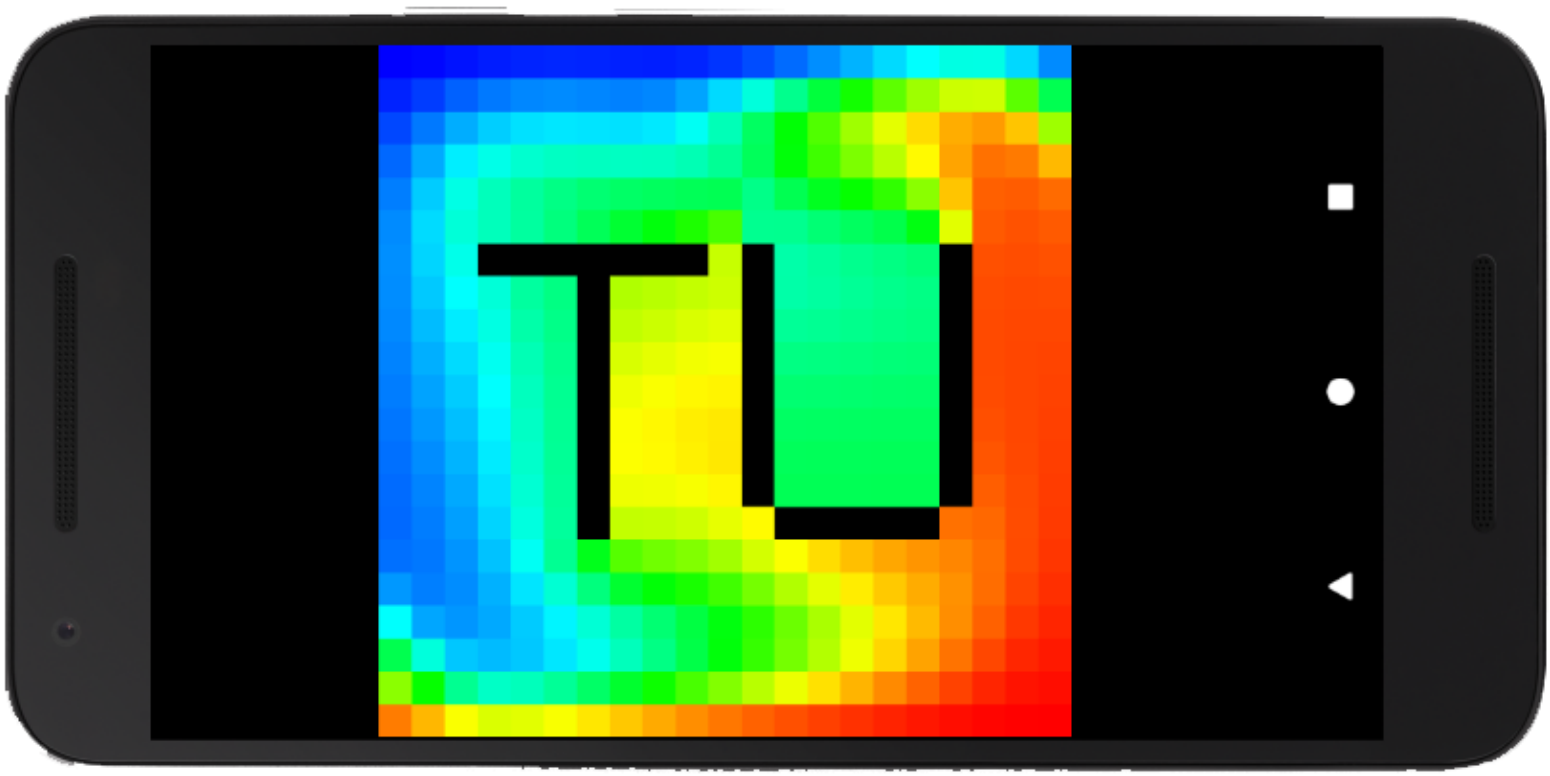
Mobile CFD solving

Solving natural convection problems in real time

W. Diepeveen



TU Delft

Mobile CFD solving

Solving natural convection problems in real time

by

W. Diepeveen

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Monday August 21, 2017 at 14:00 PM.

Student number:	4391098	
Project duration:	September 5, 2016 – August 21, 2017	
Thesis committee:	Prof. dr. ir. C. R. Kleijn,	TU Delft, supervisor
	Prof. dr. ir. C. Vuik,	TU Delft, supervisor
	Ir. K. van As,	TU Delft, supervisor
	Dr. A. R. Akhmerov,	TU Delft
	Dr. R. van der Toorn,	TU Delft

TU Delft

Abstract

We have explored the concept of mobile computational fluid dynamics (CFD) solving. Using the computational power of a smartphone we tried to generate real time simulations of relatively simple transient natural convection problems in the laminar regime. The focus lies on finding a compromise between accuracy, speed and stability: we want to make a real time mobile CFD solver that is as accurate as possible.

A JAVA application has been created that runs on Android. The SIMPLE algorithm has been implemented in order to solve for the flow and heat transfer. The SIMPLE algorithm on the application was tested for runtime performance and accuracy.

For a test problem, a real time simulation on a Nexus 5X has been realized with an accuracy of 5.4% on a 21×21 grid. Running the algorithm on a desktop gave a simulation 10 times faster than real time. Comparing this algorithm to a version converted to MATLAB, gave similar solving speed. We concluded that the bottleneck in the algorithm, solving matrix equations, could not be easily improved, because MATLAB solvers perform quite optimal: a significantly faster CFD solver implementing the SIMPLE algorithm probably does not exist.

Nevertheless, when trying to use the obtained results on water and air, we could not obtain any real time solutions. It is up to further research to define restictions that can guarantee real time solutions for fluids.

Contents

1 Introduction

Over the last six decades, computational power has increased enormously. In 1954 the IBM 704 was the first mass-produced computer able to handle complex mathematics. With only 12000 floating point operations per second (FLOPS), back then this was a whole achievement. Nowadays, the Tianhe-2 is the fastest supercomputer with 33,86 quadrillion FLOPS. This is a 1 trillion-fold (= 10^{12}) increase in computer performance [2]. In the meantime, smartphone performance has been increasing as well. A Samsung Galaxy S6, (altough not being the newest model) one of the best performing smartphones, can do almost 35 billion FLOPS. So the mobile phone is floating somewhere in the middle.

Furthermore, scientific computation has become more important than ever before: problems become more complex and so do the methods to solve them. Moreover, obtaining solutions as quick as possible is one of the challenges in the world of scientific computing. For example, if we want to generate a real time solution of a physical time-dependent process, the computation time must be faster than the actual physical time step. This is a rather interesting field of study, for the limits of solving power are really being tested. Therefore, companies such as MATHWORKS can dominate the world of computer science, since their algorithms can solve problems faster than most of the others.

In order to gain insight on the combination of processing capacity and solution generation limitation of a mobile phone, we will test the mobile phone on scientific calculation capacity. In this study we will try to make a real time mobile Computational Fluid Dynamics (CFD) solver. In particular we will be looking into a specific class of problems considering fluid flow and heat transfer, for not every arbitrarily complex problem will be real time realizable. The kind of problems we are going to solve can be characterized by the following requirements:

1. **Transient flow**: Time-dependent problems are considered.
2. **Laminar flow**: Viscosity-dominated flow withouth any small scale vortices.
3. **Incompressible flow**: The density of a fluid parcel is assumed to be constant.
4. **Buoyancy driven flow on a closed domain**: Natural convection is the primary forcing mechanism. Moreover, there is no incoming flow into the calculation domain, since the domain is closed.
5. **The system satisfies the Boussinesq approximation**: All fluid properties are assumed to be constant except for the density in the buoyancy term.
6. **Two dimensional flow on a rectangular domain**: By dropping the third dimension from the problem, the problem already becomes simpler[1]. By choosing a rectangular domain, mesh generation will become easier.
7. **Focus on water and air:** For the sake of simplicity only water and air systems will be considered.

Our goal is to create a smartphone application which can solve problems following the requirements and focussing on stability, speed and accuracy.

[1] In general, the number of cells N needed for a 2D problem is $N^{2/3}$, where N is the number of cells of a 3D problem.

In Chapter 2 the phenomena of interest will be described in more detail along with the difficulties within numerical mathematics. The SIMPLE algorithm will be introduced in Chapter 3 as a method for solving the problem. Moreover, methods for finding the results in terms of simulation error and runtime performance, will be discussed as well. The research results are discussed in Chapter 4 and the conclusions and recommendations drawn from the results are stated in Chapter 5.

2

Theory

In this chapter the problem of interest will be stated in Section 2.1 along with a numerical method for solving the problem in Section 2.2. In Section 2.3 the difficulties of the problem will be shown as well. These are the bottlenecks and therefore the challenge for achieving the final goal: solving the flow, pressure and temperature in real time constrained by the computational power of a smartphone. Finally, the error causes are discussed in Section 2.4.

2.1. Problem description

2.1.1. Physical restrictions

In Chapter 1 seven requirements were discussed to limit the class of problems of interest. However, some of these seven requirements can written down in a mathematically and physically more appropriate form. Since these problems are flow and heat problems, requirement 1 states that the Navier Stokes equation and the heat equation for unstationary problems are the governing equations.

Moreover, the Grashof number (Gr) is a non-dimensional number in fluid dynamics and heat transfer which approximates the ratio of the buoyancy to viscous force acting on a fluid. Therefore, this number can be related to requirement 2 and 4. Buoyancy-driven laminar flow in a side heated rectangular domain requires [4]:

$$\mathrm{Gr} = \frac{\beta g \Delta T L^3}{\nu^2} < 10^4, \tag{2.1}$$

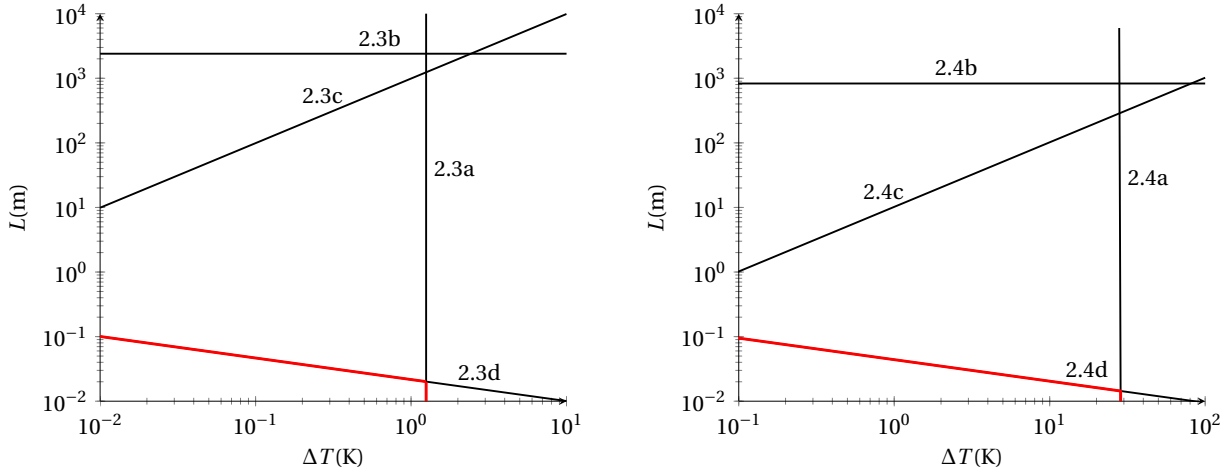
where β [K^{-1}] is the thermal expansion coefficient, g [m/s^2] is the acceleration due to gravity, ΔT [K] is the temperature difference of the extreme values on the boundary, L [m] is a characteristic length and ν [m^2/s] is the kinematic viscocity.

The third requirement states that the the velocity field is divergenceless:

$$\nabla \cdot \mathbf{v} = 0, \tag{2.2}$$

where $\mathbf{v}$ the velocity vector.

Requirement 5 states that the Boussinesq approximation must be satisfied. This approximation states that the material properties of the fluid are constant, except for the density. The change in density causing the buoyancy forcing term in the equations is linearized. However, this approximation is only valid in its validity region, which is defined as the set of combinations for L and ΔT giving a total error of at most 10%. For every fluid at a certain temperature and pressure, the vadility region can be calculated [4]. For example, when looking at water at $T = 287$ K (15°C) and $p = 1$ atm, the validity region can be described by Equations



(a) Water.

(b) Air.

Figure 2.1: The validation region of the Boussinesq approximation for water and air at $T = 287$ K and $p = 1$ atm. The red line indicates the limit of the class of simple problems.

2.3a-2.3c. Equation 2.3d guarantees the laminarity requirement.

$$\Delta T \leq 1.25\,\text{K} \tag{2.3a}$$
$$L \leq 2.4 \cdot 10^3\,\text{m} \tag{2.3b}$$
$$\frac{L}{\Delta T} \leq 9.9 \cdot 10^2\,\text{m/K} \tag{2.3c}$$
$$L^3 \Delta T < 1.02 \cdot 10^{-4}\,\text{m}^3\text{K}. \tag{2.3d}$$

The resulting region limited by the red line in Figure 2.1a indicates the class of simple problems of interest for water. For air similar results can be obtained (Figure 2.1b):

$$\Delta T \leq 28.6\,\text{K} \tag{2.4a}$$
$$L \leq 8.3 \cdot 10^2\,\text{m} \tag{2.4b}$$
$$\frac{L}{\Delta T} \leq 10.2\,\text{m/K} \tag{2.4c}$$
$$L^3 \Delta T < 8.50 \cdot 10^{-5}\,\text{m}^3\text{K}. \tag{2.4d}$$

2.1.2. Governing equations

Once the Boussinesq approximation has been satisfied, the governing equations are known. The equations of interest are [4]:

$$\nabla \cdot \mathbf{v} = 0 \tag{2.5a}$$
$$\frac{\partial u}{\partial t} + \nabla \cdot (u\mathbf{v}) = \nu_0 \nabla^2 u - \frac{1}{\rho_0} \frac{\partial (p + \rho_0 g z)}{\partial x} \tag{2.5b}$$
$$\frac{\partial v}{\partial t} + \nabla \cdot (v\mathbf{v}) = \nu_0 \nabla^2 v - \frac{1}{\rho_0} \frac{\partial (p + \rho_0 g z)}{\partial z} + \beta_0 g (T - T_0) \tag{2.5c}$$
$$\frac{\partial T}{\partial t} + \nabla \cdot (T\mathbf{v}) = \kappa_0 \nabla^2 T, \tag{2.5d}$$

where $\mathbf{v}$ [m/s] is the velocity vector in which u [m/s] is the velocity component in the horizontal x-direction and v [m/s] is the velocity component in the vertical z-direction, ν_0 [m^2/s] is the kinematic viscosity, ρ_0 [kg/m^3] is the density, $p + \rho_0 g z$ [Pa] is the hydrostatic pressure, β_0 [K^{-1}] is the thermal expansion coefficient, T [K] is the temperature, T_0 is the reference temperature and κ_0 [m^2/s] is the thermal diffusivity. The subscript 0 denotes the reference state (T_0, p_0) (with p_0 a reference pressure) which determines the values for ν_0, ρ_0, β_0 and κ_0. Furthermore, the last term in Equation 2.5c is the linearized term representing the buoyancy force.

When introducing $\tilde{p} = \frac{p+\rho_0 g z}{\rho_0}$ and $\tilde{T} = T - T_0$, the resulting equations are[1]:

$$\nabla \cdot \mathbf{v} = 0 \tag{2.6a}$$

$$\frac{\partial u}{\partial t} + \nabla \cdot (u\mathbf{v}) = \nu_0 \nabla^2 u - \frac{\partial \tilde{p}}{\partial x} \tag{2.6b}$$

$$\frac{\partial v}{\partial t} + \nabla \cdot (v\mathbf{v}) = \nu_0 \nabla^2 v - \frac{\partial \tilde{p}}{\partial z} + \beta_0 g \tilde{T} \tag{2.6c}$$

$$\frac{\partial \tilde{T}}{\partial t} + \nabla \cdot (\tilde{T}\mathbf{v}) = \kappa_0 \nabla^2 \tilde{T} \tag{2.6d}$$

2.1.3. Non-dimensional numbers

Apart from the Grashof number introduced in 2.1.1 as [7]

$$\mathrm{Gr} = \frac{\text{buoyancy forces}}{\text{viscous forces}} = \frac{\beta g \Delta T L^3}{\nu^2}, \tag{2.7}$$

other non-dimensional numbers play an important role.

The Prandtl number is defined as [7]

$$\mathrm{Pr} = \left(\frac{\text{hydrodynamic boundary layer thickness}}{\text{thermal boundary layer thickness}} \right)^{\alpha} = \frac{\nu}{\kappa} \tag{2.8}$$

for some α and says whether heat or momentum transport is dominant.

The Rayleigh number is defined as the product of the Gr and Pr [11]:

$$\mathrm{Ra} = \mathrm{Gr} \cdot \mathrm{Pr}. \tag{2.9}$$

Finally, the Nusselt number Nu is defined as [7]

$$\mathrm{Nu} = \frac{\text{total heat transfer}}{\text{conductive heat transfer}} = \frac{hL}{k}, \tag{2.10}$$

where h [W/m^2] is the heat transfer coefficient and k [W/(mK)] is the thermal conductivity. This non-dimensional number indicates the role of convective heat transport relative to conductive transport.

2.2. The Finite Volume Method

In the previous Section the problem has been properly defined. To solve the set of Equations 2.6 we will discretize them using the Finite Volume Method (FVM).

In the following sections the concept of the FVM will be discussed along with how to handle the boundary conditions, guidelines for convergence of the method and some difficulties of the problem and the method.

2.2.1. A short example

Consider the case of unsteady one-dimensional conduction for a material with a thermal diffusivity of κ [m^2/s]. The temperature is governed by

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x} \left(\kappa \frac{\partial T}{\partial x} \right). \tag{2.11}$$

The first step in solving the equation numerically, is choosing a grid. Since T lives in a one-dimensional coordinate space, the domain is a line segment.

In Figure 2.2 P is the grid point of interest. W and E are the adjacent grid points. Each grid point lies inside a control volume. For P, the control volume is the space between the interfaces w and e. The distance between P and E is denoted as $(\delta x)_e$ and the distance between P and W as $(\delta x)_w$. The control volume itself has size Δx.

Integrating Equation 2.11 from w to e and over the time interval from t to $t + \Delta t$ gives

[1] Rewriting the equations to the non dimensional form shows that the parameters of interest are merely depending on Gr and Pr (or Ra and Pr). This result will be discussed in the next chapter.

