimensionless to regular
u_fullplot = u_fullplot*u_in_true;


% v-velocity
v_fullplot = zeros(m_total+2, N_total+1);
v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
v_fullplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
v_fullplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_fullplot = v_fullplot*u_in_true;


uSN = zeros(M_total, N_total);
vSN = zeros(M_total, N_total);
for i = 2:N_total+1
    for j = 1:M_total
        uSN(j,i-1) = 1/2*(u_fullplot(j+1,i-1) + u_fullplot(j+1,i));
    end %for
end %for
for j = 2:M_total+1
    for i = 1:N_total
        vSN(j-1,i) = 1/2*(v_fullplot(j-1,i) + v_fullplot(j,i));
    end %for
end %for

% Need to make a combined velocitiy vector
combvel = sqrt(uSN.^2 + vSN.^2);

% Create a mesh for the plotting
[xSN,ySN] = meshgrid(...
    x_0:del_x:x_N-del_x, ...
    y_0+del_y/2:del_y:y_M-del_y/2);


fq1 = figure;
qn = quiver( xSN, ySN , uSN , vSN,'LineWidth',0.5,'Color','k');

%Block out the step
r = rectangle('Position',[0.03 -0.05 3 0.55]);
r.FaceColor = [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;

s = rectangle('Position',[0.03 -0.05 22 0.05]);
s.FaceColor = [1 1 1];
s.EdgeColor = 'none';%'k';
s.LineWidth = .0000010;

t = rectangle('Position',[0.03 1.5 22 0.05]);
t.FaceColor = [1 1 1];
t.EdgeColor = 'none';%'k';
t.LineWidth = .0000010;

hold on
set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
    'Marker','o','MarkerSize',1,'ShowArrowHead','on')
s = sprintf('Plot of velocities as vectors after %d iterations', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([0,22])
```

```
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([-0.05,1.55])
ytickformat('%.1f')
set(fq1,'Position', [3   250   717   420]);
saveas(gcf,'velocityquiver.png')
ax.Layer = 'top';


fq2 = figure;
qn = quiver(...
    xSN, ySN , uSN , vSN,...%u_fullplot(1:end-1,:)
    'LineWidth',0.5,'Color','k');

    r = rectangle('Position',[0.03 -0.05 3 0.55]);
r.FaceColor = [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;

s = rectangle('Position',[0.03 -0.05 22 0.05]);
s.FaceColor = [1 1 1];
s.EdgeColor = 'none';%'k';
s.LineWidth = .0000010;

hold on
set(qn,'AutoScale','on', 'AutoScaleFactor', 1.5,'Marker','o',...
    'MarkerSize',1,'MaxHeadSize',0.01);%'ShowArrowHead','off')
% qw = quiver(...
%     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
%         'LineWidth',0.5,'Color','k');
s = sprintf(...
    'Plot of velocities as vectors after %d iterations scales x 1.5', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([l-l/4,l*3])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([0,H+H/4])
ytickformat('%.1f')
set(fq2,'Position', [724   250   560   420]);
saveas(gcf,'velocityquiver_zoomed.png')
ax.Layer = 'top';


fq3 = figure;
qn = quiver(...
    xSN(1:M_wide,N_narrow+1:N_narrow*2), ...
    ySN(1:M_wide,N_narrow+1:N_narrow*2) ,...
    uSN(1:M_wide,N_narrow+1:N_narrow*2) , ...
    vSN(1:M_wide,N_narrow+1:N_narrow*2),...%u_fullplot(1:end-1,:)
    'LineWidth',0.5,'Color','k');

r = rectangle('Position',[0.03 -0.05 3 0.55]);
r.FaceColor = [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;

s = rectangle('Position',[0.03 -0.05 22 0.05]);
s.FaceColor = [1 1 1];
s.EdgeColor = 'none';%'k';
s.LineWidth = .0000010;

hold on
set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
    'Marker','o','MarkerSize',1,'ShowArrowHead','on')
% qw = quiver(...
%     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
%         'LineWidth',0.5,'Color','k');
s = sprintf(...
    'Plot of velocities as vectors after %d iterations, scaled * 2', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
```

```
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([l,2*l])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([0,H])
ytickformat('%.1f')
set(fq3,'Position', [724   250   560   420]);
saveas(gcf,'velocityquiver_zoomed.png')
ax.Layer = 'top';
```

E.5.1.7 plotColoredQuiver.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Colored velocity quiver plots                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filler = 0;   % For the quiver plots, the velocities at the step are set to
              % zero and not Inf, rectangles are therefore used to block
              % out the step from the plots afterwards.
levels = 50;            % Number of different colors for the representation
showvals = false;                           % Show the value of each color
lines = 'none';                           % Show lines in between each color

% u-velocity
u_fullplot = zeros(M_total+2, N_total+1);
u_fullplot(M_wide+2:end-1,1) = u_in;
u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
u_fullplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
u_fullplot(1:M_wide, 1:N_narrow) = 0;

% Transformation from dimensionless to regular
u_fullplot = u_fullplot*u_in_true;


% v-velocity
v_fullplot = zeros(m_total+2, N_total+1);
v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
v_fullplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
v_fullplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_fullplot = v_fullplot*u_in_true;


uSN = zeros(M_total, N_total);
vSN = zeros(M_total, N_total);
for i = 2:N_total+1
    for j = 1:M_total
        uSN(j,i-1) = 1/2*(u_fullplot(j+1,i-1) + u_fullplot(j+1,i));
    end %for
end %for
for j = 2:M_total+1
    for i = 1:N_total
        vSN(j-1,i) = 1/2*(v_fullplot(j-1,i) + v_fullplot(j,i));
    end %for
end %for


% Need to make a combined velocitiy vector
combvel = sqrt(uSN.^2 + vSN.^2);



% Create a mesh for the plotting
[xSN,ySN] = meshgrid(...
    x_0+ del_x_true/2:del_x_true:x_N-del_x_true/2, ...
    y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);

combvelwall = [zeros(1,N_total); combvel ; zeros(1,N_total)];

fq1 = figure;
% Contour plot
[M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
```

```
    [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
    combvelwall,levels);
c.LineColor = lines;
hold on
qn = quiver( xSN, ySN , uSN , vSN,'LineWidth',0.5,'Color','k');

%Block out the step
r = rectangle('Position',[0.03 -0.05 3 0.55]);
r.FaceColor = [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;


hold on
set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
    'Marker','o','MarkerSize',1,'ShowArrowHead','on')
s = sprintf('Plot of velocities as vectors after %d iterations', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([0,22])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([-0.05,1.55])
ytickformat('%.1f')
set(fq1,'Position', [3   250   717   420]);
saveas(gcf,'velocityquiver.png')
ax.Layer = 'top';


fq2 = figure;
[M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
    [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
    combvelwall,levels);
c.LineColor = lines;
hold on
qn = quiver(...
    xSN, ySN , uSN , vSN,...%u_fullplot(1:end-1,:)
    'LineWidth',0.5,'Color','k');

r = rectangle('Position',[0.03 -0.05 3 0.55]);
r.FaceColor = [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;


hold on
set(qn,'AutoScale','on', 'AutoScaleFactor', 2.1,'Marker','o',...
    'MarkerSize',1,'MaxHeadSize',0.01);%'ShowArrowHead','off')
% qw = quiver(...
%     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
%         'LineWidth',0.5,'Color','k');
s = sprintf(...
    'Plot of velocities as vectors after %d iterations scales x 1.5', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([l-l/4,l*3])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([0,H+H/4])
ytickformat('%.1f')
set(fq2,'Position', [724   250   560   420]);
saveas(gcf,'velocityquiver1zoomed.png')
ax.Layer = 'top';


fq3 = figure;
[M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
    [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
    combvelwall,levels);
c.LineColor = lines;
```

```
hold on
qn = quiver(...
    xSN(1:M_wide,N_narrow+1:N_narrow*2), ...
    ySN(1:M_wide,N_narrow+1:N_narrow*2) ,...
    uSN(1:M_wide,N_narrow+1:N_narrow*2) , ...
    vSN(1:M_wide,N_narrow+1:N_narrow*2),...%u_fullplot(1:end-1,:)
    'LineWidth',0.5,'Color','k');

r = rectangle('Position',[0.03 -0.05 3 0.55]);
r.FaceColor = [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;

hold on
set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 2.1,...
    'Marker','o','MarkerSize',1,'ShowArrowHead','on')
% qw = quiver(...
%     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
%         'LineWidth',0.5,'Color','k');
s = sprintf(...
    'Plot of velocities as vectors after %d iterations, scaled * 2', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([l,2*l])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([0,H])
ytickformat('%.1f')
set(fq3,'Position', [724   250   560   420]);
saveas(gcf,'velocityquiver2zoomed.png')
ax.Layer = 'top';
```

E.5.1.8 plotVelocityCorrection.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    Surface plots for velocity corrections                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Settings
% filler is a value that is filled in where the step is. showStep can be
% adjusted if it is desirable to plot the profiles with zero at the step.
if ~exist('showStep','var')
    filler = Inf;
else
    if showStep == true
        filler = 0;
    else
        filler = Inf;
    end %if
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Velocities to matrices
% u-velocity

u_fullplot = zeros(M_total+2, N_total+1);
u_fullplot(M_wide+2:end-1,1) = 0;
u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_corr(1:N_wide*M_wide), N_wide, M_wide);
u_fullplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_corr(N_wide*M_wide+1:end), N_total, M_narrow);
u_fullplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_fullplot = u_fullplot*u_in_true;

% Create a mesh for the plotting
[xu_plot,yu_plot] = meshgrid(...
    x_0:del_x_true:x_N, ...
    [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);

% y-points are adjusted at the inlet because the southern wall of the
% narrow channel does not align with the u-velocity nodes in the wide sec.
for i = 1:N_narrow
```

```
    % alter the points of the plot for the wall of the narrow section
    yu_plot(:,i) = [linspace(0, H_total-h, M_total-M_narrow+1), ...
        linspace(H_total-h+del_y_true/2, y_M-del_y_true/2, M_narrow), ...
        H_total];
end %for


% v-velocity
v_fullplot = zeros(m_total+2, N_total+1);                    % Inlet is zero
v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_corr(1:N_wide*m_wide), N_wide, m_wide);
v_fullplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_corr(N_wide*m_wide+1:end), N_total, m_narrow);
v_fullplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_fullplot = v_fullplot*u_in_true;

% Create a mesh for the plotting
[xv_plot,yv_plot] = meshgrid(...
    x_0:del_x_true:x_N, ...
    y_0:del_y_true:y_M);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Plot

f1 = figure;
f = surf(xu_plot,yu_plot,u_fullplot);          % surf(x,y,z)
% set(f,'edgecolor','none')
s = sprintf('Plot of $u_{corr}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
set(f1, 'Position', [3.6667   40.3333  555.3333  284.6667]); %[left bottom width height]
saveas(gcf,'ucorrBFS.png')

f2 = figure;
f = surf(xv_plot,yv_plot,v_fullplot);          % surf(x,y,z)
% set(f,'edgecolor','none')
s = sprintf('Plot of $v_{corr}$ after %d iterations', it );
% f = title(s);
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
set(f2, 'Position', [3.6667  327.0000  554.0000  314.0000]); %[left bottom width height]
saveas(gcf,'vcorrBFS.png')
```

E.5.1.9 plotIntermediates.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               Surface plots for the intermediate velocities               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
showOutletView = true;% Save profiles seen from outlet in addition to inlet
az = 37.5; % Viewpoints when seen from outlet
el = 30;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Settings
% filler is a value that is filled in where the step is. showStep can be
% adjusted if it is desirable to plot the profiles with zero at the step.
if ~exist('showStep','var')
    filler = Inf;
else
    if showStep == true
        filler = 0;
    else
        filler = Inf;
    end %if
```

```matlab
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Velocities to matrices
% u-velocity

u_fullplot = zeros(M_total+2, N_total+1);
u_fullplot(M_wide+2:end-1,1) = u_in;
u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_star(1:N_wide*M_wide), N_wide, M_wide);
u_fullplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_star(N_wide*M_wide+1:end), N_total, M_narrow);
u_fullplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_fullplot = u_fullplot*u_in_true;

% Create a mesh for the plotting
[xu_plot,yu_plot] = meshgrid(x_0:del_x_true:x_N,...
    [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);

% y-points are adjusted at the inlet because the southern wall of the
% narrow channel does not align with the u-velocity nodes in the wide sec.
jj = [linspace(0, H_total-h, M_total-M_narrow+1), linspace(H_total-h+del_y_true/2, y_M-del_y_true/2, M_narrow), H_total];
for i = 1:N_narrow
    for j = 1:M_total
        % alter the points of the plot for the wall of the narrow section
        yu_plot(j,i) = jj(j);
    end %for
end %for

% v-velocity
v_fullplot = zeros(m_total+2, N_total+1);                    % Inlet is zero
v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_star(1:N_wide*m_wide), N_wide, m_wide);
v_fullplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_star(N_wide*m_wide+1:end), N_total, m_narrow);
v_fullplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_fullplot = v_fullplot*u_in_true;

% Create a mesh for the plotting
[xv_plot,yv_plot] = meshgrid(x_0:del_x_true:x_N, y_0:del_y_true:y_M);


f1 = figure;
f = surf(xu_plot,yu_plot,u_fullplot);          % surf(x,y,z)
% set(f,'edgecolor','none')
s = sprintf('Plot of $u_{star}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
set(f1, 'Position', [3.6667   40.3333  555.3333  284.6667]); %[left bottom width height]
saveas(gcf,'ustarBFS.png')
if showOutletView
    view(az,el)
    saveas(gcf,'ustaroutletBFS.png')
    view(37.5,30) % back to normal
end % if

f2 = figure;
f = surf(xv_plot,yv_plot,v_fullplot);          % surf(x,y,z)
% set(f,'edgecolor','none')
s = sprintf('Plot of $v_{star}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
```

```
zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
set(f2, 'Position', [3.6667  327.0000  554.0000  314.0000]); %[left bottom width height]
saveas(gcf,'vstarBFS.png')
if showOutletView
    view(az,el)
    saveas(gcf,'vstaroutletBFS.png')
    view(37.5,30) % back to normal
end % if
```

E.5.1.10 plot_BFS_iterations.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Plots for velocities, pressure and pressure correction            %
%                  saved for each specified iteration                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Settings
% filler is a value that is filled in where the step is. showStep can be
% adjusted if it is desirable to plot the profiles with zero at the step.
if ~exist('showStep','var')
    filler = Inf;
else
    if showStep == true
        filler = 0;
    else
        filler = Inf;
    end %if
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Velocities to matrices
% u-velocity

u_fullplot = zeros(M_total+2, N_total+1);
u_fullplot(M_wide+2:end-1,1) = u_in;
u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
u_fullplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
u_fullplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_fullplot = u_fullplot*u_in_true;

% Create a mesh for the plotting
[xu_plot,yu_plot] = meshgrid(x_0:del_x_true:x_N,...
    [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);

% y-points are adjusted at the inlet because the southern wall of the
% narrow channel does not align with the u-velocity nodes in the wide sec.
jj = [linspace(0, H_total-h, M_total-M_narrow+1), ...
    linspace(H_total-h+del_y_true/2, y_M-del_y_true/2, M_narrow), H_total];
for i = 1:N_narrow
    for j = 1:M_total
        % alter the points of the plot for the wall of the narrow section
        yu_plot(j,i) = jj(j);
    end %for
end %for

% v-velocity
v_fullplot = zeros(m_total+2, N_total+1);                    % Inlet is zero
v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
v_fullplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
v_fullplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_fullplot = v_fullplot*u_in_true;

% Create a mesh for the plotting
[xv_plot,yv_plot] = meshgrid(x_0:del_x_true:x_N, y_0:del_y_true:y_M);


%%  Plot velocities
```

```
f1 = figure('units','normalized','outerposition',[0 0 1 1]);


subplot(2,2,1);
f = surf(xu_plot,yu_plot,u_fullplot);          % surf(x,y,z)
s = sprintf('Plot of $u_{new}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')

subplot(2,2,3);
f = surf(xv_plot,yv_plot,v_fullplot);          % surf(x,y,z)
s = sprintf('Plot of $v_{new}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pressure

p_fullplot = zeros(M_total, N_total+1);
p_fullplot(1:M_wide,N_narrow+1:end-1) = ...
    global2matrix(p_new(1:N_wide*M_wide), N_wide, M_wide);
p_fullplot(M_wide+1:end,1:end-1) = ...
    global2matrix(p_new(N_wide*M_wide+1:end), N_total, M_narrow);
p_fullplot(1:M_wide, 1:N_narrow) = filler;
p_fullplot(:, end) = p_out;

% Transformation from dimensionless to regular
p_fullplot = p_fullplot*rho_true*u_in_true + p_atm;

% Create a mesh for the plotting
[xp_plot,yp_plot] = meshgrid(...
    x_0:del_x_true:x_N, ...
    y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);

subplot(2,2,2);
f = surf(xp_plot,yp_plot,p_fullplot);          % surf(x,y,z)
s = sprintf('Plot of $p_{new}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
ztickformat('%.7f')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Pressure correction to matrix

p_corrplot = zeros(M_total, N_total+1);
p_corrplot(1:M_wide,N_narrow+1:end-1) = ...
    global2matrix(p_corr(1:N_wide*M_wide), N_wide, M_wide);
p_corrplot(M_wide+1:end,1:end-1) = ...
    global2matrix(p_corr(N_wide*M_wide+1:end), N_total, M_narrow);
p_corrplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
p_corrplot = p_corrplot*rho_true*u_in_true;

% Create a mesh for the plotting
[xp_plot,yp_plot] = meshgrid(...
    x_0:del_x_true:x_N, ...
    y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);


subplot(2,2,4);
f = surf(xp_plot,yp_plot,p_corrplot);          % surf(x,y,z)
s = sprintf('Plot of $p_{corr}$ after %d iterations', it );
```

```
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure correction $p''$, [Pa]', 'interpreter', 'latex')
ztickformat('%.2f')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Make .gif

axis tight manual % this ensures that getframe() returns a consistent size
filename = 'itdev_allfour.gif';
% Capture the plot as an image
frame = getframe(f1);
im = frame2im(frame);
[imind,cm] = rgb2ind(im,256);
% Write to the GIF File
if (it == 1 && plotInitialProfiles == false) || ...
        (it == 0 && plotInitialProfiles == true)
  imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
else
  imwrite(imind,cm,filename,'gif','WriteMode','append');
end