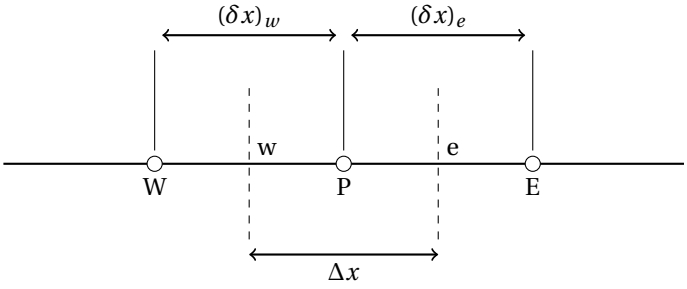


Figure 2.2: A line segment on which the heat equation will be solved. Three gridpoints and the interfaces between the gridpoints are shown as well as the distances between the grid points, $(\delta x)_w$ and $(\delta x)_e$, and the control volume size Δx.

$$\int_w^e \int_t^{t+\Delta t} \frac{\partial T}{\partial t}\,\mathrm{d}t\,\mathrm{d}x = \int_t^{t+\Delta t} \int_w^e \frac{\partial}{\partial x}\left(k\frac{\partial T}{\partial x}\right)\mathrm{d}x\,\mathrm{d}t \tag{2.12}$$

$$\Leftrightarrow \int_w^e T^1 - T^0\,\mathrm{d}x = \int_t^{t+\Delta t} \left(k\frac{\partial T}{\partial x}\right)\Big|_e - \left(k\frac{\partial T}{\partial x}\right)\Big|_w \mathrm{d}t, \tag{2.13}$$

where the superscript 1 indicates $t+\Delta t$ and the superscript 0 indicates t.

Evaluating the integrand in the right-hand side of Equation 2.13 with a piecewise linear profile between W, P and E gives

$$\int_w^e T^1 - T^0\,\mathrm{d}x = \int_t^{t+\Delta t} \left[\frac{k_e(T_E - T_P)}{(\delta x)_e} - \frac{k_w(T_P - T_W)}{(\delta x)_w}\right]\mathrm{d}t, \tag{2.14}$$

where k_e and k_w are the values for the thermal conductivity at interface e respectively w.

For evaluating the remaining integrals in Equation 2.14, profile assumptions have to be made for the two integrands. Assume a constant profile with the value at P for the left-hand side. The evaluation of the right-hand side integral can be done by proposing

$$\int_t^{t+\Delta t} T_i\,\mathrm{d}t = [\theta T_i^1 + (1-\theta)T_i^0]\Delta t, \tag{2.15}$$

where $\theta \in [0,1]$. The result is

$$\frac{\Delta x}{\Delta t}\left(T_P^1 - T_P^0\right) = \theta\left[\frac{k_e(T_E^1 - T_P^1)}{(\delta x)_e} - \frac{k_w(T_P^1 - T_W^1)}{(\delta x)_w}\right] + (1-\theta)\left[\frac{k_e(T_E^0 - T_P^0)}{(\delta x)_e} - \frac{k_w(T_P^0 - T_W^0)}{(\delta x)_w}\right]. \tag{2.16}$$

Dropping the superscript 1 from the T's and remembering that a value of T without a superscript stands for the new value, and rearranging Equation 2.16 gives

$$a_P T_P = a_E[\theta T_E + (1-\theta)T_E^0] + a_W[\theta T_W + (1-\theta)T_W^0] + [a_P^0 - (1-\theta)a_E - (1-\theta)a_W]T_P^0, \tag{2.17}$$

where

$$a_E = \frac{k_e}{(\delta x)_e} \tag{2.18a}$$

$$a_W = \frac{k_w}{(\delta x)_w} \tag{2.18b}$$

$$a_P^0 = \frac{\Delta x}{\Delta t} \tag{2.18c}$$

$$a_P = \theta a_E + \theta a_W + a_P^0 \tag{2.18d}$$

are the coefficients. Note that for every integral different profile assumptions have been made for T. This is permitted and therefore one of the eases of this method.

Since the integration steps can be performed for every gridpoint on the domain, Equation 2.17 results in a system of linear equations. When the boundary conditions are discretized in a similar way and a value for θ has been chosen, the solution to the heat problem can be calculated using linear algebra. The boundary discretization and a value for θ will be discussed in Sections 2.2.3 and 2.2.4.

2.2.2. The theory

For the more general case, consider an operator $\mathscr{L}$. We will be looking for a function ϕ that satisfies the inital and boundary conditions and furthermore

$$\mathscr{L}[\phi] = 0. \tag{2.19}$$

So ϕ satisfies a local balance which is determined by what kind of operator $\mathscr{L}$ is. Now, in order to solve for ϕ numerically, the space is again split up in discrete control volumes that fill the entire domain[2]. A single grid point is located inside each control volume. We want to know the value for ϕ at each grid point as a numerical result of the Finite Volume Method. In order to accomplish this, profile assumptions have to be made. These profile assumptions set the shape of the numerical approximation. Only the function values at the grid points themselves remain unknown. This, however, can be solved in a similar way as done in the previous section.

Consider such a resulting piecewise function $\bar{\phi}$, a control volume V and a function W_V such that

$$W_V(\mathbf{x}, t) = \begin{cases} 1 & (\mathbf{x}, t) \in V \\ 0 & (\mathbf{x}, t) \notin V \end{cases}. \tag{2.20}$$

Furthermore, since $\bar{\phi}$ probably will not satisfy Equation 2.19, we know that

$$\mathscr{L}[\bar{\phi}] = R \neq 0, \tag{2.21}$$

where R is some residual term. Regardless, the shape of $\bar{\phi}$ is known, whereas the function values at the grid points are not. So there is still some freedom in $\bar{\phi}$ to minimize the residual term. When the residual term is multiplied by W_V and integrated over the domain, the following requirement is claimed:

$$\int_{t_0}^{t} \iiint_{\Omega} R(\mathbf{x}, t) W_V(\mathbf{x}, t) \, \mathrm{d}\mathbf{x} \, \mathrm{d}t = 0. \tag{2.22}$$

So instead of requiring the residual term to be zero, the integral over every control volume V has to be zero. Using Equation 2.21 and writing $V = V_S \times (t_1, t_1 + \Delta t)$ gives

$$\int_{t_1}^{t_1+\Delta t} \iiint_{V_S} \mathscr{L}[\bar{\phi}] \, \mathrm{d}\mathbf{x} \, \mathrm{d}t = 0. \tag{2.23}$$

In other words, a control volume overall balance is another way of interpreting the requirement. Once the integrals have been evaluated, each control volume produces a linear equation for the grid point values and each control volume represents a single value for $\bar{\phi}$. So integrating over every control volume V gives enough equations to solve for the number of unknowns [8]. Furthermore, the resulting $\bar{\phi}$ has the property to satisfy not only an overall balance, but a micro balance at every single control volume.

The resulting general form of the linear equations which arise from the control volume integral balance for a certain quantity ϕ can always be written in the form

$$a_P \phi_P = \sum_{nb} a_{nb} \phi_{nb} + b, \tag{2.24}$$

where P is the grid point of interest, the summation is held over all the neighbouring control volumes and b is the result of a discretized source, the collection of terms independent of the value for ϕ.

2.2.3. Boundary conditions

The application will be able to deal with two kinds of boundary conditions: Dirichlet and Neumann boundary conditions[3]: the velocity equations have no slip boundary conditions (Dirichlet) and the temperature equation has either insulated walls (Neumann) or walls with a known temperature (Dirichlet).

In order to derive the general form of the boundary equations a slight distinction has to be made already. Later in Section 2.3.1 it will be shown that multiple grids are needed to prevent the solution from exerting oscilatory behaviour. Therefore, we have multiple forms for the boundary equations. We will distinguish between the cases where

[2] Note that domain is used in a general way: the temporal dimensions as well as the spatial dimensions have to be discretized.
[3] So other kinds of boundary conditions will not be implemented.

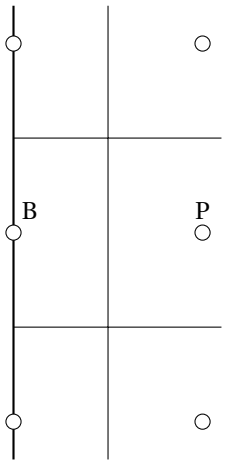


Figure 2.3: A grid point P adjacent to the boundary and a corresponding boundary cell with a grid point B at the boundary of the calculation domain.

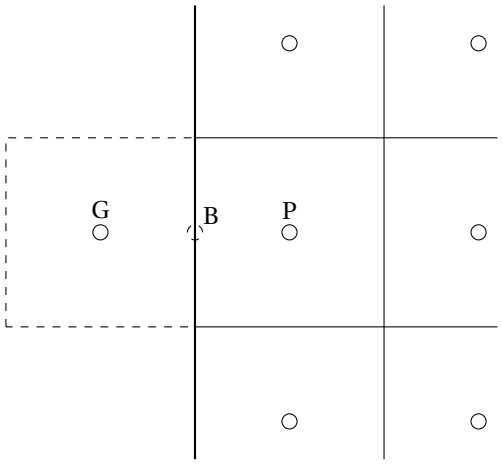


Figure 2.4: A grid point P at the boundary and a corresponding ghost cell with a grid point G outside the discretized domain.

- the boundary condition is implemented using a grid point that lies on the boundary;
- the boundary condition is implemented using a grid point which lies in a ghost cell outside the domain.

Consider the first case. The situation is sketched in Figure 2.3. This case is valid for the east and west boundaries of u and for the north and south boundaries of v. Therefore, only Dirichlet boundary conditions have to be considered. The resulting equation for the boundary condition is very straightforward:

$$a_P \phi_P = \sum_{nb \neq B} a_{nb} \phi_{nb} + a_B \phi_B + b \tag{2.25}$$

$$\Rightarrow a_P \phi_P = \sum_{nb \neq B} a_{nb} \phi_{nb} + b, \tag{2.26}$$

because $\phi_B = 0$. The subscript B denotes the boundary grid point.

Now, consider the second case. The situation is sketched in Figure 2.4. The control volume faces are in line with the boundary. Therefore, a so-called ghost cell is placed for calculation purposes. This case applies to every boundary of $\tilde{T}$ and for the remaining boundaries of u and v. For Dirichlet boundary conditions, the known boundary value ϕ_B can be approximated by

$$\phi_B = \frac{\phi_G + \phi_P}{2}. \tag{2.27}$$

Rewriting gives

$$\phi_G = -\phi_P + 2\phi_B \tag{2.28a}$$
$$\Rightarrow a_G\phi_G = -a_G\phi_P + 2a_G\phi_B. \tag{2.28b}$$

Combining Equation 2.28b with Equation 2.24 gives

$$(a_P + a_G)\phi_P = \sum_{nb\neq G} a_{nb}\phi_{nb} + 2a_G\phi_B + b. \tag{2.29}$$

Neumann boundary conditions (which will be used for $\tilde{T}$ only) can be approximated by

$$\frac{\phi_G - \phi_P}{(\delta x)_g} = \nabla\phi_B \cdot \hat{n}, \tag{2.30}$$

where $\hat{n}$ is the unit normal vector pointing outwards the control volume. Rewriting gives

$$\phi_G = \phi_P + (\delta x)_g \nabla\phi_B \cdot \hat{n} \tag{2.31a}$$
$$\Rightarrow a_G\phi_G = a_G\phi_P + a_G(\delta x)_g \nabla\phi_B \cdot \hat{n}. \tag{2.31b}$$

Combining Equation 2.31b with Equation 2.24 gives

$$(a_P - a_G)\phi_P = \sum_{nb\neq G} a_{nb}\phi_{nb} + a_G(\delta x)_g \nabla\phi_B \cdot \hat{n} + b = \sum_{nb\neq G} a_{nb}\phi_{nb} + b, \tag{2.32}$$

where zero gradient boundary conditions were used in the final equality.

2.2.4. Guidelines for consistency, stability and convergence

In order to construct a good numerical scheme, it is important that the result is a physically realistic solution. Whether the numerical method is highly accurate or not, we at least want to ensure mass conservation and avoid phenomena such as internal temperatures being higher than boundary values in the absence of an interior source. Thus, we want the approximate solution to lie close to the exact solution and the solution can be expected to become more accurate when a finer grid spacing and a smaller time step is used: we want the solution to converge. Especially when being constrained to the computational power of a smartphone, certain lower bounds on the grid spacing and the time step must be set in order to guarantee a realtime simulation. Even for those lower bounds the numerical scheme must give reasonable solutions.

Convergence for linear schemes can often be verified using Lax's Equivalence Theorem [10]. Consistency and stability are the key to convergence. It says approximately that

- consistency guarantees the local truncation error (the error between the exact solution and the numerical solution) to go to zero when the grid spacing and the time step do as well;
- stability guarantees the scheme to produce a bounded solution whenever the exact solution is bounded.

However, in practice problems do not necessarily meet the criteria imposed by the Lax Equivalence Theorem due to nonlinearity (see Section 2.3). Even if they do, consistency and convergence can be hard to prove. Therefore, our scheme will be built upon the idea of maximizing the possibilties for convergence. To accomplish this, Patankar [8] states four basic rules for numerical Finite Volume schemes. Two of these are useful for our purposes:

Rule 1: Consistency at control volume faces
When a face is common to two adjacent control volumes, the flux across it must be represented by the same expression in the discretization equations for the two control volumes.

Since the equations are conservation laws, the first rule sounds obvious. Heat or momentum leaving a control volume must be identical to the heat of momentum entering the neighbouring cells. Looking at the small example in Section 2.2.1, for the heat flux a linear profile was assumed. In this case the flux across the cell face is indeed consistent. However, it can be checked that some higher order profiles (like a quadratic profile) do not obey the first rule. Although a higher order estimate should give a more accurate description,

divergence is more likely to occur in that case.

Rule 2: Positive coefficients
All coefficients $\{a_P, a_{nb}\}$ must always be positive.

The second rule is quite straightforward as well. Consider a gridpoint P. Assume the value for ϕ, the variable of interest, increases in time in one or more neighbouring grid points of P. If all other conditions remain unchanged, one might expect ϕ_P to increase as well. Since the coefficients function as weights, a negative weight on an increasing value in a neighbour gridpoint can cause a decrease in P. Again unphysical results arise. Therefore, the coefficients should all have the same sign (which is chosen to be positive).

Now, consider the example in Section 2.2.1 again. Choosing $\theta = 1$ brings no problems at all, since all coefficients are positive. This kind of schemes are called implicit since the new values for T at the neighbour grid points are used to calculate T_P. This coupled system of equations has to be solved simultaneously in order to find all new values of T. The process of solving this system can be a challenge, since it is basically a matrix inversion. Computationally this is harder than for example matrix multiplication, which is the case when $\theta = 0$ is chosen (this will be discussed in Section 2.3.5). Nevertheless, now the possibility arises that the coefficient of T_P^0 has a negative value. In order to satisfy rule 2

$$0 < a_P^0 - a_E - a_W = \frac{\Delta x}{\Delta t} - \frac{k_e}{(\delta x)_e} - \frac{k_w}{(\delta x)_w} \leq \frac{\Delta x}{\Delta t} - 2\frac{k_{max}}{(\delta x)_{min}} \tag{2.33a}$$

$$\Leftrightarrow \Delta t < \frac{\Delta x \cdot (\delta x)_{min}}{2k_{max}} \tag{2.33b}$$

must hold. Although an extra requirement is not ideal, computationally the solution to this scheme (called an explicit scheme) is easier to calculate because the new value for T in every point can be found using old values only, which are already known. Nevertheless, for the sake of stability and eventually convergence, implicit schemes are preferred for our purposes.

2.3. Difficulties of the problem

The Finite Volume Method as described above sounds very promising. However, there are some difficulties which have to be taken into account in order to get reasonable results. In the following sections the major difficulties are described and an indication for solving these difficulties will be worked out. In the next chapter, an explicit way of tackling all these problems will be given in the form of suitable algebraic equations and an algorithmic approach of the total problem.

2.3.1. Unphysical results for a single grid

Consider $\nabla \cdot \mathbf{v}$ in Equation 2.6a and the $\nabla \tilde{p}$ term in the combined Equations 2.6b and 2.6c. Suppose our numerical method would calculate the values for u, v and $\tilde{p}$ in the same grid points. Now, discretization could give unphysical results. The phenomenon of checkerboarding arises.

To see this, consider the one dimensional version of the terms: $\frac{\mathrm{d}u}{\mathrm{d}x}$ and $\frac{\mathrm{d}\tilde{p}}{\mathrm{d}x}$. Integrating over a control volume covering gridpoint P gives

$$u_e - u_w = \frac{u_E + u_P}{2} - \frac{u_P + u_W}{2} = \frac{u_E - u_W}{2} \tag{2.34}$$

for u and

$$\tilde{p}_e - \tilde{p}_w = \frac{\tilde{p}_E + \tilde{p}_P}{2} - \frac{\tilde{p}_P + \tilde{p}_W}{2} = \frac{\tilde{p}_E - \tilde{p}_W}{2} \tag{2.35}$$

for $\tilde{p}$, if linear profiles are assumed and the control volume faced are assumed to be midway between the grid points. Now consider the field in Figure 2.5. If the field were inserted in the discretization equations in

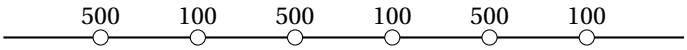


Figure 2.5: A field with oscilatory behaviour.

Equations 2.34 and 2.35, the numerical method would see a divergenceless velocity field and a zero pressure gradient. As we see, this is not the case. Furthermore, one can imagine that the higher dimensional case

allows even more complex checkerboard patterned velocity and pressure fields. Therefore, a gigantic problem has arisen.

This problem can be solved using a staggered grid [8]. Instead of solving the system of equations on one grid, three grids shifted relative to each other are used. This is shown in Figure 2.6. The grid for the u velocity is shifted in the horizontal direction and the grid of v in the vertical direction with respect to the regular grid.