close all
```

E.5.1.11 plotVelInts_BFS_iterations.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 Plots for velocity intermediates and                      %
%                  saved for each specified iteration                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Settings
% filler is a value that is filled in where the step is. showStep can be
% adjusted if it is desirable to plot the profiles with zero at the step.
if ~exist('showStep','var')
    filler = Inf;
else
    if showStep == true
        filler = 0;
    else
        filler = Inf;
    end %if
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% u-velocity
u_circplot = zeros(M_total+2, N_total+1);
u_circplot(M_wide+2:end-1,1) = u_in;
u_circplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_circ(1:N_wide*M_wide), N_wide, M_wide);
u_circplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_circ(N_wide*M_wide+1:end), N_total, M_narrow);
u_circplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_circplot = u_circplot*u_in_true;


u_starplot = zeros(M_total+2, N_total+1);
u_starplot(M_wide+2:end-1,1) = u_in;
u_starplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_star(1:N_wide*M_wide), N_wide, M_wide);
u_starplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_star(N_wide*M_wide+1:end), N_total, M_narrow);
u_starplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_starplot = u_starplot*u_in_true;


u_corrplot = zeros(M_total+2, N_total+1);
u_corrplot(M_wide+2:end-1,1) = 0; % no correction at known
u_corrplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_corr(1:N_wide*M_wide), N_wide, M_wide);
```

```
u_corrplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_corr(N_wide*M_wide+1:end), N_total, M_narrow);
u_corrplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_corrplot = u_corrplot*u_in_true;


u_newplot = zeros(M_total+2, N_total+1);
u_newplot(M_wide+2:end-1,1) = u_in;
u_newplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
u_newplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
u_newplot(1:M_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
u_newplot = u_newplot*u_in_true;


% Create a mesh for the plotting
[xu_plot,yu_plot] = meshgrid(...
    x_0:del_x_true:x_N, ...
    [0, y_0+del_y_true/2:del_y_true:y_M-del_y_true/2, H_total]);

for i = 1:N_narrow % alter the points of the plot for the wall of the narrow section
    yu_plot(:,i) = [linspace(0, H_total-h, M_total-M_narrow+1), linspace(H_total-h+
        del_y_true/2, y_M-del_y_true/2, M_narrow), H_total];
%     yu_plot(i,:) = [linspace(0, H_total-h, M-M+1), linspace(H_total-h+del_y_true/2,
    y_M-del_y_true/2, M), H_total];
end %for


f0 = figure('units','normalized','outerposition',[0 0 1 1]);


subplot(2,2,1);
f = surf(xu_plot,yu_plot,u_circplot);           % surf(x,y,z)

s = sprintf('Plot of $u_{circ}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')


subplot(2,2,3);
f = surf(xu_plot,yu_plot,u_starplot);           % surf(x,y,z)
s = sprintf('Plot of $u_{star}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $v$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')


subplot(2,2,2);
f = surf(xu_plot,yu_plot,u_corrplot);           % surf(x,y,z)
s = sprintf('Plot of $u_{corr}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')


subplot(2,2,4);
f = surf(xu_plot,yu_plot,u_newplot);           % surf(x,y,z)
s = sprintf('Plot of $u_{new}$ after %d iterations', it );
```

```
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')



axis tight manual % this ensures that getframe() returns a consistent size
filename = 'itdev_uintermediates.gif';
% Capture the plot as an image
frame = getframe(f0);
im = frame2im(frame);
[imind,cm] = rgb2ind(im,256);
% Write to the GIF File
if it == 1
  imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
else
  imwrite(imind,cm,filename,'gif','WriteMode','append');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% v-velocity
v_circplot = zeros(m_total+2, N_total+1);
v_circplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_circ(1:N_wide*m_wide), N_wide, m_wide);
v_circplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_circ(N_wide*m_wide+1:end), N_total, m_narrow);
v_circplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_circplot = v_circplot*u_in_true;


v_starplot = zeros(m_total+2, N_total+1);
v_starplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_star(1:N_wide*m_wide), N_wide, m_wide);
v_starplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_star(N_wide*m_wide+1:end), N_total, m_narrow);
v_starplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_starplot = v_starplot*u_in_true;


v_corrplot = zeros(m_total+2, N_total+1);
v_corrplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_corr(1:N_wide*m_wide), N_wide, m_wide);
v_corrplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_corr(N_wide*m_wide+1:end), N_total, m_narrow);
v_corrplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_corrplot = v_corrplot*u_in_true;


v_newplot = zeros(m_total+2, N_total+1);
v_newplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
v_newplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
v_newplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_newplot = v_newplot*u_in_true;


% Create a mesh for the plotting
[xv_plot,yv_plot] = meshgrid(...
    x_0:del_x_true:x_N, ...
    y_0:del_y_true:y_M);


f2 = figure('units','normalized','outerposition',[0 0 1 1]);
```

```
subplot(2,2,1);
f = surf(xv_plot,yv_plot,v_circplot);          % surf(x,y,z)
s = sprintf('Plot of $v_{circ}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity $u$, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')


subplot(2,2,3);
f = surf(xv_plot,yv_plot,v_starplot);          % surf(x,y,z)
s = sprintf('Plot of $v_{star}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')


subplot(2,2,2);
f = surf(xv_plot,yv_plot,v_corrplot);          % surf(x,y,z)
s = sprintf('Plot of $v_{corr}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Velocity, [m/s]', 'interpreter', 'latex')
ztickformat('%.2f')


subplot(2,2,4);
f = surf(xv_plot,yv_plot,v_newplot);
s = sprintf('Plot of $v_{new}$ after %d iterations', it );
f = title(s);
set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
zlabel('Pressure $p$, [Pa]', 'interpreter', 'latex')
ztickformat('%.2f')


axis tight manual % this ensures that getframe() returns a consistent size
filename = 'itdev_vintermediates.gif';
% Capture the plot as an image
frame = getframe(f2);
im = frame2im(frame);
[imind,cm] = rgb2ind(im,256);
% Write to the GIF File
if it == 1
  imwrite(imind,cm,filename,'gif', 'Loopcount',inf);
else
  imwrite(imind,cm,filename,'gif','WriteMode','append');
end
close all
```

E.5.1.12 isWide.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%         Function checkin if a node is in the wide section or not          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function res = isWide(a, N_narrow, N_wide, M_wide)
    res = ones(1,length(a))*false;
    for j = 1:length(a)
        i = a(j);
        rownumber = getRowNumber(i, N_wide, M_wide, N_narrow + N_wide);
        if rownumber <= M_wide || i - M_wide*N_wide - ...
                (rownumber-M_wide-1)*(N_narrow + N_wide) > N_narrow
```

```
            res(j) = true;
        end %if
    end %for
end %function
```

E.5.1.13 getRowNumber.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                Function giving the row number of a node                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% getRowNumber.m returns the row number of an arbitrary computational point
% in the domaindefined by N and M in the main BFC_globaldomain_spring.m
function rownumber = getRowNumber(a, N_wide, M_wide, N_total)
rownumber = zeros(length(a),1);
    for j = 1:length(a)
        i = a(j);
        if i <= N_wide*M_wide
            rownumber(j) = floor((N_wide+i-1)/N_wide);
        elseif i > N_wide*M_wide
            rownumber(j) = M_wide + floor((i-N_wide*M_wide-1)/N_total)+1;
        end %if
    end %for
end %function
```

E.5.1.14 getRowUnder.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Function giving the row number of a node below itself          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% getRowUnder.m returns the index of the point directly below itself.
function index = getRowUnder(i, N_wide, M_wide, N_total)
    index = zeros(1, length(i));
    for j = 1:length(i)
        if i(j) <= N_wide*M_wide
            index(j) = i(j)-N_wide;
        elseif i(j) > N_wide*M_wide
            index(j) = i(j)-N_total;
        end %if
    end %for
end %function
```

E.5.1.15 getRowOver.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Function giving the row number of a node above itself          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% getRowOver.m returns the index of the point directly above itself.
function index = getRowOver(i, N_wide, M_wide, N_total)
    index = zeros(1, length(i));
    for j = 1:length(i)
        if i(j) <= N_wide*(M_wide-1)
            index(j) = i(j)+N_wide;
        elseif i(j) > N_wide*(M_wide-1)
            index(j) = i(j)+N_total;
        end %if
    end %for
end %function
```

E.5.1.16 global2matrix.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Function transforming a globally indexed vector into a matrix      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [matrix] = global2matrix(glob, N, M)
    for j = 1:M % "down"                % the rest of the points are zero
        for i = 1:N % "left"
            matrix(j,i) = glob((j-1)*N + i);
        end % for
    end % for
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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Two dimensional fluid flow over a backwards facing step, dimensionless  %
%              Model adjusted to Reynolds number comparison                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear
clc
tic
warning on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Solver specs
maxits = 50000;   % Maximum number of iterations, stop if iterations exceed
% Choose which inlet profile to use
run('inletprofileRe400.m')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% System specifications
% Specify number of narrow points, leave the rest

N_narrow = 10;  % Number of scalar nodal points in narrow section in x-dir.
M_narrow = 10;  % Number of scalar nodal points in narrow section in x-dir.

l = 5;                                                % Narrow channel length
h = 1;                                                % Narrow channel height
L = 30;                                                 % Wide channel length
H = 1;                                                  % Wide channel height

L_total = l + L;                                       % Total channel length
H_total = h + H;                                       % Total channel height

x_0 = 0;                                 % Defining the domain using x and y
x_N = L_total;
y_0 = 0;
y_M = H_total;

N_wide = N_narrow*L/l;    % # scalar nodal points in wide section in x-dir.
M_wide = M_narrow*H/h;    % # scalar nodal points in wide section in y-dir.

% For extension to the wide channel the number of nodes in the narrow
% section needs to meet these criteria:
if floor(N_narrow)~= N_narrow || floor(N_wide)~= N_wide|| ...
        floor(M_narrow)~= M_narrow || floor(M_wide)~= M_wide
    msg = 'Points don''t match dimensions';
    error(msg)
end %if
N_total = N_narrow + N_wide;% Total # of scalar nodal points in x-direction
M_total = M_narrow + M_wide;% Total # of scalar nodal points in y-direction

m_total = M_total - 1;    % Total number of y-velocity nodes in y-direction
m_wide = M_wide;% Number of y-velocity nodes in y-direction in wide section
m_narrow = M_narrow - 1;% # of y-velocity nodes in y-dir. in narrow section

% Total number of computational points in the domain ...
totalpoints = N_narrow*M_narrow + N_wide*M_total;        %  ... for u and P
totalpoints_v = N_narrow*m_narrow + N_wide*m_total;             % ... for v
```

```
D_hyd = 2*h;                                           % Hydraulic diameter
mu_true = 8.90 * 10^-4;                                % Viscosity of water

del_z_true = 1;                                              % System depth
del_x_true = L_total/N_total;                        % Control volume width
del_y_true = H_total/M_total;                       % Control volume height
A_x_true = del_y_true*del_z_true;     % Cross-sectional area in x-direction
A_y_true = del_x_true*del_z_true;     % Cross-sectional area in y-direction

rho_true = 997;                                          % Density of water
u_in_true = u_bulk;                                      % Inlet u-velocity

g_x = 0;                                                   % No gravitation
g_y = 0;                                                   % No gravitation

Re = rho_true*D_hyd*u_in_true/mu_true;                    % Reynolds number

p_atm = 101325;                         % Atmospheric presssure at outlet
p_out_tilde = 0;                                        % Adjusted pressure
p_out = ones(1,M_total)*p_out_tilde;              % Outlet pressure profile

alpha_u = 0.01;                           % Under-relaxation factor for u
alpha_v = 0.01;                           % Under-relaxation factor for v
alpha_p = 0.02;                           % Under-relaxation factor for p
alpha_u = 0.005;                           % Under-relaxation factor for u
alpha_v = 0.005;                           % Under-relaxation factor for v
alpha_p = 0.01;                           % Under-relaxation factor for p

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Dimensionless parameters
mu = 1;                                            % Dimensionless viscosity
rho = 1;                                             % Dimensionless density
del_x = del_x_true/h;              % Dimensionless control volume width
del_y = del_y_true/h;             % Dimensionless control volume height
A_x = A_x_true/h^2; % Dimensionless cross-sectional area in x-direction
A_y = A_y_true/h^2; % Dimensionless cross-sectional area in y-direction
D_x = 2/Re*mu/del_x;   % Dimensionless diffusion conductance in x-direction
D_y = 2/Re*mu/del_y;   % Dimensionless diffusion conductance in y-direction
u_in = u_in/u_in_true;                                   % Inlet u-velocity
u_bulk_dimless = u_bulk/u_in_true;       % Bulk inlet velocity (which is 1)
v_in = 0;                                                % Inlet u-velocity
u_guess = u_max;                           % Initial guess for u-velocity
v_guess = 0.0;                             % Initial guess for v-velocity
p_guess = 0/(rho_true*u_in_true^2);          % Initial guess for pressure

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialisation of p
% Filling in initial pressure vector with the linear profile.
% This section is set up for if gravity is added, but could be more compact
% if the option to add gravity was not there.

p_circ_y_wide = linspace(p_guess, p_guess+rho*g_y*H_total,M_total);
p_circ_carthesian_wide = zeros(M_total,N_wide);
for j = 1:M_total
    for i = 1:N_wide
        p_circ_carthesian_wide(j,i) = p_circ_y_wide(j);
    end %for
end %for

p_circ_y_narrow = p_circ_y_wide(M_wide+1:end);
p_circ_carthesian_narrow = zeros(M_narrow,N_narrow);
for j = 1:M_narrow
    for i = 1:N_narrow
        p_circ_carthesian_narrow(j,i) = p_circ_y_narrow(j);
    end %for
end %for

filler = zeros(M_wide, N_narrow);
p_circ_carthesian = [[filler; p_circ_carthesian_narrow] ...
    p_circ_carthesian_wide ];
p_circ_carthesian = flip(p_circ_carthesian,1);

p_circ = p_circ_carthesian(1,:);                     % Take the first vector

for i = 2:M_total
```

```
    row = p_circ_carthesian(i);
    if i <= M_narrow                                        % Take whole row
        p_circ = [p_circ, p_circ_carthesian(i,:)];
    else                                              % Take part of the row
        p_circ = [p_circ, p_circ_carthesian(i,N_narrow+1:N_total)];
    end %if
end %for


%% Initialisation of u and v

u_circ = ones(totalpoints,1)*u_guess; % Fill in guess in the initial vector
for i = 1:totalpoints
    if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
        u_circ(i) = u_guess*(M_narrow/M_total);
    end %if
end %for


v_circ = ones(totalpoints_v,1)*v_guess; % Fill in guess in the initial vec.
for i = 1:totalpoints_v
    if isWide(i, N_narrow, N_wide, M_wide)% Lower guess after expansion
        v_circ(i) = v_guess*(m_narrow/m_total);
    end %if
end %for


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initialisation of solution vectors
p_new = zeros(1, totalpoints);                            % New pressure

u_corr = zeros(1, totalpoints);                   % u-velocity correction
u_new = zeros(1, totalpoints);                           % New u-velocity

v_corr = zeros(1, totalpoints_v);                 % v-velocity correction
v_new = zeros(1, totalpoints_v);                         % New v-velocity

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% While loop
conv = 0;                          % 0 is not converged, 1 when converged
it = 1;                                           % The current iteration

while  conv == 0
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%  Calculate velocities and pressure correction
    % Run the scripts:
    % Velocities
    BFS_u_velocity_parabolic
    BFS_v_velocity_parabolic

    % Pressure correction
    BFS_pressurecorrection_parabolic

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Velocity correction

    startCorr = 1;
    for j = startCorr:totalpoints
        if ( i <= N_wide*M_wide && mod(i, N_wide) == 0 ) ...   % Below step
             || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0)
            % Eastern boundary : eastern pressure is known, no press. corr.
            u_corr(j) =  - A_x/au(j)*(-p_corr(j));
        else
            u_corr(j) =  - A_x/au(j)*(p_corr(j+1)-p_corr(j));
        end % if
    end %for

    for k = startCorr:totalpoints_v
            v_corr(k) = - A_y/av(k)*...
                (p_corr(getRowOver(k, N_wide, M_wide, N_total))-p_corr(k));
    end %for

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Under-relaxation

    u_new = alpha_u*(u_star + u_corr') + (1-alpha_u)*u_circ;
```

```
    v_new = alpha_v*(v_star + v_corr') + (1-alpha_v)*v_circ;

    p_new = p_circ + alpha_p* p_corr';

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Check convergence
    % Make sure there are no mistakes in the matrix operations above
    if ~isvector(u_new) || ~isvector(p_new) || ~isvector(p_new)
        fprintf('u_new - %dx%d\n',size(u_new,1),size(u_new,2))
        fprintf('v_new - %dx%d\n',size(v_new,1),size(v_new,2))
        fprintf('p_new - %dx%d\n',size(p_new,1),size(p_new,2))
        error('Matrix addition gone wrong')
    end

    if isnan(rcond(U)) || isnan(rcond(V)) || isnan(rcond(T))
%          clc                              % Remove if warnings are desired
        fprintf('Stopped due to singularity in matrix\n')
        fprintf('RCOND u-velocity: %e \nRCOND v-velocity: %e \n',...
            rcond(U), rcond(V))
        fprintf('RCOND pressure correction: %e\n',rcond(T))
        fprintf('Problem occured after %d iterations\n', it)
        toc
        return
    end %if

    c1 = 1/u_bulk_dimless*sqrt((U*u_star-bu')'*(U*u_star-bu'));           % residuals
    c2 = 1/u_bulk_dimless*sqrt((V*v_star-bv')'*(V*v_star-bv'));           % residuals
    c3 = abs(sum(beta));                                  % continuity fulfulled
    c4 = 1/u_bulk_dimless*max(abs(u_circ - u_star)) ;    % change from last iteration
    c5 = 1/u_bulk_dimless*max(abs(v_circ - v_star))  ;   % change from last iteration

    c1_lim = 10^-8;                                                   % Limits
    c2_lim = 10^-8;
    c3_lim = 10^-10;
    c4_lim = 10^-8;
    c5_lim = 10^-8;

    c1_diff = c1-c1_lim;                         % How far away from convergence
    c2_diff = c2-c2_lim;
    c3_diff = c3-c3_lim;
    c4_diff = c4-c4_lim;
    c5_diff = c5-c5_lim;


   if  (c1 < c1_lim) && (c2 < c2_lim) && (c3 < c3_lim) && (c4 < c4_lim) ...
            && (c5 < c5_lim) || (it == maxits)
        conv = 1;                                                  % Converged
        if (it == maxits)
            fprintf('Stopped at max iterations (%d)\n',it);
        else
            fprintf('Solution converged after %d iterations\n',it);
        end %if

        fprintf('c1\tMomentum residual u\t\t%.2e\tLimit: %.2e\n',...
            c1,c1_lim);
        fprintf('c2\tMomentum residual v\t\t%.2e\tLimit: %.2e\n',...
            c2,c2_lim);
        fprintf('c3\tPressure correction\t\t%.2e\tLimit: %.2e\n',...
            c3,c3_lim);
        fprintf('c4\tDiff. last iteration u\t%.2e\tLimit: %.2e\n',...
            c4,c4_lim);
        fprintf('c5\tDiff. last iteration v\t%.2e\tLimit: %.2e\n',...
            c5,c5_lim);

        if max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c1_diff
            fprintf('Limiting criteria is c1\tMomentum residual u\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c2_diff
            fprintf('Limiting criteria is c2\tMomentum residual v\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c3_diff
            fprintf('Limiting criteria is c3\tPressure correction\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c4_diff
            fprintf('Limiting criteria is c4\tDiff. last iteration u\n')
        elseif max([c1_diff c2_diff c3_diff c4_diff c5_diff])== c5_diff
            fprintf('Limiting criteria is c5\tDiff. last iteration u\n')
        end %if
```

```

        plotColoredQuiver_parabolic

    else

        u_circ = u_new;                     % Not converged, updated variables
        v_circ = v_new;                     % Not converged, updated variables
        p_circ = p_new;                     % Not converged, updated variables

        it = it + 1;  % Update number of iterations
    end % if
end %while
toc
```

E.5.2.2 BFS_u_velocity_parabolic.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    u-velocity script for the BFS model                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

U = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
bu = zeros(1, totalpoints);          % Initialisation of source term vector

F_xe = zeros(1, totalpoints);    % Initialisation of convective mass fluxes
F_xw = zeros(1, totalpoints);
F_xn = zeros(1, totalpoints);
F_xs = zeros(1, totalpoints);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Generation of F_x, Convective mass fluxes


for i = 1:totalpoints

    etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )...    % below step
        || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
    wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
    ntest = totalpoints - N_total < i && i <= totalpoints  ;
    wwall = i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
    stest = (1 <= i && i <= N_wide) ...     % Excluding the corner value
        || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
    scorner = i == N_wide*M_wide + N_narrow;    % Only the corner value


    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
        F_xe(i) = rho/2*(u_circ(i)+ u_circ(i-1));
        F_xn(i) = 0;

        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xs(i) = rho/2*v_circ(i-N_total);

    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
        F_xe(i) = rho/2*(u_circ(i)+u_circ(i-1));
        F_xs(i) = 0;

        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*v_circ(i);

    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner
        F_xw(i) = rho/2*(...
            u_in(getRowNumber(i, N_wide, M_wide, N_total))+u_circ(i));
        F_xn(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xs(i) = rho/2*(v_circ(i-N_total) + v_circ(i-N_total+1));

    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner
        F_xw(i) = rho/2*(...
            u_in(getRowNumber(i, N_wide, M_wide, N_total))+u_circ(i));
        F_xs(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
```

```
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % Southwestern corner at step
    elseif ~etest && ~wtest && ~ntest && stest && wwall && ~scorner
        F_xw(i) = rho/2*(0 + u_circ(i));
        F_xs(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % At corner
    elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner
        F_xs(i) = rho/2*(0 + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)+1));
        F_xs(i)= 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % At eastern boundary (x = L)
    elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner
        F_xe(i) = rho/2*(u_circ(i-1)+u_circ(i));

        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*v_circ(i);
        F_xs(i) = rho/2*v_circ(getRowUnder(i, N_wide, M_wide, N_total));


    % At western boundary (x = 0)
    elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner
        F_xw(i) = rho/2*(...
            u_in(getRowNumber(i, N_wide, M_wide, N_total))+u_circ(i));

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
        F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );

     % At western wall at step
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner
        F_xw(i) = rho/2*(0+u_circ(i));

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
        F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );



    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~scorner
       F_xn(i) = 0;

       F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
       F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
       F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );


    % At southern boundary (y = 0)
    elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~scorner
        F_xs(i) = 0;

        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));

    % Not at any boundary
    else
        F_xe(i) = rho/2*(u_circ(i+1)+u_circ(i));
        F_xw(i) = rho/2*(u_circ(i-1)+u_circ(i));
        F_xn(i) = rho/2*(v_circ(i) + v_circ(i+1));
```

```
        F_xs(i) = rho/2*(...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)   ) +...
            v_circ( getRowUnder(i, N_wide, M_wide, N_total)+1 )  );


    end % if
    etest = false;
    wtest = false;
    wwall = false;
    ntest = false;
    stest = false;
    scorner = false;
end %for



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% u-velocity

for i = 1:totalpoints                              % Global indexing system

    etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 )...   % below step
        || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
    wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
    ntest = totalpoints - N_total < i && i <= totalpoints  ;
    wwall = i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
    stest = (1 <= i && i <= N_wide) ...    % Excluding the corner value
        || (N_wide*M_wide < i && i < N_wide*M_wide + N_narrow) ;
    scorner = i == N_wide*M_wide + N_narrow;    % Only the corner value



    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall && ~scorner

        bu(i) =  -(p_out(end)-p_circ(i))*A_x;

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff =  F_xe(i)*A_x;

        % At northern boundary
        nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y) + 2*D_y*A_y;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall && ~scorner

        bu(i) =  -(p_out(1)-p_circ(i))*A_x;

        % At eastern boundary (x = L)
        E_coeff = -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff =  F_xe(i)*A_x;

        % At southern boundary (y = 0)
        sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall && ~scorner
```

```

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x +(max(F_xw(i)*A_x,0)...
            + D_x*A_x)*u_in(getRowNumber(i, N_wide, M_wide, N_total));

        % At western boundary (x = 0)
        wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

        % At northern boundary
        nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x+(max(F_xw(i)*A_x,0) ...
            + D_x*A_x)*u_in(getRowNumber(i, N_wide, M_wide, N_total));

        % At western boundary (x = 0)
        wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

        % At southern boundary (y = 0)
        sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Southwestern corner at step
    elseif  ~etest && ~wtest && ~ntest && stest && wwall  && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x...
            +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;

        % At western boundary (x = 0)
        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;

        % At southern boundary (y = 0)
        S_coeff = -max(F_xs(i)*A_y,0) - 2*D_y*A_y;
        sP_coeff = -S_coeff -F_xs(i)*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % At corner
    elseif ~etest && ~wtest && ~ntest && ~stest && ~wwall && scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;

        % At southern boundary (y = 0)
        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;


        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;
```

```

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % At eastern boundary (x = L)
    elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~scorner

        bu(i) =  -(p_out(1)-p_circ(i))*A_x;

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff =  F_xe(i)*A_x;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western boundary (x = 0)
    elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x +(max(F_xw(i)*A_x,0) ...
            + D_x*A_x)*u_in(getRowNumber(i, N_wide, M_wide, N_total));

        % At western boundary (x = 0)
        wP_coeff = max(F_xw(i)*A_x,0) + D_x*A_x - F_xw(i)*A_x;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western wall
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x ...
            +(max(F_xw(i)*A_x,0) + D_x*A_x)*0;

        % At western boundary (x = 0)
        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