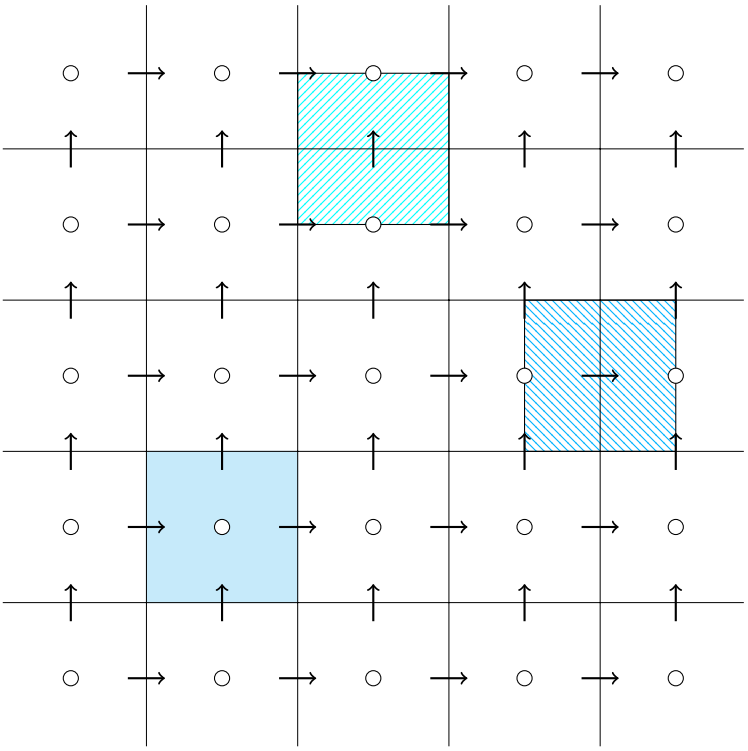


Figure 2.6: Staggered locations for u and v. $u = \rightarrow$, $v = \uparrow$ and the other variables are located at the regular grid points. An example of the control volumes in the different grids is also shown.

Now, the left hand side of Equation 2.34 (generalized to the higher dimensional case) can be used immediately, velocities are stored in the faces of the regular grid. A similar result holds for the pressure.

2.3.2. Nonlinearity

When using nonlinear partial differential equations, the result of discretization is not a system of linear equations: the coefficients in the discretization equations are functions of the nonlinear variable. So in that way solving the system of equations for ϕ requires the resulting values of ϕ for the coefficients. A problem arises, which can be solved using iteration:

1. Start with an initial guess for ϕ for all grid points.
2. Assemble the coefficients for the nonlinear discretization equations. A linear system of equations is obtained.
3. Solve the system of linear equations using the guessed values.
4. Use the new values as a better guess for ϕ and return to step 2 until the difference between the values for the previous and the current iteration becomes sufficiently small.

The result of the iterative process, resembles the solution of the set of nonlinear equations. However, convergence of the iterative process cannot be guaranteed [8], but a good numerical method maximizes the odds. The basic rules stated in 2.2.4 help, but there are more strategies.

One such strategy is underrelaxation. As mentioned, iterative processes can diverge. This divergence can be caused by abruptly changing values found at step 4 (compared to the old values) and therefore changing coefficients in step 2 of the next iteration. This can aggravate the next iteration even more and divergence

occurs. Especially for highly nonlinear equations numerical convergence can be hard to obtain. In this case it would be useful to slow down changes.

Consider the general discretization of the form

$$a_P \phi_P = \sum_{nb} a_{nb} \phi_{nb} + b. \tag{2.36}$$

The new value for ϕ can be modified inserting the underrelaxation constant $\alpha \in [0,1]$ and the value of the previous iteration ϕ_P^*:

$$\phi_P = \phi_P^* + \alpha \left(\frac{\sum_{nb} a_{nb} \phi_{nb} + b}{a_P} - \phi_P^* \right) \tag{2.37a}$$

$$\Leftrightarrow \frac{a_P}{\alpha} \phi_P = \sum_{nb} a_{nb} \phi_{nb} + b + (1-\alpha) \frac{a_P}{\alpha} \phi^*. \tag{2.37b}$$

The resulting scheme is the system of equations in Equation 2.37b. The effect, however, can best be understood investigating Equation 2.37a. Since α is chosen between 0 and 1, the new value for ϕ will be closer to the previous one: changes are decelerated. However, when the solution convergences, we have $\phi \approx \phi^*$. So the term between brackets in Equation 2.37a must be approximately zero and the non linear algebraic equations are solved.

2.3.3. Representation of the continuity equation and the pressure-gradient term

The four governing equations of our problem, Equations 2.6, describe four variables which might be solvable[4].

However, the system gives 3 criteria for the velocity field(Equation 2.6a-2.6c), but no direct equation for pressure. Nevertheless, obtaining an equation for $\tilde{p}$ should be possible, since we have four equations and four unknowns. In order to derive the equation for $\tilde{p}$ we take the divergence of Equation 2.6b and 2.6c. Writing these equations in Cartesian tensor notation, the deriviation will be clearer:

$$\frac{\partial}{\partial x_i} \left(\frac{\partial v_i}{\partial t} + \frac{\partial}{\partial x_j} (v_i v_j) \right) = \frac{\partial}{\partial x_i} \left(\frac{\partial}{\partial x_j} \left(\frac{\partial v_i}{\partial x_j} \right) - \frac{\partial \tilde{p}}{\partial x_i} + \beta_0 g \tilde{T} \delta_{i2} \right), \tag{2.38a}$$

$$\Leftrightarrow \quad \frac{\partial}{\partial t} \frac{\partial v_i}{\partial x_i} + \frac{\partial^2}{\partial x_i \partial x_j} (v_i v_j) = \frac{\partial}{\partial x_j} \left(\frac{\partial}{\partial x_j} \frac{\partial v_i}{\partial x_i} \right) - \frac{\partial}{\partial x_i} \left(\frac{\partial \tilde{p}}{\partial x_i} \right) + \beta_0 g \frac{\partial \tilde{T}}{\partial x_2}. \tag{2.38b}$$

In the expression above δ_{ij} is the Kronecker delta and $i, j = 1,2$ represent x and z. Now we know from Equation 2.6a that $\frac{\partial v_i}{\partial x_i} \equiv \nabla \cdot \mathbf{v} = 0$. This results in a Laplace equation for the pressure:

$$\nabla^2 \tilde{p} = -\frac{\partial^2}{\partial x_i \partial x_j} (v_i v_j) + \beta_0 g \frac{\partial \tilde{T}}{\partial z}. \tag{2.39}$$

The coupled system of equations, with the continuity equation replaced by Equation 2.39 could be used to solve for the four variables. In practice another solving method is used for the problem, since this approach (using an equation for pressure) contains even more nonlinearities: possibilities for divergence are again created and this is not desirable.

In Section 3.4 of the next chapter, an algorithmic approach is introduced, which avoids the nonlinearities of the pressure and therefore gives a more reliable scheme.

2.3.4. Boundary layer effects

Due to natural convection in the vicinity of the boundaries, boundary layer effects occur: a layer adjacent to the wall arises in which the heat flow is higher than in the bulk[5] and the velocity is smaller than in the bulk due to the no slip boundary conditions.

For any numerical solution to give an accurate description, it is important to know what happens within the boundary layer.

Therefore, we want to know how many grid cells we need in the boundary layer and the thickness of the boundary layer. The minimum number of grid cells within the thermal boundary layer δ_T [m] can be estimated by [9]

[4] Note, having four equations is necessary but not sufficient.

[5] This holds for fixed T at the wall only, not for insulated walls.

$$N_{BL}^{T} \geq \begin{cases} \sqrt{2a}\mathrm{Nu}^{1/2}\mathrm{Pr}^{-3/4}A^{3/2}\pi^{3/4}, & \mathrm{Pr} < 3\cdot 10^{-4}, \\ \sqrt{2a}\mathrm{Nu}^{1/2}\mathrm{Pr}^{-0.5355+0.033\log \mathrm{Pr}}, & 3\cdot 10^{-4} \leq \mathrm{Pr} \leq 1, \\ \sqrt{2a}\mathrm{Nu}^{1/2}\mathrm{Pr}^{-0.0355+0.033\log \mathrm{Pr}}, & 1 < \mathrm{Pr} \leq 3, \\ \sqrt{2a}\mathrm{Nu}^{1/2}E^{3/2}, & \mathrm{Pr} > 3, \end{cases} \tag{2.40}$$

while the minimum number of grid points in the kinetic boundary layer δ_u [m] is

$$N_{BL}^{u} \geq \begin{cases} \sqrt{2a}\mathrm{Nu}^{1/2}\mathrm{Pr}^{-1/4}A^{1/2}\pi^{1/4}, & \mathrm{Pr} < 3\cdot 10^{-4}, \\ \sqrt{2a}\mathrm{Nu}^{1/2}\mathrm{Pr}^{-0.1785+0.011\log \mathrm{Pr}}, & 3\cdot 10^{-4} \leq \mathrm{Pr} \leq 1, \\ \sqrt{2a}\mathrm{Nu}^{1/2}\mathrm{Pr}^{0.3215+0.011\log \mathrm{Pr}}, & 1 < \mathrm{Pr} \leq 3, \\ \sqrt{2a}\mathrm{Nu}^{1/2}\mathrm{Pr}^{1/3}E^{1/2}, & \mathrm{Pr} > 3, \end{cases} \tag{2.41}$$

where $a = 0,482$, $A = 0,332$ and $E = 0,982$.

The boundary thicknesses can be found trough

$$\frac{\delta_T}{\delta_u} = \begin{cases} A\sqrt{\pi}\mathrm{Pr}^{-1/2}, & \mathrm{Pr} < 3\cdot 10^{-4}, \\ \mathrm{Pr}^{-0.357+0.22\log \mathrm{Pr}}, & 3\cdot 10^{-4} \leq \mathrm{Pr} \leq 3, \\ E\mathrm{Pr}^{-1/3}, & \mathrm{Pr} > 3, \end{cases} \tag{2.42}$$

$$\frac{\delta_u}{L} = \begin{cases} 0,5\mathrm{Nu}^{-1}\mathrm{Pr}^{1/2}A^{-1}\pi^{-1/2}, & \mathrm{Pr} < 3\cdot 10^{-4}, \\ 0,5\mathrm{Nu}^{-1}\mathrm{Pr}^{0.357-0.022\log \mathrm{Pr}}, & 3\cdot 10^{-4} \leq \mathrm{Pr} \leq 3, \\ 0,5\mathrm{Nu}^{-1}\mathrm{Pr}^{1/4}E, & \mathrm{Pr} > 3. \end{cases} \tag{2.43}$$

The correlation for the Nu number for a vertical wall is used [3]:

$$\mathrm{Nu} = 0.68 + \frac{0.67\ \mathrm{Ra}^{1/4}}{[1+(0.492/\mathrm{Pr})^{9/16}]^{4/9}} \qquad \text{for } \mathrm{Ra} < 10^{9}. \tag{2.44}$$

The resulting minimum number of cells along a boundary wall on a uniform grid is:

$$n \geq \max\left(\frac{L}{\delta_u}N_{BL}^{u}, \frac{L}{\delta_T}N_{BL}^{T}\right). \tag{2.45}$$

Considering water and air at $T = 287$ K (15°C) and $p = 1$ atm: $n_{\text{water}} \geq 8$ and $n_{\text{air}} \geq 17$.

2.3.5. The size of the matrices

As the number of grid cells has to be increased due to convergence and accuracy reasons, the number of linear equations also increases and thus the matrices grow along in size. The problem with these large matrices is that solving matrix equations regularly involves inverting the matrix, which becomes increasingly difficult when the matrices become bigger.

Figure 2.7 shows that a typical solving time for a 1000×1000-matrix is about a third of a second for the regular JAVA libraries (so excluding MTJ-N). Every iteration at least four matrix equations of a similar size have to be solved[6]. At that rate, a realtime solution will not be realizable. Furthermore, it is unknown in advance how many iterations are necessary.

Nevertheless, the sparse character of the matrices has not yet been used. Since most of the matrix elements are zero, sparse matrix algorithms can help. These algorithms can be split up into direct solvers and iterative solvers. Iterative solvers can be optimized really well, but have the disadvantage of possible divergence. Direct solvers do not have this disadvantage, but cannot always compete with iterative solvers in terms of excecution time. In order to guarantee convergence, a direct solver will be used in our research: LU decomposition.

The idea of LU decomposition is to split up a matrix A into a lower triangular matrix L and an upper triangular matrix U of the same size such that

$$A = LU. \tag{2.46}$$

Now, given a vector $\mathbf{b}$ and a system of equations

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow LU\mathbf{x} = \mathbf{b}, \tag{2.47}$$

the solution $\mathbf{x}$ can be found in two logical steps:

[6] horizontal velocity, vertical velocity, pressure and temperature

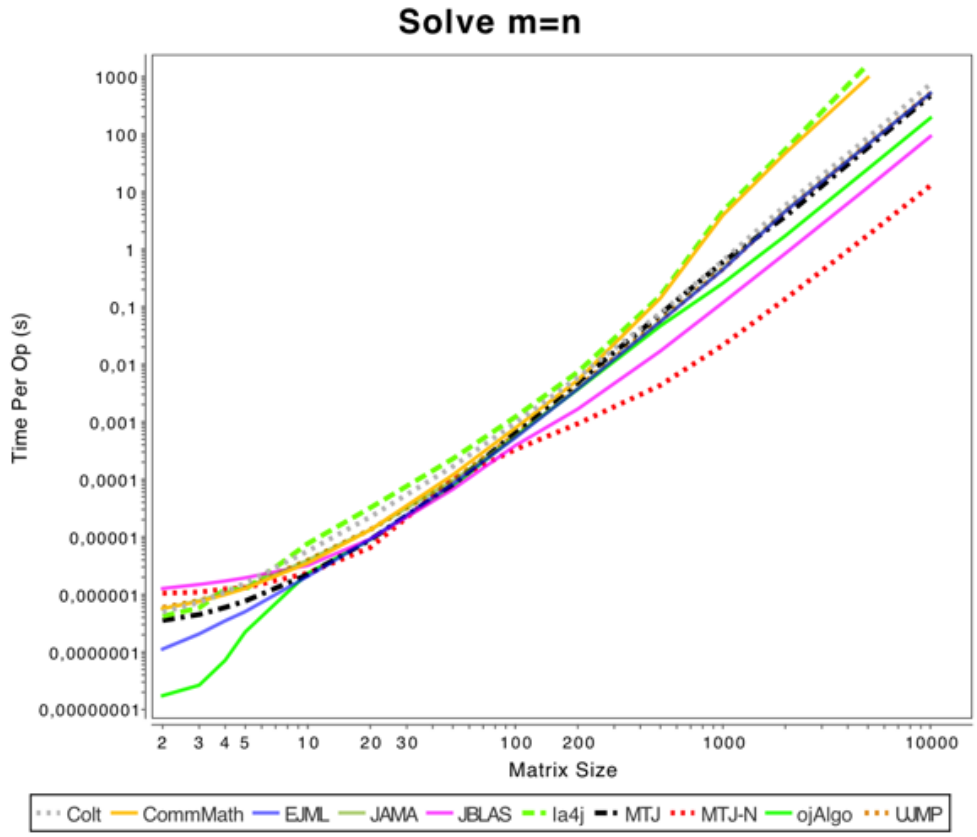


Figure 2.7: A JAVA benchmark for time it takes to solve a matrix equation for different matrix sizes. [1]

1. First, we solve $L\mathbf{y} = \mathbf{b}$, which can be performed using forward substitution;
2. Second, we solve $U\mathbf{x} = \mathbf{y}$, which can be performed using backward substitution.

We will not discuss the LU decomposition algorithms here.

2.4. Accuracy

Finally, we will want to have an estimate for the error of the obtained solution. In order to give an error, the cause of the error has to be inspected:

1. **Integration error**: Every integration over a control volume results in an error being made.
2. **Interpolation error**: Every probed value for the calculation of the coefficients through interpolation results in an error being made.
3. **Error due to nonlinearity**: At a certain moment, the iteration for the nonlinearity truncates. An error remains.
4. **Error due to the Boussinesq approximation**: The assumption that material properties of the fluid can be kept constant, results in an error being made.

The total error is the result of the error contributions. Although this error is hard to estimate, we can guess which of the four terms will give the biggest contribution. In general, the error of the Boussinesq approximation is small (within the region of validity), because varying fluid properties turn out to be not that big of an influence [4]. The error shall thus mainly be determined by the mathematical limitations of the numerical method.

These mathematical limitations can be tested when the results are compared to an analytical solution, experiments or to other simulation data. For error estimation, a distance between a solution and our estimate

solution has to be defined, which is done by choosing a metric. The relative error using $\|\cdot\|_2$ is the chosen metric. The error is defined as

$$\frac{\|\hat{\mathbf{a}} - \mathbf{a}\|_2}{\|\mathbf{a}\|_2}, \tag{2.48}$$

where $\hat{\mathbf{a}}$ is the vector of estimated values on a set of grid points and $\mathbf{a}$ is the vector containing the correct values on this specific set of grid points.

Repeating the simulation for multiple grid sizes gives a set of errors from which the order of the global error can be deduced. Although this error is only valid for the particular problem for which a (numerical) solution is already known, it can also give an indication about the global error of a different problem.

3

Method

In this chapter, a general overview of the possibilities and functionality of the application are described first in Section 3.1. Then in Section 3.2, the resulting discretization equations obtained from the FVM are discussed. These equations cannot be solved immediately. First, the initialization will be described in Section 3.3. Then, an algorithmic approach, the SIMPLE algorithm, is discussed in Section 3.4 to obtain a solution. In Section 3.5, the method implementation in JAVA is discussed. Finally in Section 3.6, we will consider how we try to find real time solutions with a JAVA script and estimate the accuracy. The comparison to MATLAB is also discussed.

3.1. The application

In order to gain insight into the functionality of the application, a use case diagram of the application is shown in Figure 3.1.

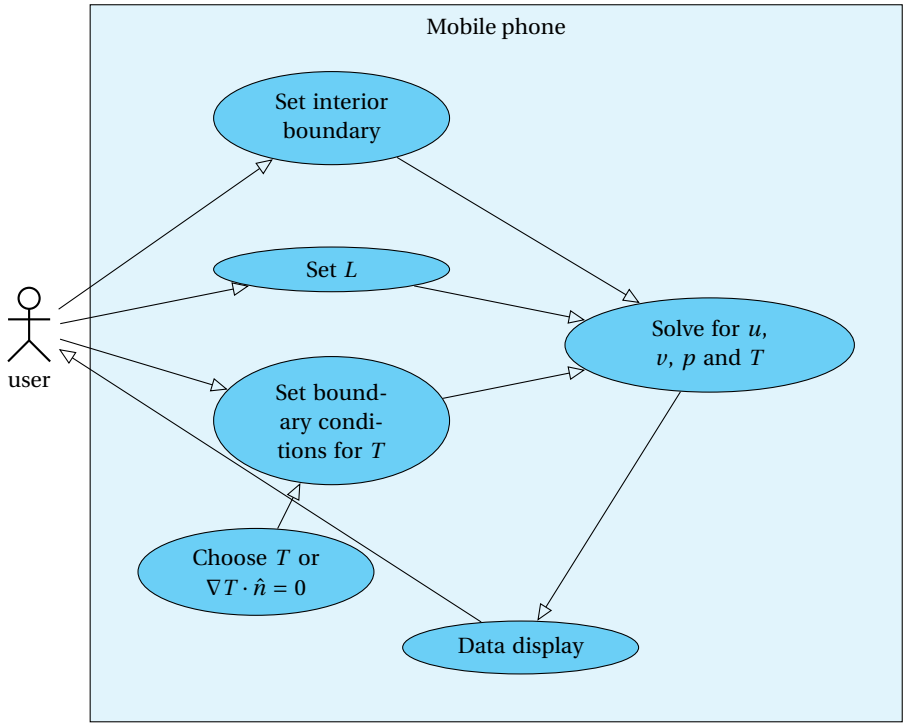


Figure 3.1: The use case diagram for the mobile CFD solving application, where L is the length of the side of the LxL square in which a solution is calculated.