```

```

    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest && ~wwall  && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;

        % At northern boundary
        nP_coeff = F_xn(i)*A_y + max(0,-F_xn(i)*A_y)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At southern boundary (y = 0)
    elseif ~etest && ~wtest && ~ntest && stest && ~wwall  && ~scorner

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;

        % At southern boundary (y = 0)
        sP_coeff = -F_xs(i)*A_y +max(F_xs(i)*A_y,0)+ 2*D_y*A_y;

        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    %Not at any boundary
    else

        bu(i) = -(p_circ(i+1)-p_circ(i))*A_x;
        E_coeff =  -max(0,-F_xe(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_xe(i)*A_x;
        U(i, i+1) = E_coeff;

        W_coeff =  -max(F_xw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_xw(i)*A_x;
        U(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_xn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_xn(i)*A_y;
        U(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_xs(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_xs(i)*A_y;
        U(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    end % if

    % Filling in the rest of the matrix, adding all point coefficients
    U(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;

    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;

end %for
u_star = U\bu';                                         % Matrix inversion
```

E.5.2.3 BFS_v_velocity_parabolic.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       v-velocity script for the BFS model                   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V = zeros(totalpoints_v, totalpoints_v);  % Initialisation of coeff. matrix
bv = zeros(1, totalpoints_v);        % Initialisation of source term vector

F_ye = zeros(1, totalpoints_v);  % Initialisation of convective mass fluxes
F_yw = zeros(1, totalpoints_v);
F_yn = zeros(1, totalpoints_v);
F_ys = zeros(1, totalpoints_v);




%% Generation of F_y, Convective mass fluxes


for i = 1:totalpoints_v % Global indexing system

    etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ...  % below step
          || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0);
    wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
    ntest = totalpoints_v - N_total < i && i <= totalpoints_v  ;
    wwall = i <= N_wide*m_wide  && mod(i-1, N_wide) == 0; %
    stest = (1 <= i && i <= N_wide) ...    % Excluding the corner value
            || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow) ;
    wcorner = i == N_wide*(m_wide-1) + 1;       % Only the corner value



    % Northwestern corner
    if  wtest && ntest && ~stest && ~wwall && ~wcorner
        F_yw(i) = rho/2*(u_in(getRowNumber(i, N_wide, M_wide, N_total))...
            +u_in(getRowNumber(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

    % Southwestern corner at inlet
    elseif  wtest && ~ntest && stest && ~wwall && ~wcorner
        F_yw(i) = rho*u_in(getRowNumber(i, N_wide, M_wide, N_total));
        F_ys(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));


    % Southwestern corner at step
    elseif  ~wtest && ~ntest && stest && wwall && ~wcorner
        F_yw(i) = rho*0;
        F_ys(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));


    % At western boundary (x = 0)
    elseif  wtest && ~ntest && ~stest && ~wwall && ~wcorner
        F_yw(i) = rho*u_in(getRowNumber(i, N_wide, M_wide, N_total));

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));
```

```


    % At western wall
    elseif  ~wtest && ~ntest && ~stest && wwall && ~wcorner
        F_yw(i) = rho*0;

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));


    % At corner, right point from the corner
    elseif  ~wtest && ~ntest && ~stest && wwall && wcorner
        F_yw(i)= 0;

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

    % At northern boundary (y = h)
    elseif  ~wtest && ntest && ~stest && ~wwall && ~wcorner
        F_yn(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yw(i) = rho/2*(u_circ(i-1) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));


    % At southern boundary (y = 0)
    elseif ~wtest && ~ntest && stest && ~wwall && ~wcorner
        F_ys(i) = rho/2*v_circ(i);

        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yw(i) = rho/2*(u_circ(i-1) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));
        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));


    %Not at any boundary, including eastern boundary
    else
        F_ye(i) = rho/2*(u_circ(i) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_yw(i) = rho/2*(u_circ(i-1) + ...
            u_circ(getRowOver(i, N_wide, M_wide, N_total)-1));

        F_yn(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowOver(i, N_wide, M_wide, N_total)));
        F_ys(i) = rho/2*(v_circ(i) + ...
            v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

    end % if
    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;
    wcorner = false;

end % for

%% v-velocity

for i = 1:totalpoints_v                            % Global indexing system

    etest = ( i <= N_wide*m_wide && mod(i, N_wide) == 0 ) ...  % below step
```

```
           || ( i > N_wide*m_wide && mod(i-N_wide*m_wide, N_total) == 0);
    wtest = i > N_wide*m_wide && mod(i-1-N_wide*m_wide, N_total) == 0;
    ntest = totalpoints_v - N_total < i && i <= totalpoints_v  ;
    wwall = i <= N_wide*m_wide  && mod(i-1, N_wide) == 0; %
    stest = (1 <= i && i <= N_wide) ...    % Excluding the corner value
            || (N_wide*m_wide < i && i <= N_wide*m_wide + N_narrow) ;
    wcorner = i == N_wide*(m_wide-1) + 1;       % Only the corner value




    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = F_ye(i)*A_x;


        % At northern boundary
        nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall    && ~wcorner
        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At eastern boundary (x = L)
        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = F_ye(i)*A_x;

        % At southern boundary (y = 0),
        sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall && ~wcorner
        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))-...
                p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At western boundary (x = 0)
        wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

        % At northern boundary
        nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall && ~wcorner

```

```
            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y;

            % At western boundary (x = 0)
            wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

            % At southern boundary (y = 0),
            sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        % Southwestern corner at step
        elseif ~etest && ~wtest && ~ntest && stest && wwall    && ~wcorner

            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
                0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);

            % At western boundary (x = 0)
            W_coeff =  -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;

            % At southern boundary (y = 0),
            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        % At eastern boundary (x = L)
        elseif etest && ~wtest && ~ntest && ~stest && ~wwall && ~wcorner

            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y;

            % At eastern boundary (x = L)
            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = F_ye(i)*A_x;

            W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
            wP_coeff = -W_coeff - F_yw(i)*A_x;
            V(i, i-1) = W_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
            nP_coeff = -N_coeff + F_yn(i)*A_y;
            V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

            S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
            sP_coeff =  -S_coeff - F_ys(i)*A_y;
            V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

        % At western boundary (x = 0)
        elseif ~etest && wtest && ~ntest && ~stest && ~wwall && ~wcorner

            bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
                -p_circ(i))*A_y + rho*g_y*del_y*A_y;

            % At western boundary (x = 0)
            wP_coeff = - F_yw(i)*A_x + max(F_yw(i)*A_x,0) + 2*D_x*A_x;

            E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
            eP_coeff = -E_coeff + F_ye(i)*A_x;
            V(i, i+1) = E_coeff;

            N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
```

```
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    % At west wall (x = 0) [EXCLUDED CORNER]
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
            0*(-max(F_yw(i)*A_x,0) - 2*D_x*A_x);

        % At western boundary (x = 0)
        W_coeff =  -max(F_yw(i)*A_x,0) - 2*D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    % At corner
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall && wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y +...
            0*(-max(F_yw(i)*A_x,0) - D_x*A_x);

        % At western boundary (x = 0)
        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest && ~wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

       % At northern boundary
        nP_coeff = F_yn(i)*A_y + max(0, -F_yn(i)*A_y) + D_y*A_y ;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

```

```
    % At southern boundary (y = 0)
    elseif ~etest && ~wtest && ~ntest && stest && ~wwall && ~wcorner

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        % At southern boundary (y = 0),
        sP_coeff = -F_ys(i)*A_y + max(F_ys(i)*A_y,0) + D_y*A_y;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

    %Not at any boundary
    else

        bv(i) = -(p_circ(getRowOver(i, N_wide, M_wide, N_total))...
            -p_circ(i))*A_y + rho*g_y*del_y*A_y;

        E_coeff =  -max(0,-F_ye(i)*A_x) - D_x*A_x;
        eP_coeff = -E_coeff + F_ye(i)*A_x;
        V(i, i+1) = E_coeff;

        W_coeff =  -max(F_yw(i)*A_x,0) - D_x*A_x;
        wP_coeff = -W_coeff - F_yw(i)*A_x;
        V(i, i-1) = W_coeff;

        N_coeff = -max(0,-F_yn(i)*A_y) - D_y*A_y;
        nP_coeff = -N_coeff + F_yn(i)*A_y;
        V(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -max(F_ys(i)*A_y,0) - D_y*A_y;
        sP_coeff =  -S_coeff - F_ys(i)*A_y;
        V(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    end % if

    % Filling in the rest of the matrix, adding all point coefficients
    V(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;