The user input comprises the locations of internal boundaries, the characteristic length, boundary conditions for T on the internal boundary points and on the edges of the domain. Then the user will be able to see

the solution to his problem using coloured pixel map of the variables. The application itself should be able to generate solutions in realtime for all physical variables.

3.2. Discretization

For a two dimensional grid with square cells (so $\Delta x = \Delta z$) the resulting set of linear equations for a general quantity ϕ subject to an unsteady convection and diffusion equation is [8]:

$$a_P\phi_P = a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + a_{P0}\phi_{P0} + b, \tag{3.1}$$

where

$$a_E = \Gamma A\left(\frac{u_e\Delta x}{\Gamma}\right) + max(-u_e\Delta x, 0) \tag{3.2a}$$

$$a_W = \Gamma A\left(\frac{u_w\Delta x}{\Gamma}\right) + max(u_w\Delta x, 0) \tag{3.2b}$$

$$a_N = \Gamma A\left(\frac{v_n\Delta x}{\Gamma}\right) + max(-v_n\Delta x, 0) \tag{3.2c}$$

$$a_S = \Gamma A\left(\frac{v_s\Delta x}{\Gamma}\right) + max(v_s\Delta x, 0) \tag{3.2d}$$

$$a_{P0} = \frac{\Delta x^2}{\Delta t} \tag{3.2e}$$

$$a_P = a_E + a_W + a_N + a_S + a_{P0}. \tag{3.2f}$$

In the expressions above, Γ [m^2/s] is the diffusion constant for the process of interest, the lower case subscripts e, w, n and s denote the quantity value at the eastern, western, nothern and southern face of cell P and $A(\zeta)$ is a function resulting from the kind of scheme that is used. The other variables were already introduced in the previous chapter.

In our work, the hybrid discretization was used:

$$A(\zeta) = \max(0, 1 - 0.5|\zeta|). \tag{3.3}$$

This scheme is an approximation of the analytical solution of the steady-state one dimensional convection-diffusion equation. Note that $A(\zeta) \geq 0$ for all P. Therefore, all coefficients in Equation 3.1 will be positive and Rule 1 of Section 2.2.4 is satisfied. Furthermore, it can be checked that also Rule 2 as well for this scheme.

The discretization equations for u, v, and T (the tilde for T and p and the subscript 0 below ν and κ is dropped from now) can be obtained from Equation 3.1 applied to Equation 2.6b, 2.6c and 2.6d. A few sidenotes have to be made first:

- For u and v underrelaxation as in Equation 2.37b with α is applied due to nonlinearity;
- Since a staggered grid was used, a grid point index like P and a face index (w for example) do not represent the same location in all three resulting equations (See Figure 2.6). These indices only define the locations in and around the unit cells of interest, which are thus shifted with respect to each other. For these ambiguities a superscript u or v has been added to some quantities to avoid confusion.

The resulting discretization equations are[1]:

$$\frac{a_P^u}{\alpha}u_P = a_E^u u_E + a_W^u u_W + a_N^u u_N + a_S^u u_S + a_{P0}^u u_{P0} + \frac{(1-\alpha)}{\alpha}a_P^u \bar{u}_P + (p_w^u - p_e^u)\Delta x \tag{3.4a}$$

$$\frac{a_P^v}{\alpha}v_P = a_E^v v_E + a_W^v v_W + a_N^v v_N + a_S^v v_S + a_{P0}^v v_{P0} + \frac{(1-\alpha)}{\alpha}a_P^v \bar{v}_P + (p_s^v - p_n^v)\Delta x + \beta g T_P \Delta x^2 \tag{3.4b}$$

$$a_P^T T_P = a_E^T T_E + a_W^T T_W + a_N^T T_N + a_S^T T_S + a_{P0}^T T_{P0}, \tag{3.4c}$$

[1] Note that these are the equations for the problem in the interior domain. For the cells near the boundaries the equations must be modified as done in Section 2.2.3.

where

$$a_E^u = \nu A\left(\left|\frac{u_e^u \Delta x}{\nu}\right|\right) + max(-u_e^u \Delta x, 0) \qquad a_E^v = \nu A\left(\left|\frac{u_e^v \Delta x}{\nu}\right|\right) + max(-u_e^v \Delta x, 0)$$
$$a_W^v = \nu A\left(\left|\frac{u_w^u \Delta x}{\nu}\right|\right) + max(u_w^u \Delta x, 0) \qquad a_W^u = \nu A\left(\left|\frac{u_w^v \Delta x}{\nu}\right|\right) + max(u_w^v \Delta x, 0)$$
$$a_N^u = \nu A\left(\left|\frac{v_n^u \Delta x}{\nu}\right|\right) + max(-v_n^u \Delta x, 0) \qquad a_N^v = \nu A\left(\left|\frac{v_n^v \Delta x}{\nu}\right|\right) + max(-v_n^v \Delta x, 0)$$
$$a_S^u = \nu A\left(\left|\frac{v_s^u \Delta x}{\nu}\right|\right) + max(v_s^u \Delta x, 0) \qquad a_S^v = \nu A\left(\left|\frac{v_s^v \Delta x}{\nu}\right|\right) + max(v_s^v \Delta x, 0)$$
$$a_{P0}^u = \frac{\Delta x^2}{\Delta t} \qquad a_{P0}^v = \frac{\Delta x^2}{\Delta t}$$
$$a_P^u = a_E^u + a_W^u + a_N^u + a_S^u + a_{P0}^u \qquad a_P^v = a_E^v + a_W^v + a_N^v + a_S^v + a_{P0}^v$$

$$a_E^T = \kappa A\left(\left|\frac{u_e \Delta x}{\kappa}\right|\right) + max(-u_e \Delta x, 0)$$
$$a_W^T = \kappa A\left(\left|\frac{u_w \Delta x}{\kappa}\right|\right) + max(u_w \Delta x, 0)$$
$$a_N^T = \kappa A\left(\left|\frac{v_n \Delta x}{\kappa}\right|\right) + max(-v_n \Delta x, 0)$$
$$a_S^v = \kappa A\left(\left|\frac{v_s \Delta x}{\kappa}\right|\right) + max(v_s \Delta x, 0)$$
$$a_{P0}^T = \frac{\Delta x^2}{\Delta t}$$
$$a_P^T = a_E^T + a_W^T + a_N^T + a_S^T + a_{P0}^T$$

The temperature and pressure term in the equations have been found by the discretization of the source term as follows

Integrating over the u grid
$$\int_t^{t+\Delta t}\int_s^n\int_w^e -\frac{\partial p}{\partial x}\,\mathrm{d}x\mathrm{d}z\mathrm{d}t = -\int_t^{t+\Delta t}\int_s^n p\Big|_w^e\,\mathrm{d}z\mathrm{d}t = (p_w^u - p_e^u)\Delta x\Delta t \tag{3.5}$$

Integrating over the v grid
$$\int_t^{t+\Delta t}\int_w^e\int_s^n -\frac{\partial p}{\partial z}\,\mathrm{d}z\mathrm{d}x\mathrm{d}t = -\int_t^{t+\Delta t}\int_w^e p\Big|_s^n\,\mathrm{d}x\mathrm{d}t = (p_s^v - p_n^v)\Delta x\Delta t \tag{3.6}$$

Integrating over the v grid
$$\int_t^{t+\Delta t}\int_s^n\int_w^e \beta g T\,\mathrm{d}x\mathrm{d}z\mathrm{d}t = \beta g T_P \Delta x^2 \Delta t, \tag{3.7}$$

where the Δt term in all three equations drops out in the final discretization equations, bacause it is left in the a_{P0} term after dividing the whole discretization equation by Δt as done in Section 2.2.1. Furthermore, piecewise constant profiles have been assumed by performing the final integration steps.

The remaining equation, the continuity equation (Equation 2.6a), does not contain p, but p is the last unknown. In Section 3.4 a plan will be discussed in order to solve for this problem.

3.3. Initialization of the grids

When using the application, the user places walls inside the interior and sets the boundary conditions. The grid points lying in this region will be the internal boundary points. As seen in Section 2.2.3, the cells near these selected boundaries (and the regular boundaries at the edge of the screen) must satisfy other equations than the interior cells. Therefore, the application must be able to know which cells are boundary cells and which are not: it has to know which equations hold in each cell.

The total domain is rectangular and its cells are rectangular (square in fact) as well. Since the structure is the same and repeated over the whole domain, an important advantage is that it turns out that the only information needed to assemble all equations on all three grids is the set of interior boundary grid points of the regular grid and information on the boundary conditions.

3.3.1. The ordered set approach for the standard grid

To give an indication how this works, consider the grid in Figure 3.2. The interior boundary cells are filled with a striped pattern and the interior cells are white. The boudary faces are thick black lines.

The cells are numbered in the way as shown in the figure (the black numbers) and form the initial grid G^*. Next, the interior boundary cells are deleted from the full grid and the remaining set of numbers is sorted from smallest to biggest. This set is called G. The index of the number in this ordered set of numbers is the new number representing the location of the cell in the grid. These are the blue numbers starting at zero in the figure[2]. This new numbering is the new representation of the standard grid, which is called lG. The l stand for location.

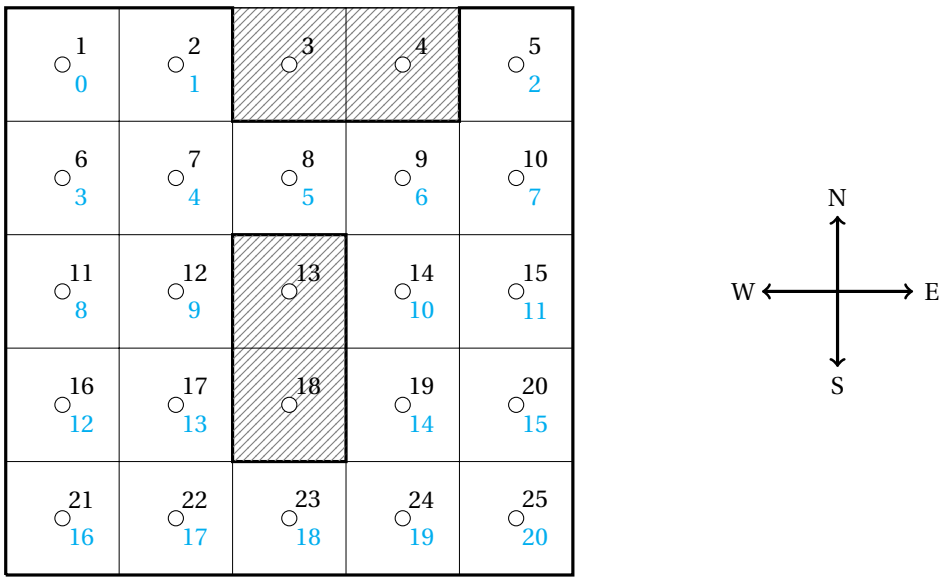


Figure 3.2: A 5x5 grid with 4 striped interior boundary cells and 21 interior grid cells. The north, east, south and west directions are also denoted.

For the algorithm to run, the coefficients of the linear equations must be calculated and the location of these coefficients in the matrices must be known (because solving a set of linear equations implies solving a matrix equation). So a distinction must be made in order to know whether a boundary equation or an interior equation holds for a location in the grid. Therefore the grid is split up into twelve sets:

- The grid points with a boundary at the North side: $lG_N = (0,1,2,5,6,18)$.
- The grid points with a boundary at the East side: $lG_E = (1,2,7,9,11,13,15,20)$.
- The grid points with a boundary at the South side: $lG_S = (5,16,17,18,19,20)$.
- The grid points with a boundary at the West side: $lG_W = (0,2,3,8,10,12,14,16)$.
- The four sets which are complementary to the first four with respect to lG. These are called lG_N^c, lG_E^c, lG_S^c and lG_W^c. These are the sets G_I^c that are not adjacent to a boundary in the I direction. For example: $lG_N^c = (3,4,7,8,9,10,11,12,13,14,15,16,17,19,20)$.
- The four sets of grid points which lie at the north, east, south and west side of the previous four. These are called lG_N^{c+}, lG_E^{c+}, lG_S^{c+} and lG_W^{c+}. These are the sets that are obtained from a set G_I^c taking one step into the I direction. For example: $lG_N^{c+} = (0,1,2,3,4,6,7,8,9,10,11,12,13,14,15)$.

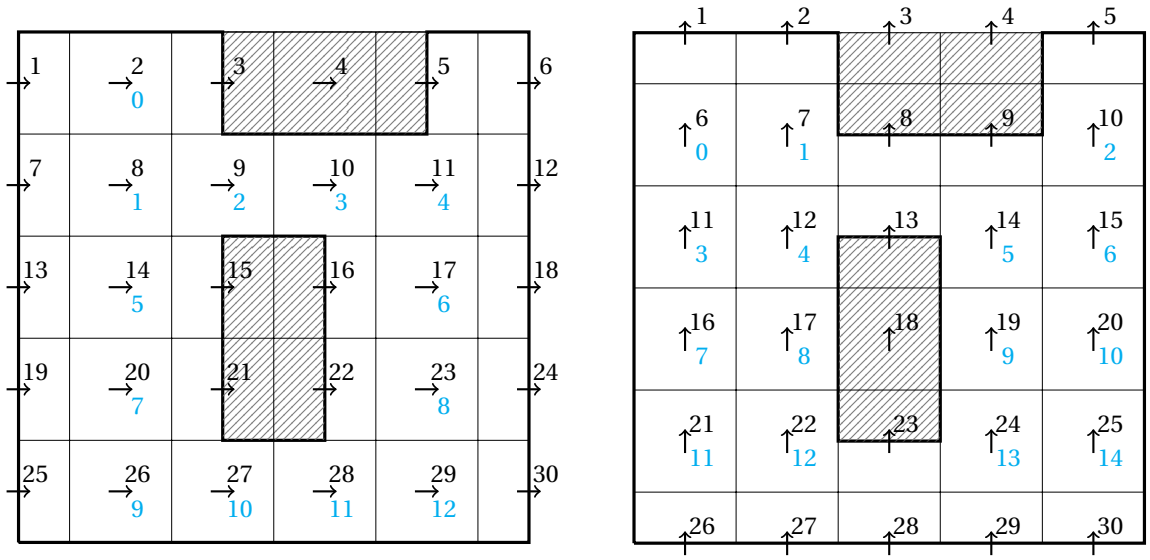
The sets lG_N, lG_E, lG_S, lG_W, lG_N^c, lG_E^c, lG_S^c and lG_W^c can be used for making distictions when calculating the coefficients and the sets lG_N^c, lG_E^c, lG_S^c, lG_W^c, lG_N^{c+}, lG_E^{c+}, lG_S^{c+} and lG_W^{c+} can be used when placing coefficients in the matrix equations. Finally, the boundary vector can be calculated using the sets lG_N, lG_E, lG_S and lG_W.

[2] JAVA is a zero based programming language, that is why we chose to start numbering at zero.

3.3.2. The ordered set approach for the u-and v-grid

Consider the u-grid version of Figure 3.2, shown in Figure 3.3a. The numbering of the original grid G_u^* already starts at the faces. In the same way as for the standard grid, a reduction can be made in order to form the u-grid G_u. Then the new representation of this grid is the set of blue numbered cells, lG_u. Again twelve sets are needed in order to assemble the equations.

For the v-grid G_v a similar numbering (starting at the top boundary) can be initiated as for the u-grid. The result of the example is shown in Figure 3.3b.



(a) From the u-grid perspective.

(b) From the v-grid perspective.

Figure 3.3: The initialized regular grid from the u-and v-grid perspective.The boundaries are differently defined, since gray interior boundary cells the regular grid are not adapting to other grids.

3.3.3. Images to the faces of the standard grid

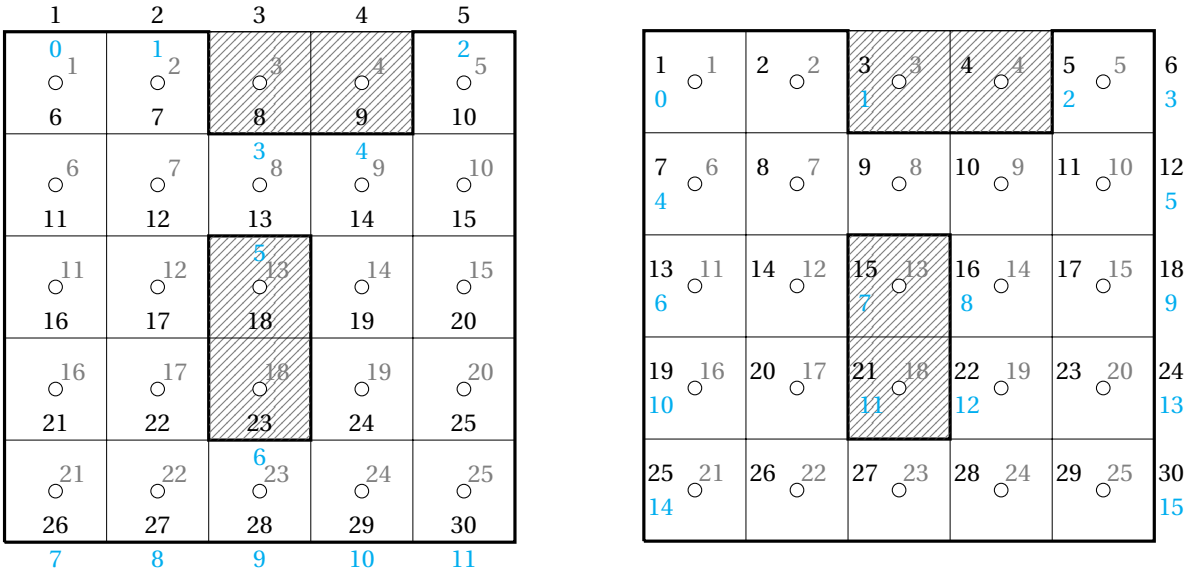
In order to calculate the coefficients, the velocities on the boundary faces must be known (and listed in sets) as well. Since every cell has a north, east, south and west side, for every direction there exists a bijection between the gridpoints and the face in that direction. Now we can see why the sets were ordered: for simplicity it is favourable if the bijective mapping perserves order.