    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;

end % for
v_star = V\bv';                                          % Matrix inversion
```

E.5.2.4 BFS_pressurecorrection_parabolic.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               Pressure correction script for the BFS model                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

T = zeros(totalpoints, totalpoints); % Initialisation of coefficient matrix
beta = zeros(1, totalpoints);        % Initialisation of source term vector

au = diag(U);           % a^center-coefficients from the momentum equations
av = diag(V);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Calculation
for i = 1:totalpoints                                  % Global indexing system

    etest = ( i <= N_wide*M_wide && mod(i, N_wide) == 0 ) ...   % below step
        || ( i > N_wide*M_wide && mod(i-N_wide*M_wide, N_total) == 0);
```

```
    ntest = totalpoints - N_total < i && i <= totalpoints  ;
    wtest = i > N_wide*M_wide && mod(i-1-N_wide*M_wide, N_total) == 0;
    wwall = i <= N_wide*M_wide  && mod(i-1, N_wide) == 0;
    stest = (1 <= i && i <= N_wide) ...    % Excluding the corner value
            || (N_wide*M_wide < i && i <= N_wide*M_wide + N_narrow) ;



    % Northeastern corner
    if etest && ~wtest && ntest && ~stest && ~wwall

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
            + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At eastern boundary (x = L)
        eP_coeff = rho*A_x^2/au(i);

        % At northern boundary (y = h) (y = H)
        nP_coeff = 0 ;

        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southeastern corner
    elseif etest && ~wtest && ~ntest && stest && ~wwall

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) ...
            -A_y*v_star(i));

        % At eastern boundary (x = L)
        eP_coeff = rho*A_x^2/au(i);

        % At southern boundary (y = 0)
        sP_coeff = 0;

        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Northwestern corner
    elseif ~etest && wtest && ntest && ~stest && ~wwall

        beta(i) =  rho*(-A_x*u_star(i) ...
            +A_x*u_in(getRowNumber(i, N_wide, M_wide, N_total))  ...
            + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At western boundary (x = 0)
        wP_coeff = 0;

        % At northern boundary (y = h) (y = H)
        nP_coeff = 0 ;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % Southwestern corner at inlet
    elseif ~etest && wtest && ~ntest && stest && ~wwall

        beta(i) =  rho*(-A_x*u_star(i) ...
```

```
            +A_x*u_in(getRowNumber(i, N_wide, M_wide, N_total)) ...
            -A_y*v_star(i));

        % At western boundary (x = 0)
        wP_coeff = 0;

        % At southern boundary (y = 0)
        sP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % Southwestern corner at step
    elseif  ~etest && ~wtest && ~ntest && stest && wwall

        beta(i) =  rho*(-A_x*u_circ(i)...
                    +A_x*0 -A_y*v_circ(i)); % wall/"inlet" velocity is zero

        % At western boundary (x = 0)
        wP_coeff = 0;

        % At southern boundary (y = 0)
        sP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;


    % At eastern boundary (x = L)
    elseif etest && ~wtest && ~ntest && ~stest && ~wwall

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
            -A_y*v_star(i) + ...
            A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At eastern boundary (x = L)
        eP_coeff = rho*A_x^2/au(i);


        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western boundary at inlet (x = 0)
    elseif ~etest && wtest && ~ntest && ~stest && ~wwall

        beta(i) =  rho*(-A_x*u_star(i) ...
            +A_x*u_in(getRowNumber(i, N_wide, M_wide, N_total)) ...
            -A_y*v_star(i) ...
            + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At western boundary (x = 0)
        wP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
```

```
        T(i, i+1) = E_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff =- rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At western wall
    elseif ~etest && ~wtest && ~ntest && ~stest && wwall

        beta(i) =  rho*(-A_x*u_circ(i)...
                    +A_x*0 -A_y*v_circ(i) +...
                    A_y*v_circ(getRowUnder(i, N_wide, M_wide, N_total)));

        % At western boundary (x = 0)
        wP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff =- rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest && ~wwall

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
            + A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        % At northern boundary (y = h)
        nP_coeff = 0 ;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;


    % At southern boundary (y = 0)
    elseif ~etest && ~wtest && ~ntest && stest && ~wwall

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1)...
            -A_y*v_star(i));

        % At southern boundary (y = 0)
        sP_coeff = 0;

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;
```

```


    %Not at any boundary
    else

        beta(i) = rho*(-A_x*u_star(i) +A_x*u_star(i-1) -A_y*v_star(i) + ...
            A_y*v_star(getRowUnder(i, N_wide, M_wide, N_total)));

        E_coeff = -rho*A_x^2/au(i);
        eP_coeff = -E_coeff ;
        T(i, i+1) = E_coeff;

        W_coeff =  -rho*A_x^2/au(i-1);
        wP_coeff = -W_coeff;
        T(i, i-1) = W_coeff;

        N_coeff = -rho*A_y^2/av(i);
        nP_coeff = -N_coeff;
        T(i, getRowOver(i, N_wide, M_wide, N_total)) = N_coeff;

        S_coeff = -rho*A_y^2/av(getRowUnder(i, N_wide, M_wide, N_total));
        sP_coeff =  -S_coeff;
        T(i, getRowUnder(i, N_wide, M_wide, N_total)) = S_coeff;

    end % if

    % Filling in the rest of the matrix, adding all point coefficients
    T(i,i) = wP_coeff + eP_coeff + nP_coeff + sP_coeff;


    etest = false;
    wtest = false;
    ntest = false;
    stest = false;
    wwall = false;

end %for
p_corr = T\beta';                                           % Matrix inversion
```

E.5.2.5 plotColoredQuiver_parabolic.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       Colored velocity quiver plots                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
filler = 0;    % For the quiver plots, the velocities at the step are set to
               % zero and not Inf, rectangles are therefore used to block
               % out the step from the plots afterwards.
levels = 50;            % Number of different colors for the representation
showvals = false;                             % Show the value of each color
lines = 'none';                           % Show lines in between each color

% u-velocity
u_fullplot = zeros(M_total+2, N_total+1);
u_fullplot(2:end-1,1) = u_in;
u_fullplot(2:M_wide+1,N_narrow+2:end) = ...
    global2matrix(u_new(1:N_wide*M_wide), N_wide, M_wide);
u_fullplot(M_wide+2:end-1,2:end) = ...
    global2matrix(u_new(N_wide*M_wide+1:end), N_total, M_narrow);
u_fullplot(1:M_wide, 1:N_narrow) = 0;

% Transformation from dimensionless to regular
u_fullplot = u_fullplot*u_in_true;


% v-velocity
v_fullplot = zeros(m_total+2, N_total+1);
v_fullplot(2:m_wide+1,N_narrow+2:end) = ...
    global2matrix(v_new(1:N_wide*m_wide), N_wide, m_wide);
v_fullplot(m_wide+2:end-1,2:end) = ...
    global2matrix(v_new(N_wide*m_wide+1:end), N_total, m_narrow);
v_fullplot(1:m_wide, 1:N_narrow) = filler;

% Transformation from dimensionless to regular
v_fullplot = v_fullplot*u_in_true;


```

```
uSN = zeros(M_total, N_total);
vSN = zeros(M_total, N_total);
for i = 2:N_total+1
    for j = 1:M_total
        uSN(j,i-1) = 1/2*(u_fullplot(j+1,i-1) + u_fullplot(j+1,i));
    end %for
end %for
for j = 2:M_total+1
    for i = 1:N_total
        vSN(j-1,i) = 1/2*(v_fullplot(j-1,i) + v_fullplot(j,i));
    end %for
end %for

% Need to make a combined velocitiy vector
combvel = sqrt(uSN.^2 + vSN.^2);

% Create a mesh for the plotting
[xSN,ySN] = meshgrid(...
    x_0+ del_x_true/2:del_x_true:x_N-del_x_true/2, ...
    y_0+del_y_true/2:del_y_true:y_M-del_y_true/2);

combvelwall = [zeros(1,N_total); combvel ; zeros(1,N_total)];


fq1 = figure;
% Contour plot
[M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
    [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
    combvelwall,levels);
c.LineColor = lines;
hold on
qn = quiver( xSN, ySN , uSN , vSN,'LineWidth',0.5,'Color','k');

%Block out the step
r = rectangle('Position',[0.03 0 1 1]);
r.FaceColor =  [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;

hold on
set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 0.7,...
    'Marker','o','MarkerSize',1,'ShowArrowHead','on')
s = sprintf('Plot of velocities as vectors after %d iterations', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([0,L_total])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ytickformat('%.1f')
set(fq1,'Position', [3   250   717   420]);
saveas(gcf,'velocityquiver.png')
ax.Layer = 'top';


fq2 = figure;
[M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
    [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
    combvelwall,levels);
c.LineColor = lines;
hold on
qn = quiver(...
    xSN, ySN , uSN , vSN,...%u_fullplot(1:end-1,:)
    'LineWidth',0.5,'Color','k');

r = rectangle('Position',[0.03 0 1 1]);
r.FaceColor =  [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;


hold on
set(qn,'AutoScale','on', 'AutoScaleFactor', 2.1,'Marker','o',...
    'MarkerSize',1,'MaxHeadSize',0.01);%'ShowArrowHead','off')
```

```
% qw = quiver(...
%     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
%         'LineWidth',0.5,'Color','k');
s = sprintf(...
    'Plot of velocities as vectors after %d iterations scales x 1.5', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([l-l/4,l*3])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([0,H+H/4])
ytickformat('%.1f')
set(fq2,'Position', [724   250   560   420]);
saveas(gcf,'velocityquiver1zoomed.png')
ax.Layer = 'top';


fq3 = figure;
[M,c] = contourf([xSN(1,:) ; xSN ;xSN(end,:)],...
    [ones(1,N_total)*y_0; ySN ; ones(1,N_total)*y_M], ...
    combvelwall,levels);
c.LineColor = lines;
hold on
qn = quiver(...
    xSN(1:M_wide,N_narrow+1:N_narrow*2), ...
    ySN(1:M_wide,N_narrow+1:N_narrow*2) ,...
    uSN(1:M_wide,N_narrow+1:N_narrow*2) , ...
    vSN(1:M_wide,N_narrow+1:N_narrow*2),...%u_fullplot(1:end-1,:)
    'LineWidth',0.5,'Color','k');

r = rectangle('Position',[0.03 0 1 1]);
r.FaceColor =  [1 1 1];
r.EdgeColor = 'none';%'k';
r.LineWidth = .0000010;


hold on
set(qn,'AutoScale','on', 'LineWidth',0.1,'AutoScaleFactor', 2.1,...
    'Marker','o','MarkerSize',1,'ShowArrowHead','on')
% qw = quiver(...
%     xv_plot, yv_plot , uplot(1:end-1,:), vplot,...
%         'LineWidth',0.5,'Color','k');
s = sprintf(...
    'Plot of velocities as vectors after %d iterations, scaled * 2', it );
% f = title(s);
ax = gca;
% set(f, 'interpreter', 'latex', 'fontsize', 16)
set(gca,'TickLabelInterpreter','latex')
ax.FontSize = 12;
xlabel('$x$-direction [m]', 'interpreter', 'latex')
xlim([l,2*l])
ylabel('$y$-direction [m]', 'interpreter', 'latex')
ylim([0,H])
ytickformat('%.1f')
set(fq3,'Position', [724   250   560   420]);
saveas(gcf,'velocityquiver2zoomed.png')
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

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                         Transfinite Interpolation                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clc
%% Settings
N = 71; % Number of points in q1/x-direction
M = 21; % Number of points in q2/y-direction

x_max = 35; % Total length of physical domain (including step)
y_max = 2; % Total height of physical domain (including step)

h = 1; % Height of the step
l = 5; % Length / width of the step

% Placement of points E and F splits the line segment AD in s equal pieces.
s = 3;


%% Boundary points
q1 = 0:N; % Specifying the q1-points with spacing of delta q1 = 1
q2 = 0:M; % Specifying the q1-points with spacing of delta q1 = 1

% Specifying the locations of points A-F in the physical domain
xA = 0;
xB = 0;
xC = x_max;
xD = x_max;
xE = l;
xF = l;

yA = h;
yB = y_max;
yC = y_max;
yD = 0;
yE = 0;
yF = h;

% Place points E and F to split the line segment AD in s equal pieces.
AFfrac = 1/s; % Fraction of total width of q1
AEfrac = 1/s; % Fraction of total width of q1
AFpoints = ceil(AFfrac * N); % Number of q1-points in line segment AF
FEpoints = floor(AEfrac * N); % Number of q1-points in line segment FE

q1AF = 0:AFpoints; % Vector of coordinates q1 for the line segment AF
q1FE = 0:FEpoints; % Vector of coordinates q1 for the line segment FE
q1ED = 0:(N-AFpoints-FEpoints); % Vector of coordinates q1 for ...
                                 % the line segment ED

% Calculation of the boundary points:
xAB = (1-q2/q2(end))* xA + q2/q2(end)*xB;
xBC = (1-q1/q1(end))* xB + q1/q1(end)*xC;
xDC = (1-q2/q2(end))* xD + q2/q2(end)*xC;
xED = (1-q1ED/q1ED(end))* xE + q1ED/q1ED(end)*xD;
xFE = (1-q1FE/q1FE(end))* xF + q1FE/q1FE(end)*xE;
xAF = (1-q1AF/q1AF(end))* xA + q1AF/q1AF(end)*xF;

yAB = (1-q2/q2(end))* yA + q2/q2(end)*yB;
yBC = (1-q1/q1(end))* yB + q1/q1(end)*yC;
yDC = (1-q2/q2(end))* yD + q2/q2(end)*yC;
yED = (1-q1ED/q1ED(end))* yE + q1ED/q1ED(end)*yD;
yFE = (1-q1FE/q1FE(end))* yF + q1FE/q1FE(end)*yE;
yAF = (1-q1AF/q1AF(end))* yA + q1AF/q1AF(end)*yF;

% Plot with the boundary points
```

```
% figure
% plot(xAB,yAB,'x',xBC,yBC,'x',xDC,yDC,'x',...
%     xED,yED,'x',xFE,yFE,'x',xAF,yAF,'x')
% % xlim([-0.1,1.1])
% % ylim([-0.1,1.1])
% legend({'$AB$','$BC$','$CD$','$DE$','$EF$','$FA$'},...
%     'Interpreter','latex','Location','best')

%% Center domain points
% Combining the x- and y-points for the line segments AF, FE and ED to ...
% one vector for AD. The points located exactly at F and E are ...
% overlapping and removed from xFE by taking xFE(2:end-1). Likewise for y.

xAD = [xAF xFE(2:end-1) xED];% Combining the x-points for the line segment
yAD = [yAF yFE(2:end-1) yED];% Combining the y-points for the line segment

% Initialising the matrix x of points in the physical domain
x = zeros(length(q2),length(q1));
% Initialising the matrix y of points in the physical domain
y = zeros(length(q2),length(q1));

% Calculating the center points
for j =1:length(q2)
    for i = 1:length(q1)
        x(j,i) = (1-q1(i)/q1(end))* xAB(j) +(q1(i)/q1(end)) *xDC(j)...
            +(1-q2(j)/q2(end))*xAD(i) +(q2(j)/q2(end))* xBC(i)...
            -(1-q1(i)/q1(end))* (1-q2(j)/q2(end))* xA...
            -(1-q1(i)/q1(end))* (q2(j)/q2(end))* xB...
            -(q1(i)/q1(end))*(1-q2(j)/q2(end))* xD...
            -(q1(i)/q1(end))*(q2(j)/q2(end))* xC;
        y(j,i) = (1-q1(i)/q1(end))* yAB(j) +(q1(i)/q1(end)) *yDC(j)...
            +(1-q2(j)/q2(end))*yAD(i) +(q2(j)/q2(end))* yBC(i)...
            -(1-q1(i)/q1(end))* (1-q2(j)/q2(end))* yA...
            -(1-q1(i)/q1(end))* (q2(j)/q2(end))* yB...
            -(q1(i)/q1(end))*(1-q2(j)/q2(end))* yD...
            -(q1(i)/q1(end))*(q2(j)/q2(end))* yC;
    end %for
end %for

% Plotting the resulting grid
figure
plot(x,y,'k',x',y','k')
xlim([xA,xD])
ylim([yD,yC])
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
saveas(gcf,'transfinite.png')
```

E.6.1.2 elliptic.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                        Elliptic grid generation                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clc
clear

maxits = 75;

P1 = 0;                                          % Poisson control function
P2 = 0;                                          % Poisson control function
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Create the algebraic grid for an initial guess
transfinite
N = length(q1);
M = length(q2);
n = N-2;             % dimensions of the inner point matrix to be solved for
m = M-2;                 % with the elliptic grid generation equations below
alpha = 0.001;

conv = 0;
it = 1;

while conv == 0

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Area Components
AM11 = zeros(m,n); % Indexed top bottom
AM12 = zeros(m,n); % A^1_2
AM21 = zeros(m,n); % A^2_1
AM22 = zeros(m,n);
for i = 2:N-1
    for j = 2:M-1
        AM11(j-1,i-1) = 1/2*(y(j+1,i) - y(j-1,i));
        AM21(j-1,i-1) = -1/2*(y(j,i+1) - y(j,i-1));
        AM12(j-1,i-1) = -1/2*(x(j+1,i) - x(j-1,i));
        AM22(j-1,i-1) = 1/2*(x(j,i+1) - x(j,i-1));
    end %for
end %for

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Jacobi Determinant
J2 = zeros(m,n);
for i = 2:N-1
    for j = 2:M-1
        J2(j-1,i-1) = (1/4*(x(j,i+1)-x(j,i-1))*(y(j+1,i)-y(j-1,i))...
                    - 1/4*(y(j,i+1)-y(j,i-1))*(x(j+1,i)-x(j-1,i)))^2;
    end %for
end %for

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Contravariant Tensor Components
gM11 = zeros(m,n);
gM12 = zeros(m,n);
gM21 = zeros(m,n);
gM22 = zeros(m,n);
for i = 1:n
    for j = 1:m
        gM11(j,i) = 1/J2(j,i)*...
            (AM11(j,i)*AM11(j,i) + AM12(j,i)*AM12(j,i));
        gM21(j,i) = 1/J2(j,i)*...
            (AM21(j,i)*AM11(j,i) + AM22(j,i)*AM12(j,i));
        gM12(j,i) = 1/J2(j,i)*...
            (AM11(j,i)*AM21(j,i) + AM12(j,i)*AM22(j,i));
        gM22(j,i) = 1/J2(j,i)*...
            (AM21(j,i)*AM21(j,i) + AM22(j,i)*AM22(j,i));
    end %for
end %for

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Matrices 2 Globals
A11 = matrix2global(AM11,n,m);
A12 = matrix2global(AM12,n,m);
A21 = matrix2global(AM21,n,m);
A22 = matrix2global(AM22,n,m);

g11 = matrix2global(gM11,n,m);
g12 = matrix2global(gM12,n,m);
g21 = matrix2global(gM21,n,m);
g22 = matrix2global(gM22,n,m);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% New x and y
X = zeros(n*m,n*m);
Y = zeros(n*m,n*m);

% The source term is zero and is updated if the point is at a boundary
bx = zeros(1,n*m);
by = zeros(1,n*m);


for i = 1:n*m

    etest = mod(i, n) == 0;
    ntest = n*m - n < i;
    wtest =  mod(i-1, n) == 0;
    stest = i <= n;

    % Northeastern corner
    if etest && ~wtest && ntest && ~stest

```

```
            X(i,i) =  -2*g11(i)-2*g22(i);
            % X(i,i+1)=(g11(i)+P1/2) ;
            bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
            X(i,i-1)=(g11(i)-P1/2) ;
            % X(i+n,i)=(g22(i)+P2/2) ;
            bx(i) = bx(i) - x(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
            X(i-n,i)=(g22(i)-P2/2) ;
            % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
            %  X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
            % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
            X(i-n,i-1)=(g12(i)/4+g21(i)/4);

            Y(i,i) =  -2*g11(i)-2*g22(i);
            % Y(i,i+1)=(g11(i)+P1/2) ;
            by(i) = by(i) - y(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
            Y(i,i-1)=(g11(i)-P1/2) ;
            % Y(i+n,i)=(g22(i)+P2/2) ;
            by(i) = by(i) - y(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
            Y(i-n,i)=(g22(i)-P2/2) ;
            % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
            % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
            % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
            Y(i-n,i-1)=(g12(i)/4+g21(i)/4);

        % Southeastern corner
        elseif etest && ~wtest && ~ntest && stest

            X(i,i) = -2*g11(i)-2*g22(i);
            % X(i,i+1)=(g11(i)+P1/2) ;
            bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
            X(i,i-1)=(g11(i)-P1/2) ;
            X(i+n,i)=(g22(i)+P2/2) ;
            % X(i-n,i)=(g22(i)-P2/2) ;
            bx(i) = bx(i) - x(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
            % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
            X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
            % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
            bx(i) = bx(i)...
                - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);


            Y(i,i) = -2*g11(i)-2*g22(i);
            % Y(i,i+1)=(g11(i)+P1/2) ;
            by(i) = by(i) - y(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
            Y(i,i-1)=(g11(i)-P1/2) ;
            Y(i+n,i)=(g22(i)+P2/2) ;
            % Y(i-n,i)=(g22(i)-P2/2) ;
            by(i) = by(i) - y(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
            % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
            Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
            % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
            by(i) = by(i)...
                - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