The faces are split up twice. The first distinction is made between horizontal and vertical faces: the north and south faces form the horizontal faces and the east and west faces form the vertical faces. The second distinction is whether the faces are boundary faces or interior faces.

Consider the horizontal faces. These are numbered the same way as the v-grid. In terms of the original (black) numbers the faces of interest are:

- The north faces which lie on the boundary: $(1,2,5,8,9,23)$.
- The south faces which lie on the boundary: $(13,26,27,28,29,30)$.
- The north faces which lie in the interior: $(6,7,10,11,12,14,15,16,17,19,20,21,22,24,25)$.
- The south faces which lie in the interior: $(6,7,10,11,12,14,15,16,17,19,20,21,22,24,25)$.

The north boundary faces will be denoted as F_n and the South boundary faces as F_s. The sorted set obtained from the union between these two sets are the horizontal boundary faces F_{hb}. The locations of the numbers in this set is the new representation of the north and south faces: lF_n and lF_s. It can be seen immediately that bijection between the gridpoints near a north boundary and its faces, $f : lG_N \rightarrow lF_n$, is given by $f(lG_N(i)) = lF_n(i)$: the i^{th} index of the first set is directly linked to the i^{th} face index due to order perservation. The same can be done for the south boundary faces. The bijection between the interior faces is



(a) The horizontal faces

(b) The vertical faces

Figure 3.4: A 5x5 grid with 4 striped interior boundary cells and 21 interior grid cells. The horizontal faces are numbered left and the vertical faces are numbered right.

in both cases (north respectively south) the direct link between the elements of lG_N^c and lG_v respectively lG_S^c and lG_v with the same index[3].

When using the u-grid for the horizontal faces, we find the horizontal boundary faces F_{hb} which is divided into the East and West faces F_e and F_w.

Then, the total list of bijective images to the faces of the standard grid is:

$$lG_N \to lF_n \qquad lG_E \to lF_e \qquad lG_S \to lF_s \qquad lG_W \to lF_w$$
$$lG_N^c \to lG_v \qquad lG_E^c \to lG_u \qquad lG_S^c \to lG_v \qquad lG_W^c \to lG_u$$

Moreover, for the regular grid Dirichlet boundary conditions can hold for temperature. This is the part where the boundary information becomes important (while the other ordered sets can be constructed without any knowledge the boundary information). The boundary information tells which boundary faces have Dirichlet like information. The northern, eastern, southern and western boundary faces can be reduced to the sets lF_{nDir},lF_{eDir},lF_{sDir} and lF_{wDir} containing the indices of the faces where Dirichlet boundary conditions hold. Due to the the bijection between the faces and the cells near the grid, as mentioned above, the sets lG_{NDir}, lG_{EDir}, lG_{SDir} and lG_{WDir} can be obtained through taking the inverse image.

3.3.4. Images to the faces of the u-and v-grid

In the case of the u-and the v-grid, the situation is more complicated. One of the results is that we have more possibilities for a cell to be a boundary cell than in the case of the standard grid.

Consider the u-grid. Note that the east and west faces do not have multiple possibilities: only the north and south faces do. The different possibilities are shown in Figure 3.5. When calculating the coefficients for the linear equations, the boundary conditions on the faces in the figure must be satisfied. Since $u = 0$ holds on the boundaries for our research, all cases result in the same type of equation.

In the case of interior cells that are not near the boundary, the u and v values on the faces are obtained through interpolation of the values from the previous iteration:

- The east face u-values are obtained as the average of the value at P and E with respect to the u-grid.
- The west face u-values are obtained as the average of the value at P and W with respect to the u-grid.
- The north face v-values are obtained as the average of the value at the northeast grid point and the northwest grid point with respect to the u-grid.

[3]We will write $A \to B$ for the bijection between the elements of the same indices of A and B.

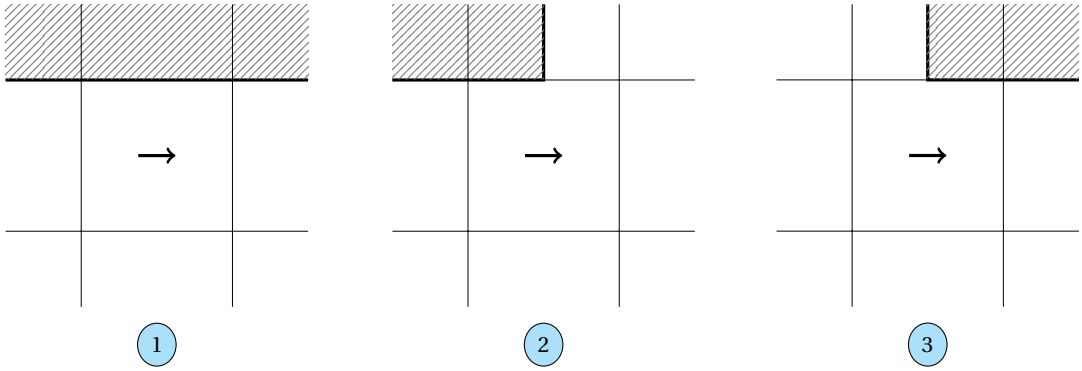


Figure 3.5: The three cases for boundary cells at the north. The same holds for the south boundaries, when mirroring the images in the horizontal plane.

- The south face v-values are obtained as the average of the value at the southeast grid point and the southwest grid point with respect to the u-grid.

The interpolation of the north and south face can be understood through Figure 3.6.

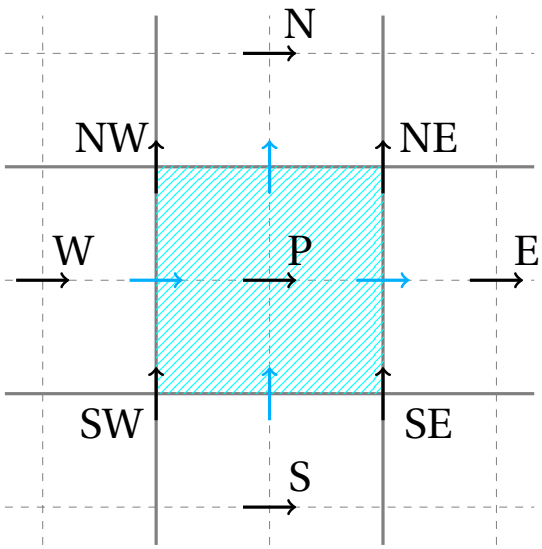


Figure 3.6: The interpolation of the boundary velocity on the u-grid: the average of the adjacent velocities. The v-grid is dashed.

The images from the lG_u to lG_v directing to the locations needed for interpolation can also be found using sets. These are listed in Appendix A.

Now consider the v-grid. Contrary to the u-grid, at the east and west boundary faces more possibilities become available. This is shown in Figure 3.7. Again the same type of equations arise for these cells due to the boundary condition $v = 0$. A similar interpolation scheme for the velocities holds for this grid. Furtheremore, the temperature is calculated through averaging the values at the north and the south face, as that is where the T-values are saved.

3.4. An algorithmic approach: SIMPLE

In the previous chapter the difficulties of the set of equations were highlighted: unphysical results for a single grid, nonlinearity, the continuity equation in combination with the pressure term and numerical limitations. The algorithm used to solve the problem[4] is known as SIMPLE, which stands for Semi-Implicit Method for Pressure Linked Equations. The flowchart in Figure 3.8 is a short overview of this algorithm. The steps are highlighted below [8].

[4]Note that the numerical limitations in Sections 2.3.4 and 2.3.5 are still not solved.

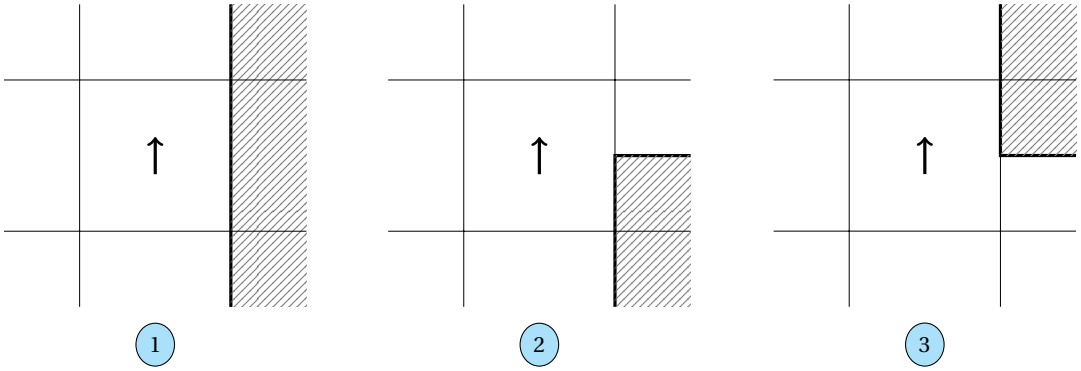


Figure 3.7: The three cases for boundary cells at the east. The same holds for the west boundaries, when mirroring the images in the vertical axis.

Step 1: Initialization
The first step in the algorithm is the initialization: setting up the framework for the algorithm to run. This has been discussed in Section 3.3.

Step 2: $\mathbf{t} = \mathbf{t} + \Delta\mathbf{t}$
As shown in Figure 3.8, there are two loops in the flowchart: one is the time iteration loop and the other loop is the one guaranteeing that the values for u, v, T and p satisfy Equation 2.6. At this stage in the algorithm, the next time step is initiated.

Step 3: Assemble and solve the equations for $\mathbf{u}^*$ and $\mathbf{v}^*$
The first step in the second loop of the SIMPLE algorithm is predicting a pressure, velocity and temperature field. We need to do this in order to calculate the coefficients for the discretization equations. Since the system starts at rest for our research, we can choose our initial condition, which gives zero everywhere as a first guess. In the following iterations the results of the previous iteration will be used as predicted value for the calculation of the coefficients. Notice that we are not solving the correct equations for u and v (which are Equations 3.4a and 3.4b)[5], but we are solving for the predicted values u^* and v^* using the equations:

$$\frac{a_P^u}{\alpha} u_P^* = a_E^u u_E^* + a_W^u u_W^* + a_N^u u_N^* + a_S^u u_S^* + a_{P0}^u u_{P0} + \frac{(1-\alpha)}{\alpha} a_P^u \bar{u}_P + (\bar{p}_w^u - \bar{p}_e^u)\Delta x \tag{3.8a}$$

$$\frac{a_P^v}{\alpha} v_P^* = a_E^v v_E^* + a_W^v v_W^* + a_N^v v_N^* + a_S^v v_S^* + a_{P0}^v v_{P0} + \frac{(1-\alpha)}{\alpha} a_P^v \bar{v}_P + (\bar{p}_s^v - \bar{p}_n^v)\Delta x + \beta g \bar{T}_P \Delta x^2, \tag{3.8b}$$

where the bars over the pressure and temperature denote that the values of the previous iteration (or the initial conditions in the case of the first iteration) are being used as predicted values.

Step 4: Pressure correction for $\nabla \cdot \mathbf{v} = 0$
During this step of the algorithm, the value for the pressure is being improved. Let us propose that the correct pressure is given by

$$p = \bar{p} + p', \tag{3.9}$$

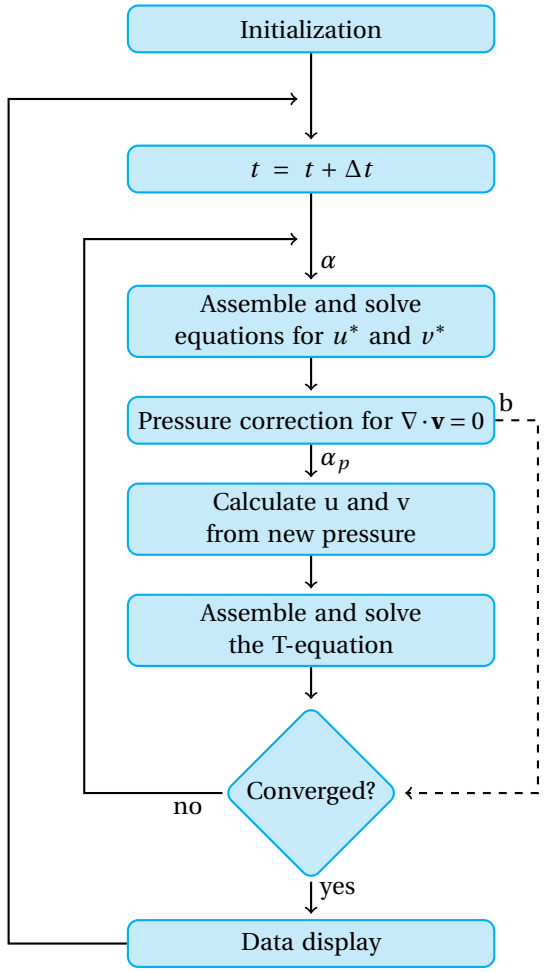
where p' is the pressure correction. Then we need to know how the velocity changes due to the correction. We write

$$u = u^* + u' \qquad\qquad v = v^* + v', \tag{3.10}$$

where u' and v' are the velocity corrections. If we substract Equation 3.4a from Equation 3.8a, we have something of the form:

$$\frac{a_P^u}{\alpha} u_P' = \sum_{nb} a_{nb} u_{nb}' + (p_w^{u\prime} - p_e^{u\prime})\Delta x. \tag{3.11}$$

[5]The coefficients of the u and v equations need the current u and v values to solve for u and v



variables
α : Relaxation factor for velocities
α_p: Relaxation factor for pressure
b : Convergence criterium

Figure 3.8: A flowchart for the SIMPLE algorithm [5].

Now, the SIMPLE algorithm states that the $\sum a_{nb} u'_{nb}$ term is dropped. The result is:

$$\frac{a_P^u}{\alpha} u'_P = (p_w^{u\prime} - p_e^{u\prime})\Delta x \Leftrightarrow u'_P = d_P^u (p_w^{u\prime} - p_e^{u\prime}), \tag{3.12}$$

where we used $\alpha = 0.5$ in our work and

$$d_P^u = \frac{\alpha \Delta x}{a_P^u}. \tag{3.13}$$

This can be done in a similar way for the vertical velocity v'. The resulting equations u and v can be deduced when rewriting Equation 3.10:

$$u = u^* + d_P^u (p_w^{u\prime} - p_e^{u\prime}) \tag{3.14}$$
$$v = v^* + d_P^v (p_s^{v\prime} - p_n^{v\prime}) \tag{3.15}$$

The remaining task is to find an equation for p'. This equation can be obtained through the continuity equation:

$$\nabla \cdot \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0. \tag{3.16}$$

Discretizing this equation gives

$$u_e - u_w + v_n - v_s = 0 \tag{3.17}$$

when integrating over a cell in the standard grid. Therefore, the subscripts which denote the locations are again rewritten for the standard grid. This can be done using Figure 2.6 again. Filling in Equation 3.14 and

Equation 3.15 gives

$$a_P^p p'_P = a_E^p p'_E + a_W^p p'_W + a_N^p p'_N + a_S^p p'_S + b, \tag{3.18}$$

where

$$a_E^p = d_e^u \tag{3.19a}$$
$$a_W^p = d_w^u \tag{3.19b}$$
$$a_N^p = d_n^v \tag{3.19c}$$
$$a_S^p = d_s^v \tag{3.19d}$$
$$a_P^p = a_E^p + a_W^p + a_N^p + a_S^p \tag{3.19e}$$
$$b = u_w^* - u_e^* + v_s^* - v_n^*. \tag{3.19f}$$

The boundary condition at all boundaries are zero valued Neumann boundary conditions, because at all boundary walls the velocity is known ($\mathbf{v} = 0$) for our work and therefore no correction is needed. From Equation 3.14 and Equation 3.15 then follows that the pressure differences must be zero and Neumann boundary conditions with zero gradient arise.

Again underrelaxation will be used:

$$p = p^* + \alpha_p p'. \tag{3.20}$$

For our research, $\alpha_p = 0.8$ is chosen.

A problem with the obtained matrix equation is that there is a zero eigenvalue present due to the zero gradient boundary condition and the symmetric character of the pressure correction matrix. The result is that the pressure correction can become arbitrarily large. This can be solved by adjusting one single linear equation: the equation of the upper left cell. This cell has two boundaries with Neumann boundary conditions. Changing these to zero valued Dirichlet boundary conditions, removes the singularity. The resulting equation for the upper left cell is:

$$c(a_E + a_S)p'_P = a_E p'_E + a_S p'_S + b, \tag{3.21}$$

where $c \gg 1$ and b as defined in Equation 3.19f. For our research we used $c = 10000$. Keeping the pressure of the first cell equal to zero at all times can be done without loss of generality, for we are merely interested in the pressure differences.

Step 5: Calculate u and v from new pressure

From Equation 3.14 and Equation 3.15 the new velocity field can be calculated. Note that this field is divergenceless.

Step 6: Assemble and solve the T-equation

Equation 3.4c can be assembled and solved, since the velocity field is now known.

Step 7: Converged?

At the end of every iteration in the second loop the result is a velocity field, a pressure distribution and a temperature field that satisfies the heat equation with a small error margin. If this is the case, the whole set of equations is solved and we have an solution for this time step. However, if this is not the case, the iteration starts again at step 3. Thus we want to know when the solution can be considered converged.

We know the solution is converged, if the pressure correction is very small. Consider Equation 3.18. This is actually a discretized version of a Laplace equation very similar to the result derived in Section 2.3.3. Therefore, for the correction to be small, the source term must be small[6]. In the case of Equation 3.18, the value for b in every gridpoint must be small. Let $\mathbf{b}$ be the vector containing all b-values. The criterium which will be used is that the solution is converged whenever

$$\|\mathbf{b}\|_\infty = \max(\mathbf{b}) \ll v_{\text{ref}} L, \tag{3.22}$$

where v_{ref} a characteristic velocity and L is a characteristic length. This criterium states that the maximum velocity flux of a single cell is much smaller than a charachteristic flux.

[6]We are using the source and not the correction values themselves, because of the zero eigenvalue mentioned in step 4.

For this situation studied in this work, the characteristic length is the height of the box. The characteristic velocity is a velocity which is typical for the flow. Consider the nondimensional form of Equation 2.6c [11]:

$$\frac{\partial \mathrm{v}}{\partial \mathrm{t}} + \mathrm{u}\frac{\partial \mathrm{v}}{\partial \mathrm{x}} + \mathrm{v}\frac{\partial \mathrm{v}}{\partial \mathrm{z}} = -\frac{\partial \mathrm{p}}{\partial \mathrm{z}} + \mathrm{Pr}\left(\frac{\partial^2 \mathrm{v}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{v}}{\partial \mathrm{z}^2}\right) + \mathrm{RaPrT}, \tag{3.23}$$

where

$$\mathrm{x} = \frac{x}{L} \qquad \mathrm{z} = \frac{z}{L} \qquad \mathrm{u} = \frac{uL}{\kappa} \qquad \mathrm{v} = \frac{vL}{\kappa} \qquad \mathrm{t} = \frac{t\kappa}{L^2}$$

$$\mathrm{p} = \frac{pL^2}{\kappa^2} \qquad \mathrm{T} = \frac{T - T_C}{\Delta T} \qquad \mathrm{Ra} = \frac{g\beta\Delta T L^3}{\nu\kappa} \qquad \mathrm{Pr} = \frac{\nu}{\kappa}$$

and T_C [K] the temperature of the cold wall. Note that the hydrostatic pressure p already is divided by the density. Furthermore, we see the non-dimensional numbers determining the flow are Ra and Pr.

Since buoyancy driven flow is being considered, the RaPrT term is mostly responsible for the advection term. Therefore,

$$\mathrm{u}\frac{\partial \mathrm{v}}{\partial \mathrm{x}} + \mathrm{v}\frac{\partial \mathrm{v}}{\partial \mathrm{z}} \simeq \mathrm{RaPrT} \Rightarrow 2\frac{\mathrm{v}_{\mathrm{ref}}^2}{\mathrm{L}_{\mathrm{ref}}} \simeq \mathrm{RaPrT}_{\mathrm{ref}} \Rightarrow \mathrm{v}_{\mathrm{ref}} \simeq \sqrt{\frac{\mathrm{RaPr}}{2}}, \tag{3.24}$$

where it was used that both $\mathrm{L}_{\mathrm{ref}}$ and $\mathrm{T}_{\mathrm{ref}}$ are of order 1. Using the definition of v and replacing the similarity sign with an equality sign gives

$$v_{\mathrm{ref}} = \sqrt{\frac{\mathrm{RaPr}\kappa^2}{2L^2}} = \sqrt{\frac{g\beta\Delta T L}{2}} \tag{3.25}$$

as an estimate for the typical value for the velocity. Let $f \in (0,1)$ then the convergence criterium states: if

$$\|\mathbf{b}\|_\infty < f\sqrt{\frac{g\beta\Delta T L^3}{2}} \tag{3.26}$$

is satisfied, then the solution has converged. $f = 10^{-3}$ is used for the calculations in our work.

Step 8: Data display
After every converged iteration set, the solution is shown on the screen, e.g. a pixelmap for a selected quantity of interest. Then the next iteration in the time loop starts and the loop starts again at step 2.

3.5. Implementing SIMPLE

In order to write an application, a programming language has to be chosen. JAVA will be used for programming the application because of the library called LIBGDX. This library is designed for games on mobile phone and functions as a platform which can compile the code in such a way that it can not only run on Android (JAVA based), but also on iOS (C++ based) and on HTML based systems. Furthermore, a desktop launcher is present as well, which allows for performing quick tests. Instead of deploying the code to Android or iOS, you can run and debug the application from a laptop.

Since LIBGDX is only a framework, an additional Integrated Development Environment (IDE) in which the code is written, is convenient. Android Studio was chosen for the following reasons:

- LIBGDX can be simply incorporated into this IDE;
- Android Studio has as built-in mobile phone emulator.

The second reason is the most important one. Since we have this emulator, the android version can be tested without installing the code on a physical device via a cable all the time.

Nevertheless, there is a disadvantage to JAVA. As mentioned in Section 2.3.5, the real bottleneck of the problem is the speed we can achieve. In other words the bottle neck is solving the linear equations. Unlike MATLAB, JAVA does not have a standard backslash operator which chooses the optimal way to solve the system of equations. We need to choose a linear algebra library which performs at least good enough to enable us to generate a realtime solution.

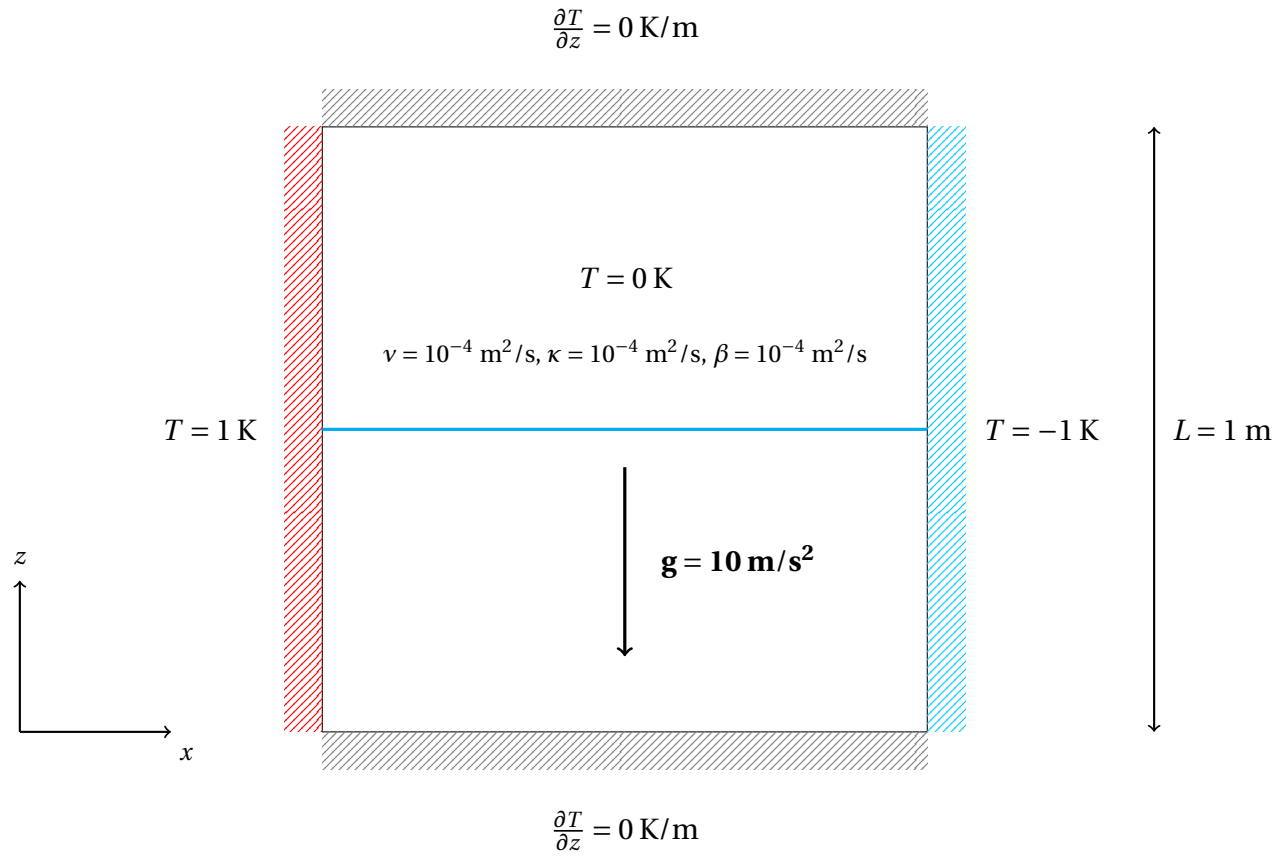


Figure 3.9: The problem investigated for the convergence check: a fluidum initially at $T = 0$ with $\nu = 10^{-4}$, $\kappa = 10^{-4}$ and $\beta = 10^{-4}$ is situated within a cavity. The east wall of the cavity is cold, the west side is hot and the north and south walls are insulated. The values on the blue line are compared to other simulation data.

PARRALLEL COLT is a collection of open source libraries for high performance scientific and technical computing written in JAVA with a focus on multi-threaded algorithms. Furthermore, PARRALLEL COLT can deal with large sparse matrices and perform quick LU decompositions, which is what we are looking for. Whether the library can compete with MATLAB is investigated.

3.6. Testing SIMPLE

For analysing runtime and accuracy, a test problem will be the fundament of the investigation: consider a fluidum initially at $T = 0\,\mathrm{K}$ with $\nu = 10^{-4}\,\mathrm{m^2/s}$, $\kappa = 10^{-4}\,\mathrm{m^2/s}$ and $\beta = 10^{-4}\,\mathrm{m^2/s}$, situated within a cavity as in Figure 3.9. The east wall of the cavity is cold, the west side is hot and the north and south walls are insulated. Simulation data over the blue line at half height is known [11] and this region the algorithm will be tested for accuracy.

Note that this problem has Gr$= 2 \cdot 10^5$, but the result turns out to give a laminar solution. The idea is to tackle a problem harder than other problems with smaller Gr number, but still obtain a real time solution. From this result, we will try to derive problems for water and air which are real time solvable.

3.6.1. Runtime analysis

The runtime analysis is split up into two parts: comparing the JAVA script to the MATLAB script and investigating for which time step sizes the script can give a real time result. In both cases the algorithm is running and solving the test problem on a $n \times n$ grid in the following way:

1. 50 time steps, starting at 0.02 s till 1.0 s with steps of 0.02 s, are considered.

2. For every step size, 20 consecutive time steps are being calculated. The durations are saved in a text file.

3. The runtime measurement for each step size is performed 3 times, because the cache memory being full can disturb the measurements. Taking the median of the 3 measurements for a specific time step, decreases the influence of the cache to the measurements.

4. The mean of the 20 resulting median values for each step size, estimates the average runtime τ [s] of the SIMPLE algorithm for this step size. The standard deviation in the 20 medians σ_τ [s] indicates the inconsistency of the SIMPLE algorithm for this step size.

Real time analysis

The goal of the runtime analysis is finding time step sizes for which the SIMPLE algorithm on a mobile phone can solve all physical variables on a $n \times n$ grid real time. We will be looking for the biggest n for which a $n \times n$ grid gives a real time result for the test problem, so the accuracy of the solution is optimized as well. In our work, we used a Nexus 5X (OS version 7.1.2) as the physical device.

Furthermore, we say a step size gives a real time simulation, if the runtime is smaller or equal to Δt for all runtimes (even the worst cases). Therefore, we deem a step size to give a real time solution, if $\tau + \sigma_\tau \leq \Delta t$ holds.

Next, the optimal step size is the step size for which the ratio

$$r = \frac{\tau}{\Delta t} \tag{3.27}$$

gives the minimum result and for which the uncertainty in the ratio is the smallest. The uncertainty in the ratio is estimated by [6]:

$$\sigma_r = \frac{\partial r}{\partial \tau}\sigma_\tau = \frac{\sigma_\tau}{\Delta t}. \tag{3.28}$$

The measurements are running on the Nexus 5X under two conditions: one time with other apps being activated (mail, whatsapp and internet running on the background), thus testing under standard conditions, and one time when all apps are closed, thus testing under optimal conditions. The testing continues until the biggest n is found.

MATLAB vs JAVA

If the biggest n giving a real time solution to the test problem has been found, the next step is checking the relative performance of compared to MATLAB: executing the runtime analysis for both the MATLAB and the JAVA script indicates whether the MATLAB matrix solver and PARALLEL COLT LU solver have similar solving times, e.g. whether we could look for another Java library or we cannot easily find better solvers. Measuring the runtime for multiple step sizes and not for one step size only, removes the effect of implementation differences that give a wrong view. The grid size for which the comparison is made, is be obtained from the real time analysis.

3.6.2. Real time solutions for water and air systems

If a real time solution for the test system has been found for a certain step size Δt_{rt}, we will try to extrapolate the result to the case for water and air. Again, we want to have a suitable time step size. In our research, the step size for water and air is obtained through the non-dimensional time of Equation 3.23: by taking the ratios of the non-dimensional times of the fluidum of interest to the test fluid, the time step can be converted. The conversion can be calculated via:

$$\Delta t = \frac{\kappa_{\text{test}}}{\kappa}\left(\frac{L}{L_{\text{test}}}\right)^2 \Delta t_{\text{rt}} = \frac{L^2}{\kappa}\Delta t_{\text{rt}} \cdot 10^{-4} \text{ s}. \tag{3.29}$$

Furthermore, we want to get an indication which water and air problems are in real time solvable. Therefore, a new restriction in the $(\Delta T, L)$ plane in Figure 2.1 is estimated. Not every arbitrarily small time step will give a real time solution, because every processor has a certain overhead for the script, a minimum time it takes to process. If the minimum time is referred to as Δt_{min}, the following restriction has to hold:

$$\Delta t \geq \Delta t_{min} \Rightarrow L^2 \geq \kappa \frac{\Delta t_{rt}}{\Delta t_{min}} \cdot 10^4 \text{ m}^2 \Leftrightarrow L \geq \sqrt{\kappa \frac{\Delta t_{rt}}{\Delta t_{min}}} \cdot 10^2 \text{ m}. \tag{3.30}$$

The obtained area in the $(\Delta T, L)$ plane is investigated for real time solutions and thus the estimations in Equations 3.29 and 3.30 are checked.

3.6.3. Error analysis

In order to check how accurate a generated (real time) solution is, the long term solution is compared to the results of a different numerical method as explained in Section 2.4. Data from Wan [11] will be used to make the comparison according to the error defined in Equation 2.48. This comparison data contains 9 vertical velocity values at equidistant locations on the blue line in Figure 3.9.

Furtheremore, using the boundary layer calculations in Section 2.3.4, we find that $n \geq 29$ should hold. Depending on the real time results of the test fluid (and water and air) the accuracy around the estimated minimum grid size and the grid size resulting from the real time analysis will be considered.

4

Results

4.1. The profiled JAVA code

The profiling results of the JAVA code is shown in Figures 4.1 and 4.2. These consist of two important threads: a thread solving a matrix equation and a thread initiating the time step and drawing the result. In these figures the *Invocation Count* is the number of times the method is allocated, the *Inclusive Time* is the time spent in this method and all methods called by this method and the *Exclusive Time* is the time spent in the allocated method only for a single thread. We will be looking at the exclusive times, unless inclusive time is mentioned explicitly.

In the result of the matrix solving thread in Figure 4.1, we see that most of the time in this thread (74.9% inclusive time) is spent on the *quickSolve* method. This is the method which implements the LU solving algorithm of the PARALLEL COLT library. All methods from *lu* till *searchFromTo* are part of the library[1]. Therefore, we cannot gain any speed from this other than reducing the amount of invocations. The method which comes second is *size* with 4.5% of the time. This method is used to get the sizes of the ordered sets (as defined in Section 3.3). The effect of improving this method is neglegible compared to the matrix solving algorithm.

In the other thread in Figure 4.2, we see that most of the time is used by the *doTimestep* method. The time spent in this method is mostly caused by for loops. The amount of for loops cannot be decreased easily, so we will not discuss this method any further. Furthermore, the *getQuick* method[2] and the *coeffcall* method occupying a significant part of the thread time. However, only the first one could be reduced by rewriting code, but the second one not, because the number of coefficients which are calculated by this method cannot be reduced (and the code calculating it cannot be simplified). Thus optimally, only 13.6% could be gained for this thread. For now, we have chosen not to rewrite the code for this thread, because the benifit will be small compared to finding faster solvers.

We can conclude that the code is real time results can only be achieved through fast solvers.

4.2. Results of the runtime analysis

The real time analysis has been performed for a 21×21 grid, because this grid size turns out be the real time limit for the Nexus 5X discussed in Section 3.6.1. The comparison between JAVA script compiled to the Nexus 5X, the MATLAB script and the JAVA script on the desktop application will be made in the following sections. The requirement of the minimum number of 29×29 cells (Section 2.3.4) will be discussed in Section 4.4.

4.2.1. Real time simulation on mobile phone

Figure 4.3 shows the average runtime of the JAVA script compiled to the Nexus with other apps running on the background (so under normal operating conditions). Figure 4.5 shows the runtime when only the solver is running and other programs on the phone are turned off. The ratio of the simulated step size to the physical step size is shown in Figure 4.4 and 4.6 (with other apps running on the background and without them). The errorbars indicate the range of the time it takes to fulfill a time step in the runtime figures and the uncertainty in the real time ratio in Figure 4.4 and 4.6.

[1] We can also see that the exclusive times of these methods sum op to approximately the inclusive time of *quickSolve*.
[2] Note that this method is allocated in both threads and thus has nothing to do with the matrix solving thread.

Name	Invocation Count	Inclusive Time (µs)		Exclusive Time (µs)	
Thread GLThread 99		68.497.298	100,0%		
GLThread 99.	1	68.497.298	100,0%	6.548.861	9,6%
com.mygdx.cfdsolver.upSparMat2D.quickSolve	1	51.271.393	74,9%	291	0,0%
cern.colt.matrix.tdouble.algo.SparseDoubleAlgebra.lu	1	51.254.087	74,8%	240	0,0%
cern.colt.matrix.tdouble.algo.decomposition.SparseDoubleLUDecomposition.<init>	1	51.253.847	74,8%	1.048	0,0%
edu.emory.mathcs.csparsej.tdouble.Dcs_lu.cs_lu	1	51.248.995	74,8%	901.939	1,3%
edu.emory.mathcs.csparsej.tdouble.Dcs_spsolve.cs_spsolve	60	49.622.649	72,4%	209.380	0,3%
edu.emory.mathcs.csparsej.tdouble.Dcs_reach.cs_reach	60	49.386.800	72,1%	1.523.023	2,2%
edu.emory.mathcs.csparsej.tdouble.Dcs_dfs.cs_dfs	178	44.137.424	64,4%	19.731.145	28,8%
edu.emory.mathcs.csparsej.tdouble.Dcs_util.CS_MARKED	31.858	16.059.938	23,4%	16.059.938	23,4%
edu.emory.mathcs.csparsej.tdouble.Dcs_util.CS_MARK	4.765	7.326.350	10,7%	4.802.924	7,0%
cern.colt.matrix.tdouble.DoubleMatrix2D.get	1.678	5.556.443	8,1%	2.337.428	3,4%
edu.emory.mathcs.csparsej.tdouble.Dcs_util.CS_UNFLIP	4.091	4.630.455	6,8%	3.339.384	4,9%
edu.emory.mathcs.csparsej.tdouble.Dcs_util.CS_FLIP	7.520	3.814.497	5,6%	3.814.497	5,6%
cern.colt.matrix.tdouble.impl.SparseCCDoubleMatrix2D.getQuick	1.678	3.219.015	4,7%	1.918.189	2,8%
cern.colt.list.tint.AbstractIntList.size	6.245	3.089.843	4,5%	3.089.843	4,5%
cern.colt.matrix.tdouble.impl.SparseCCDoubleMatrix2D.searchFromTo	1.678	1.300.826	1,9%	1.300.826	1,9%
cern.colt.list.tint.IntArrayList.getQuick	1.678	1.223.552	1,8%	1.223.552	1,8%
cern.colt.list.tint.IntArrayList.get	1.678	786.834	1,1%	786.834	1,1%
java.lang.Math.abs	1.512	721.496	1,1%	721.496	1,1%
edu.emory.mathcs.csparsej.tdouble.Dcs_util.CS_CSC	421	142.826	0,2%	142.826	0,2%
com.mygdx.cfdsolver.upSparMat2D.<init>	1	20.372	0,0%	317	0,0%
cern.colt.matrix.tdouble.impl.SparseCCDoubleMatrix2D.<init>	1	20.055	0,0%	2.223	0,0%

Figure 4.1: The profiling results for the JAVA code for a the thread in which a matrix is solved. The percentages indicate the part of the total time the script uses to excecute a method. The *Invocation Count* is the amount of times the method is allocated, the *Inclusive Time* is the time spent in this method and all methods called by this method and the *Exclusive Time* is the time spent in the allocated method only.