```

```
        % Northwestern corner
        elseif ~etest && wtest && ntest && ~stest

            X(i,i) =  -2*g11(i)-2*g22(i);
            X(i,i+1)=(g11(i)+P1/2) ;
            % X(i,i-1)=(g11(i)-P1/2) ;
            bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
            % X(i+n,i)=(g22(i)+P2/2) ;
            bx(i) = bx(i) - x(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
            X(i-n,i)=(g22(i)-P2/2) ;
            % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
            % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
            X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
            bx(i) = bx(i)...
                - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

            Y(i,i) =  -2*g11(i)-2*g22(i);
            Y(i,i+1)=(g11(i)+P1/2) ;
            % Y(i,i-1)=(g11(i)-P1/2) ;
            by(i) = by(i) - y(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
            % Y(i+n,i)=(g22(i)+P2/2) ;
            by(i) = by(i) - y(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
            Y(i-n,i)=(g22(i)-P2/2) ;
            % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
            % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
            Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
            by(i) = by(i)...
                - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);


        % Southwestern corner
        elseif ~etest && wtest && ~ntest && stest

            X(i,i) = -2*g11(i)-2*g22(i);
            X(i,i+1)=(g11(i)+P1/2) ;
            % X(i,i-1)=(g11(i)-P1/2) ;
            bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
            X(i+n,i)=(g22(i)+P2/2) ;
            % X(i-n,i)=(g22(i)-P2/2) ;
            bx(i) = bx(i) - x(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
            X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4);
            % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
            % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
            bx(i) = bx(i)...
                - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

            Y(i,i) = -2*g11(i)-2*g22(i);
            Y(i,i+1)=(g11(i)+P1/2) ;
            % Y(i,i-1)=(g11(i)-P1/2) ;
            by(i) = by(i) - y(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
            Y(i+n,i)=(g22(i)+P2/2) ;
            % Y(i-n,i)=(g22(i)-P2/2) ;
            by(i) = by(i)...
                - y(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
            Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4);
            % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
```

```
        % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
        by(i) = by(i)...
            - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

    % At eastern boundary (x = L)
    elseif etest && ~wtest && ~ntest && ~stest

        X(i,i) =  -2*g11(i)-2*g22(i);
        % X(i,i+1)=(g11(i)+P1/2) ;
        bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
        X(i,i-1)=(g11(i)-P1/2) ;
        X(i+n,i)=(g22(i)+P2/2) ;
        X(i-n,i)=(g22(i)-P2/2) ;
        % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
        bx(i) = bx(i)...
            - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
        X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
        % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
        bx(i) = bx(i)...
             - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
        X(i-n,i-1)=(g12(i)/4+g21(i)/4);

        Y(i,i) =  -2*g11(i)-2*g22(i);
        % Y(i,i+1)=(g11(i)+P1/2) ;
        by(i) = by(i) - y(getRow(i,n),getCol(i,n)+1)*(g11(i)+P1/2);
        Y(i,i-1)=(g11(i)-P1/2) ;
        Y(i+n,i)=(g22(i)+P2/2) ;
        Y(i-n,i)=(g22(i)-P2/2) ;
        % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
        by(i) = by(i)...
             - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4);
        Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
        % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
        by(i) = by(i)...
             - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
        Y(i-n,i-1)=(g12(i)/4+g21(i)/4);

    % At western boundary
    elseif ~etest && wtest && ~ntest && ~stest

        X(i,i) =  -2*g11(i)-2*g22(i);
        X(i,i+1)=(g11(i)+P1/2) ;
        % X(i,i-1)=(g11(i)-P1/2) ;
        bx(i) = bx(i) - x(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
        X(i+n,i)=(g22(i)+P2/2) ;
        X(i-n,i)=(g22(i)-P2/2) ;
        X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
        % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
        bx(i) = bx(i)...
             - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4);
        X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
        % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
        bx(i) = bx(i)...
             - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

        Y(i,i) =  -2*g11(i)-2*g22(i);
        Y(i,i+1)=(g11(i)+P1/2) ;
        % Y(i,i-1)=(g11(i)-P1/2) ;
        by(i) = by(i) - y(getRow(i,n),getCol(i,n)-1)*(g11(i)-P1/2);
        Y(i+n,i)=(g22(i)+P2/2) ;
        Y(i-n,i)=(g22(i)-P2/2) ;
        Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
        % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
        by(i) = by(i)...
             - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4);
        Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
        % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
        by(i) = by(i)...
             - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

    % At northern boundary (y = h)
    elseif ~etest && ~wtest && ntest && ~stest

        X(i,i) =  -2*g11(i)-2*g22(i);
        X(i,i+1)=(g11(i)+P1/2) ;
        X(i,i-1)=(g11(i)-P1/2) ;
```

```
            % X(i+n,i)=(g22(i)+P2/2) ;
            bx(i) = bx(i) - x(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
            X(i-n,i)=(g22(i)-P2/2) ;
            % X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            bx(i) = bx(i)...
                 - x(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
            % X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                 - x(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
            X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            X(i-n,i-1)=(g12(i)/4+g21(i)/4);

            Y(i,i) =  -2*g11(i)-2*g22(i);
            Y(i,i+1)=(g11(i)+P1/2) ;
            Y(i,i-1)=(g11(i)-P1/2) ;
            % Y(i+n,i)=(g22(i)+P2/2) ;
            by(i) = by(i) - y(getRow(i,n)+1,getCol(i,n))*(g22(i)+P2/2);
            Y(i-n,i)=(g22(i)-P2/2) ;
            % Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            by(i) = by(i)...
                 - y(getRow(i,n)+1,getCol(i,n)+1)*(g12(i)/4+g21(i)/4) ;
            % Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                 - y(getRow(i,n)+1,getCol(i,n)-1)*(-g12(i)/4-g21(i)/4) ;
            Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            Y(i-n,i-1)=(g12(i)/4+g21(i)/4);


        % At southern boundary (y = 0)
        elseif ~etest && ~wtest && ~ntest && stest

            X(i,i) = -2*g11(i)-2*g22(i);
            X(i,i+1)=(g11(i)+P1/2) ;
            X(i,i-1)=(g11(i)-P1/2) ;
            X(i+n,i)=(g22(i)+P2/2) ;
            % X(i-n,i)=(g22(i)-P2/2) ;
            bx(i) = bx(i) - x(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
            X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            % X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            bx(i) = bx(i)...
                 - x(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
            % X(i-n,i-1)=(g12(i)/4+g21(i)/4);
            bx(i) = bx(i)...
                 - x(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

            Y(i,i) = -2*g11(i)-2*g22(i);
            Y(i,i+1)=(g11(i)+P1/2) ;
            Y(i,i-1)=(g11(i)-P1/2) ;
            Y(i+n,i)=(g22(i)+P2/2) ;
            % Y(i-n,i)=(g22(i)-P2/2) ;
            by(i) = by(i) - y(getRow(i,n)-1,getCol(i,n))*(g22(i)-P2/2);
            Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            % Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            by(i) = by(i)...
                 - y(getRow(i,n)-1,getCol(i,n)+1)*(-g12(i)/4-g21(i)/4);
            % Y(i-n,i-1)=(g12(i)/4+g21(i)/4);
            by(i) = by(i)...
                 - y(getRow(i,n)-1,getCol(i,n)-1)*(g12(i)/4+g21(i)/4);

        %Not at any boundary
        else

            X(i,i) =  -2*g11(i)-2*g22(i);
            X(i,i+1)=(g11(i)+P1/2) ;
            X(i,i-1)=(g11(i)-P1/2) ;
            X(i+n,i)=(g22(i)+P2/2) ;
            X(i-n,i)=(g22(i)-P2/2) ;
            X(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            X(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            X(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            X(i-n,i-1)=(g12(i)/4+g21(i)/4);

            Y(i,i) =  -2*g11(i)-2*g22(i);
            Y(i,i+1)=(g11(i)+P1/2) ;
```

```
            Y(i,i-1)=(g11(i)-P1/2) ;
            Y(i+n,i)=(g22(i)+P2/2) ;
            Y(i-n,i)=(g22(i)-P2/2) ;
            Y(i+n,i+1)=(g12(i)/4+g21(i)/4) ;
            Y(i+n,i-1)=(-g12(i)/4-g21(i)/4) ;
            Y(i-n,i+1)=(-g12(i)/4-g21(i)/4) ;
            Y(i-n,i-1)=(g12(i)/4+g21(i)/4);

        end % if

        etest = false;
        wtest = false;
        ntest = false;
        stest = false;

    end %for

    xx = X\bx';                                             % Matrix inversion
    yy = Y\by';                                             % Matrix inversion

    x_mat = global2matrix(xx, n, m);
    y_mat = global2matrix(yy, n, m);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Check convergence
    cx = max(max(abs(x_mat-x(2:M-1,2:N-1))));
    cy = max(max(abs(y_mat-y(2:M-1,2:N-1))));

    cx_lim = 10^-3;
    cy_lim = 10^-3;

    if (cx < cx_lim && cy < cy_lim ) || it == maxits
        conv = 1; % Stop
    else

        it = it + 1;
    end %if

    % Under-relaxation:
    x(2:M-1,2:N-1) = (1-alpha)*x(2:M-1,2:N-1) +  alpha * x_mat;
    y(2:M-1,2:N-1) = (1-alpha)*y(2:M-1,2:N-1) + alpha * y_mat;



end % while
figure
plot(x,y,'k',x',y','k')
xlim([xA,xD])
ylim([yD,yC])
set(gca,'TickLabelInterpreter','latex')
xlabel('$x$-direction [m]', 'interpreter', 'latex')
ylabel('$y$-direction [m]', 'interpreter', 'latex')
saveas(gcf,'elliptic.png')
```

E.6.1.3 getCol.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                Function giving the column number of a node             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function colnumber = getCol(a, N)
colnumber = zeros(length(a),1);
    for j = 1:length(a)
        i = a(j);
        colnumber(j) =  mod(i-1, N)+1;
    end %for

    % Adjusting since the x matrix also contains boundary points
    colnumber = colnumber +1;
end %function
```

E.6.1.4 getRow.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                 Function giving the row number of a node               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function rownumber = getRow(a, N)
rownumber = zeros(length(a),1);
    for j = 1:length(a)
        i = a(j);
        rownumber(j) = floor((N+i-1)/N);
    end %for
    % Adjusting since the x matrix also contains boundary points
    rownumber = rownumber + 1;
end %function
```

E.6.1.5 matrix2global.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Function transforming a matrix into a globally indexed vector      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function res = matrix2global(vec, N, M)
    for j = 1:M
            vstart = 1;
            rowstartpoint = N*M + (j-M-1)*N + 1;
            rowendpoint = N*M + (j-M)*N;
        res(rowstartpoint:rowendpoint) = vec(j,vstart:N);
    end %for
end %function
```

E.6.1.6 global2matrix.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Function transforming a globally indexed vector into a matrix      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [matrix] = global2matrix(glob, N, M)
    for j = 1:M % "down"                 % the rest of the points are zero
        for i = 1:N % "left"
            matrix(j,i) = glob((j-1)*N + i);
        end % for
    end % for
end %function
```

Anita Klavenes

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

NTNU
Norwegian University of
Science and Technology