We see in Figure 4.3 and Figure 4.5 that there is an overhead, which causes the smallest time step not to achieve the real time limit. However, when time steps become too big, a real time result is not realizable either for two reasons: more iterations per time step are needed and the cache of the phone must be emptied more often to make room for the new matrices, causing extra time and eventually inconsistency in runtime (thus large error bars). This will become worse, since the chance of divergence grows as the time step grows. We see that the real time limit can be achieved with and without extra apps running on the background by choosing a time step size in the middle. $\Delta t = 0.1$ s is the best choice for the time step for this grid size for two reasons: Figure 4.4 and Figure 4.6 show that for this time step a real time solution is always possible and the time it takes to fullfil a timestep is the most consistent (small error bar).

4.2.2. Runtime in MATLAB vs JAVA

Figure 4.7 shows the average runtime of the MATLAB script for different time steps. These results for the JAVA script is shown in Figure 4.8. The real time ratios for both plots are shown in Figure 4.9 respectively 4.10. Again, the errorbars are included in the plot to see the dispersion in the simulation time and the uncertainty in the ratio. The shape of the runtime curves is quite similar. The only difference is that the MATLAB runtimes remain low longer as the JAVA runtimes start increasing from the start. This is caused by the way the algorithms have been programmed and the way the cache memory is used by the two scripts. For example, in the MATLAB script vectorization is used which is not possible in JAVA and some boundary vectors are being calculated a little different in both scripts. Furthermore, note that JAVA is faster than MATLAB for small step sizen, so MATLAB has more overhead.

Although profiling has not been excecuted on the MATLAB script, we can assume that the matrix solving is the most time-demanding as for the case of the JAVA script. Therefore, we can conclude that our solver can compete with MATLAB for a 21x21 grid. So the optimised solvers in MATLAB will probably give not very much benefit for real time purposes.

4.3. A mobile real time CFD solver

So far we have seen that a real time solution is realizable for the test problem. For other problems, e.g. with objects within the domain, the test problem can still be solved real time with the step size $\Delta t_{rt} = 0.1$ s. A Nexus 5X emulator screenshot of the temperature simulation for the test system with additional insulated walls in the interior is shown in Figure 4.11.

Nevertheless, the solver does not only work for the test problem, but for water and air at $T = 287$ K (15°C) and $p = 1$ atm as well. The only difference is the time step which has to be chosen. Using the time step scaling

Name	Invocation Count	Inclusive Time (µs)		Exclusive Time (µs)	
Thread GLThread 103		68.229.135	100,0%		
GLThread 103.	1	68.229.135	100,0%	2.835	0,0%
com.badlogic.gdx.backends.android.AndroidGraphics.onDrawFrame	2	68.209.738	100,0%	2.445	0,0%
com.mygdx.cfdsolver.MyGdxCFDsolver.render	2	68.169.414	99,9%	736	0,0%
com.mygdx.cfdsolver.cfd.doTimestep	1	68.126.985	99,9%	19.560.446	28,7%
cern.colt.matrix.tdouble.impl.SparseCCDoubleMatrix2D.getQuick	8.657	15.010.396	22,0%	9.297.358	13,6%
com.mygdx.cfdsolver.cfd.coeffcal	3.720	13.987.819	20,5%	7.582.237	11,1%
cern.colt.list.tint.IntArrayList.get	12.838	6.930.189	10,2%	6.930.189	10,2%
cern.colt.list.tint.AbstractIntList.size	10.569	6.013.667	8,8%	6.013.667	8,8%
cern.colt.matrix.tdouble.impl.SparseCCDoubleMatrix2D.searchFromTo	8.657	5.713.038	8,4%	5.713.038	8,4%
java.lang.Math.max	7.440	4.055.477	5,9%	3.986.277	5,8%
cern.colt.matrix.tdouble.DoubleMatrix2D.get	1.399	3.780.715	5,5%	1.144.591	1,7%
cern.colt.matrix.tdouble.DoubleMatrix1D.get	1.860	2.492.289	3,7%	1.667.450	2,4%
cern.colt.list.tint.AbstractIntList.contains	939	2.384.430	3,5%	887.578	1,3%
java.lang.Math.abs	3.720	2.350.105	3,4%	2.350.105	3,4%
cern.colt.list.tint.IntArrayList.indexOfFromTo	939	1.496.852	2,2%	912.566	1,3%
cern.colt.matrix.tdouble.impl.DenseDoubleMatrix1D.getQuick	1.860	824.839	1,2%	824.839	1,2%
java.lang.Math.pow	931	601.616	0,9%	601.616	0,9%
cern.colt.list.AbstractList.checkRangeFromTo	939	584.286	0,9%	584.286	0,9%
java.lang.Double.doubleToLongBits	60	69.200	0,1%	47.382	0,1%
com.android.internal.os.LoggingPrintStream.println	1	39.897	0,1%	446	0,0%
com.android.internal.os.LoggingPrintStream.flush	1	38.158	0,1%	899	0,0%

Figure 4.2: The profiling results for the JAVA code for a thread, which does initiates the time steps and draws the result to the screen. The percentages indicate the part of the total time the script uses to excecute a method. The *Invocation Count* is the amount of times the method is allocated, the *Inclusive Time* is the time spent in this method and all methods called by this method and the *Exclusive Time* is the time spent in the allocated method only.

as mentioned in Section 3.6.2, we can rewrite the time step into a more suitable one:

$$\Delta t = \frac{L^2}{\kappa} \cdot 10^{-5} \text{ s}. \tag{4.1}$$

As we saw in Figures 4.3 and 4.5, the algorithm has an overhead. Therefore, not all systems will be solvable. The class of real time solvable problems is bounded by a certain minimum time as mentioned in Section 3.6.2. Using Figure 4.3, the estimated bound is given by:

$$\Delta t \geq 0.1 \text{ s} \Rightarrow L \geq \sqrt{\kappa} \cdot 10^2 \text{ m}. \tag{4.2}$$

In the case of water, this inequality gives $L \geq 0.0369$ m as a result. For air, the result is $L \geq 0.4488$ m.

Comparing this new requirement with the class of simple problems, we see that air cannot be solved in real time. The number of $(\Delta T, L)$ combinations for water drops, but is should remain real time solvable for the correct input.

Nevertheless, looking for $(\Delta T, L)$ pairs within the laminar regime of water has not given any result yet[3]. The assumption in Equation 4.2 does not seem to hold in the case of water and air. Especially for smaller characteristic lengths, calculating a time step takes longer. Therefore, a correct view of the problem set for which we can calculate the solution real time cannot be given. Finding a better estimate for the lower bound is left for further research.

4.4. Results of the error analysis

The long term velocity solutions of the algorithm applied to the test problem are shown in Figures 4.12, 4.13 and 4.14. These are the solutions for a 11×11 grid, a 19×19 grid and a 29×29 grid. The result for a 21×21 grid is shown as well in Figure 4.15.

The expectation for the problem was that the grid needed at least an 29×29 grid (Section 3.6.3). In Figure 4.14 we can see that the boundary layer is indeed visible. For a too small grid such as an 11×11 grid, we can clearly see in Figure 4.12 that the boundary layer cannot be seen at all. For grids bigger than 19×19 the boundary layer starts being visible. This can be seen in Figure 4.13. Although the cells adjacent to the east and west boundary are only a little smaller than the bulk velocities, the velocities adjacent to the north and south boundary do feel the boundary quite well: the result gives an acceptable approximation. From this grid size the boundary layer starts being better visible. The results will be similar to result on the 29×29 grid. We can see this as well for solution on the 21×21 grid, which was shown to be real time solvable.

[3] For air neither.

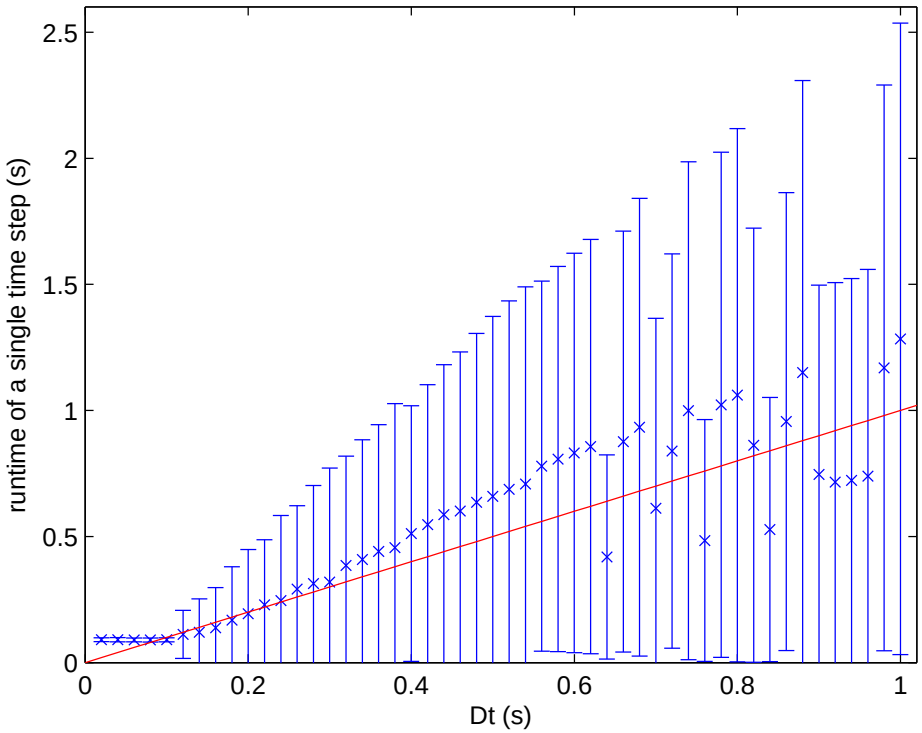


Figure 4.3: The average runtime of the CFD calculation of the test problem on a 21×21 grid of the JAVA script on the Nexus 5X for different step sizes with background CPU usage. The error bars indicate the range of the runtimes per stepsize and thus give an indication how consistent the average actually is. The red line is the real time line: having runtimes below the line indicates a real time simulation.

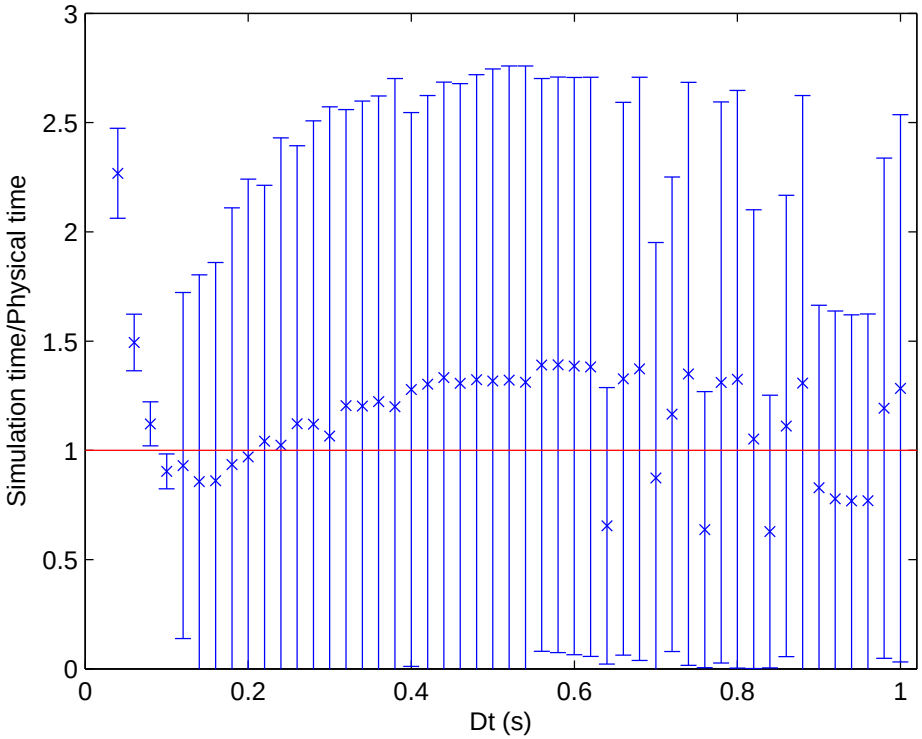


Figure 4.4: The results for the real time analysis on the Nexus 5X like Figure 4.3, but now for the ratio of the simulation to physical time.

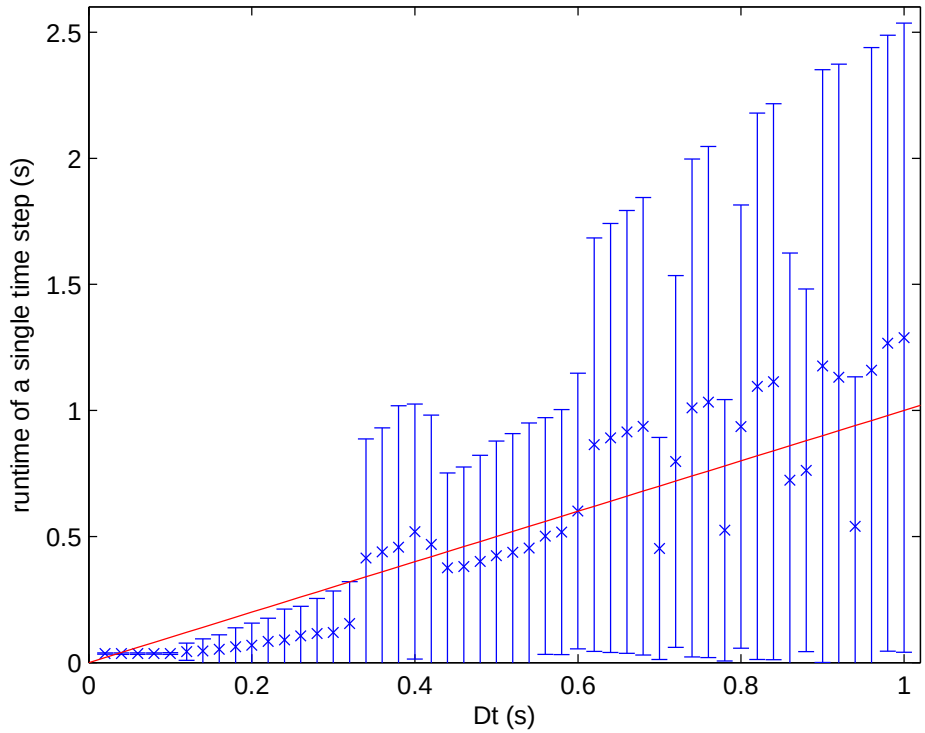


Figure 4.5: The average runtime of the CFD calculation of the test problem on a 21×21 grid of the JAVA script on the Nexus 5X for different step sizes without background CPU usage. The error bars indicate the range of the runtimes per stepsize and thus give an indication how consistent the average actually is. Again, the red line is the real time line.

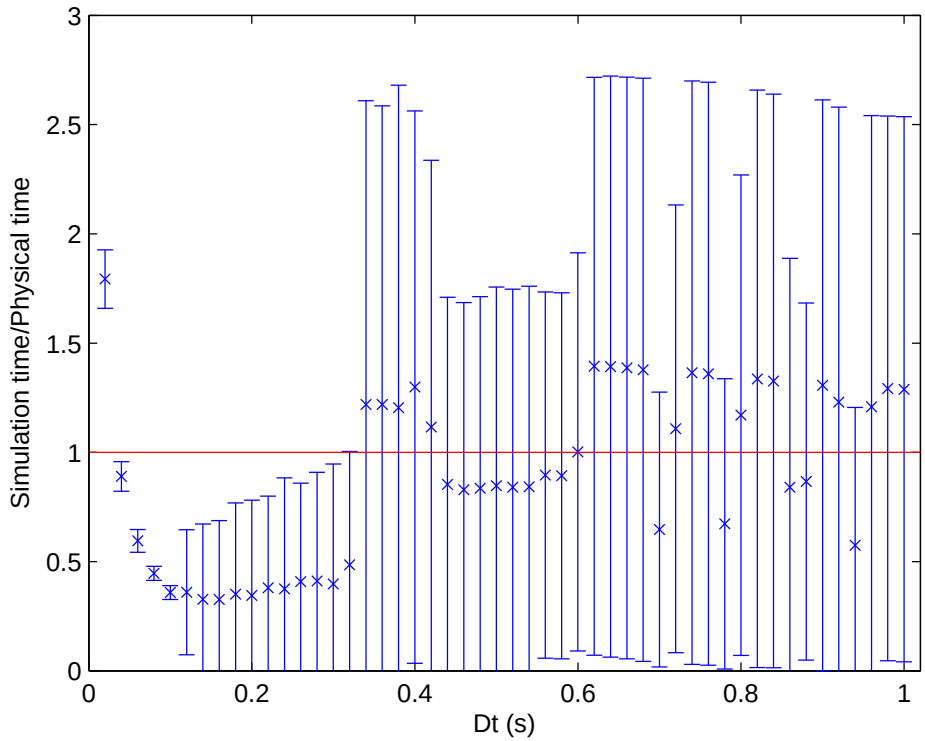


Figure 4.6: The results for the real time analysis on the Nexus 5X like Figure 4.5, but now for the ratio of the simulation to physical time.

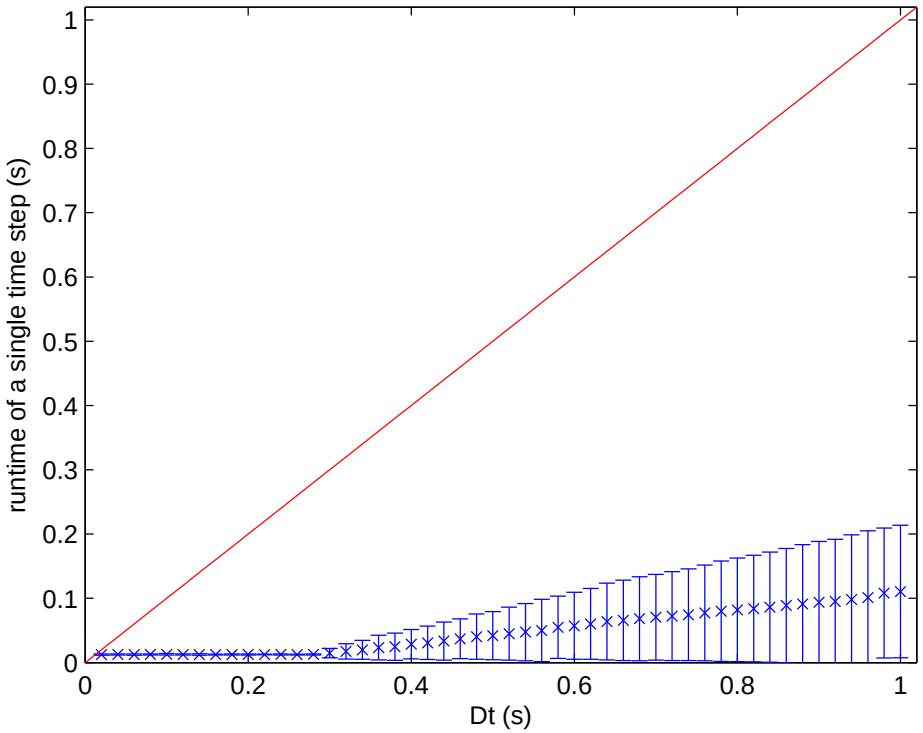


Figure 4.7: The average runtime of the CFD calculation of the MATLAB script for different step sizes. The error bars indicate the range of the runtimes per stepsize and thus give an indication how consistent the average actually is.

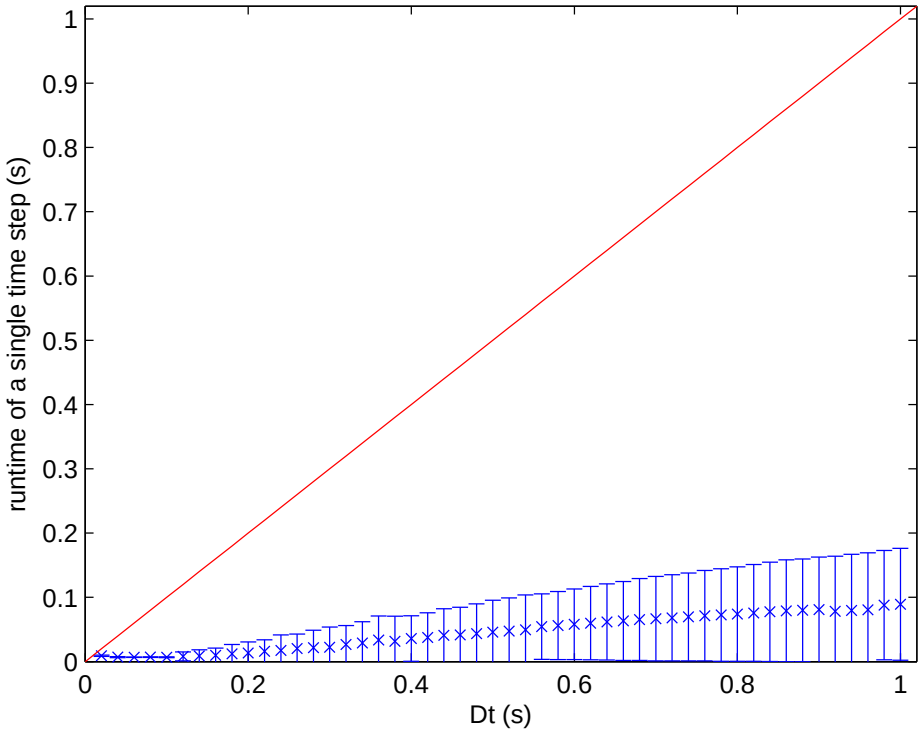


Figure 4.8: The average runtime of the CFD calculation of the JAVA script on the desktop for different step sizes. The error bars indicate the range of the runtimes per stepsize and thus give an indication how consistendt the average actually is.

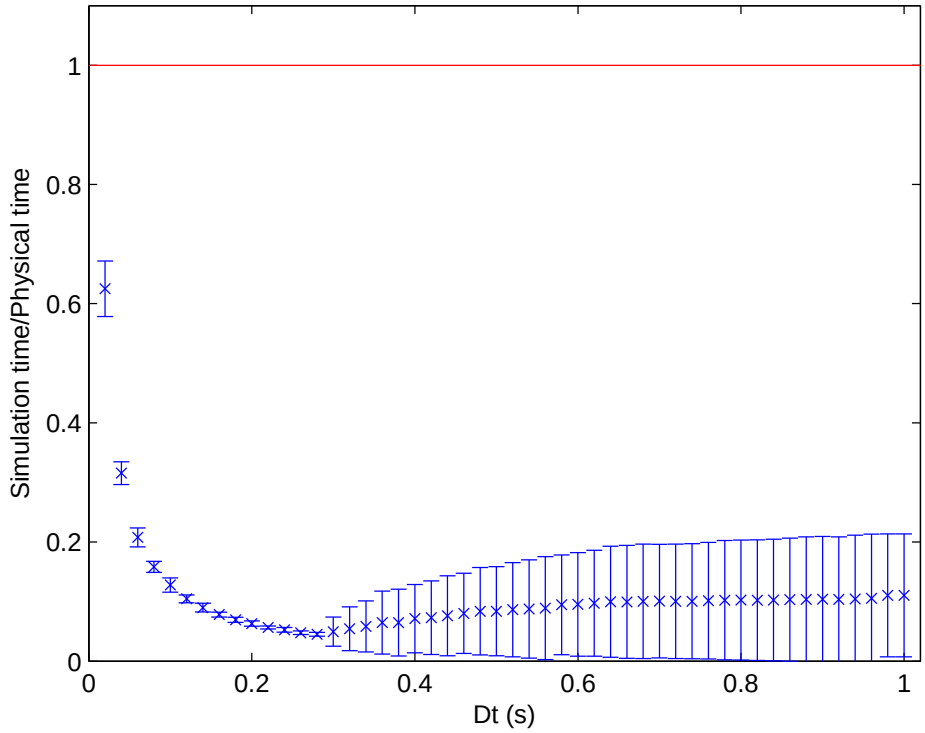


Figure 4.9: The ratio of the average simulation time of the MATLAB script to physical timestep for different step sizes.

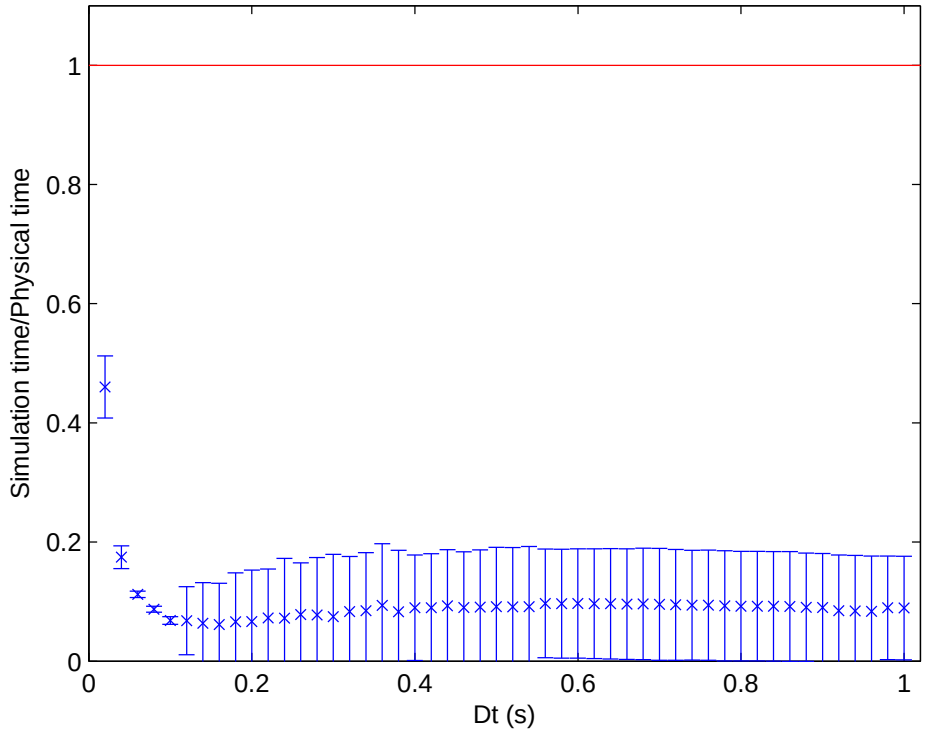


Figure 4.10: The ratio of the average simulation time of the JAVA script to physical timestep for different step sizes.

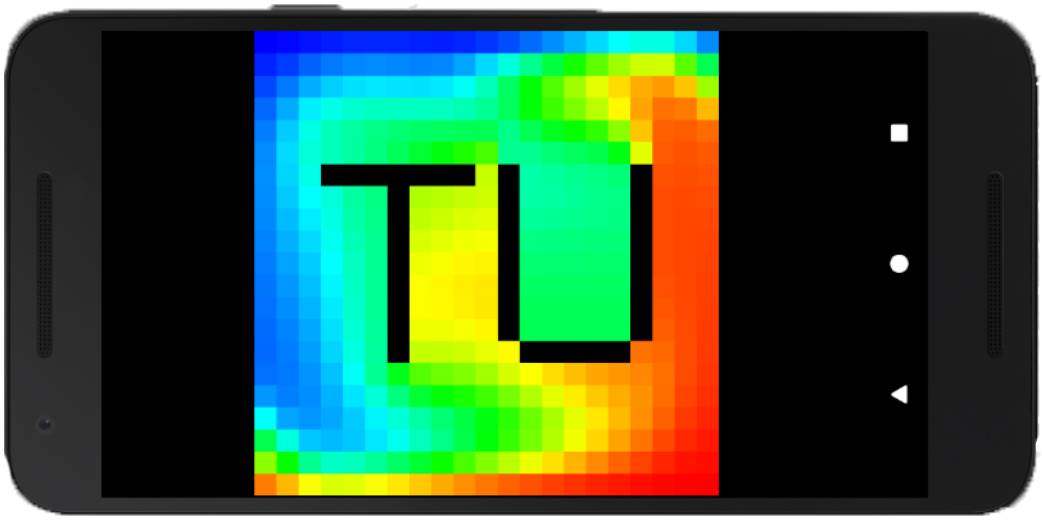


Figure 4.11: A screenshot of a running simulation of the test system. The south wall is hot (red), the north one is cold (blue) and the rest of the walls along with the TU letters are insulated.

Now, for the estimation of the relative error in the real time solution and the relative error in the algorithm in general, we are using the method described in Section 3.6.3. The relative errors of the long term solution of $n \times n$ grids are plotted in Figure 4.16. The n vary from 19 till 33 with step sizes of 2 and from 33 till 57 with step sizes of 6. A quadratic polynomial fitted to the data is shown in Figure 4.16 as well. The fit is given by $e = 22.92\Delta x^2$.

We see that the error estimates lie around the fitted curve. For the errors of grids smaller than 29×29, a little distortion could be expected, because the boundary layer is not quite there yet. However, for even larger grids the same pattern in the relative error can be seen. We conclude that the boundary layer is not the cause, because the distortion hold for bigger grids as well.

One reason for this phenomenon is that the solution does not seem to become stationary in the measurement time: the solution oscillates around the the reference data. Figure 4.17 shows some of these oscillations. Nevertheless, we can also see that the error has become quite steady. Therefore, we can conclude that the estimated error gives a reasonable view. From these results we can conclude that the relative error in the 21x21 grid is approximately 5.4% and the error decreases with order 2.

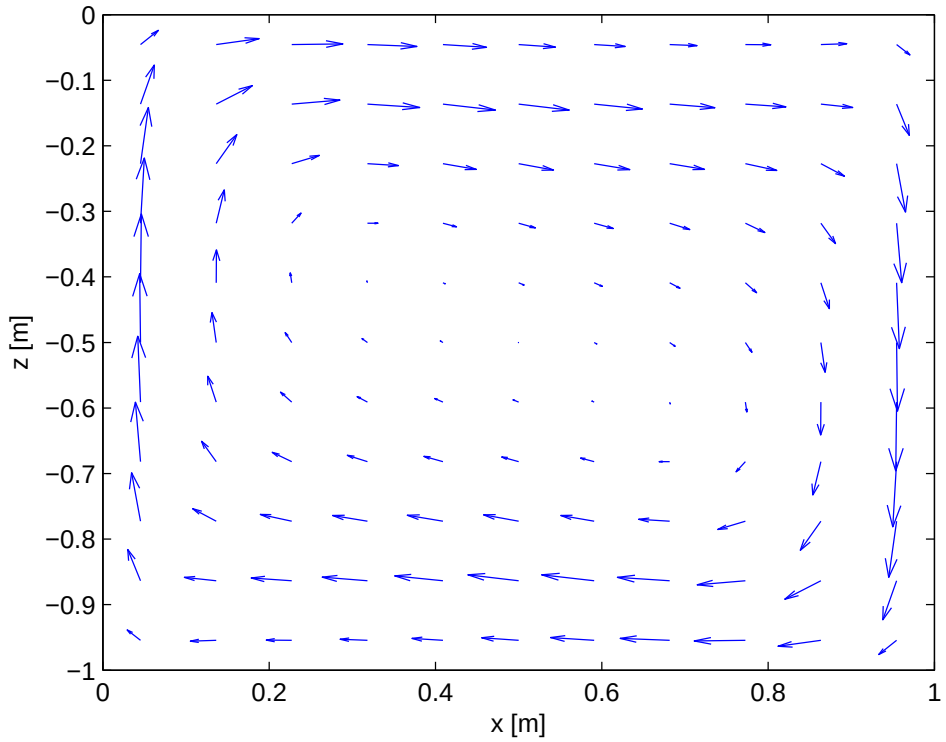


Figure 4.12: The solution after long time of the test problem on a 11x11 grid.

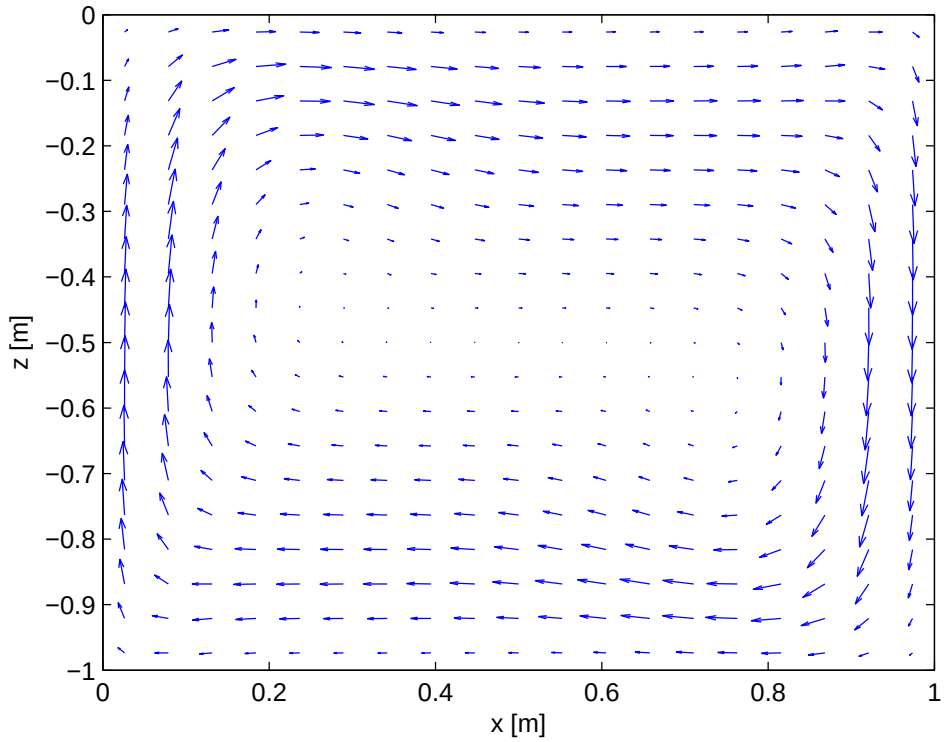


Figure 4.13: The solution after long time of the test problem on a 19x19 grid.

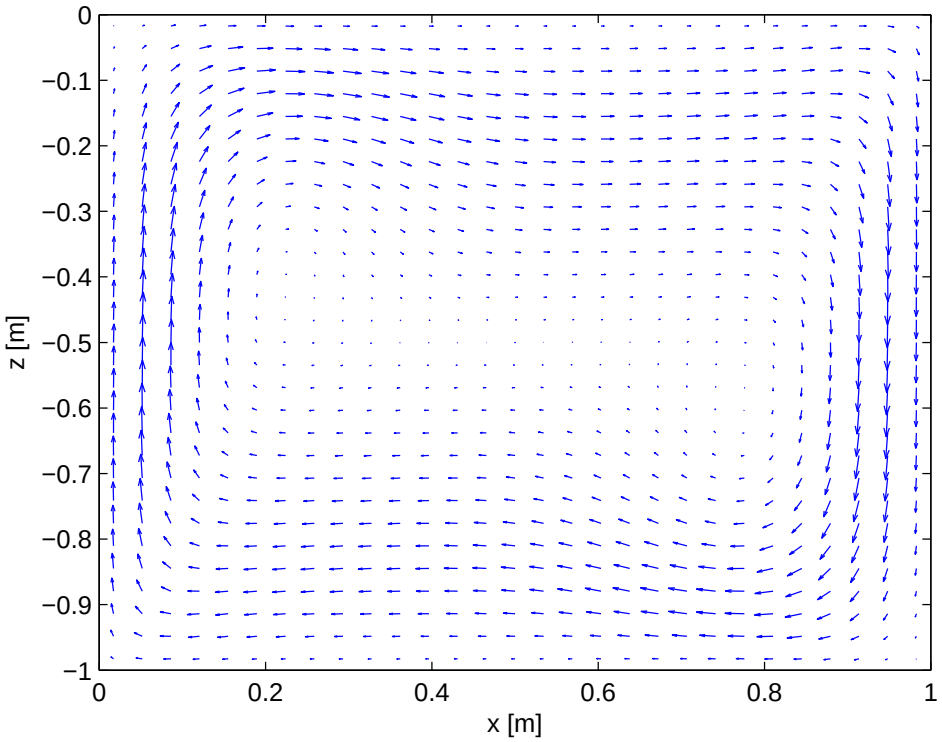


Figure 4.14: The solution after long time of the test problem on a 29x29 grid.

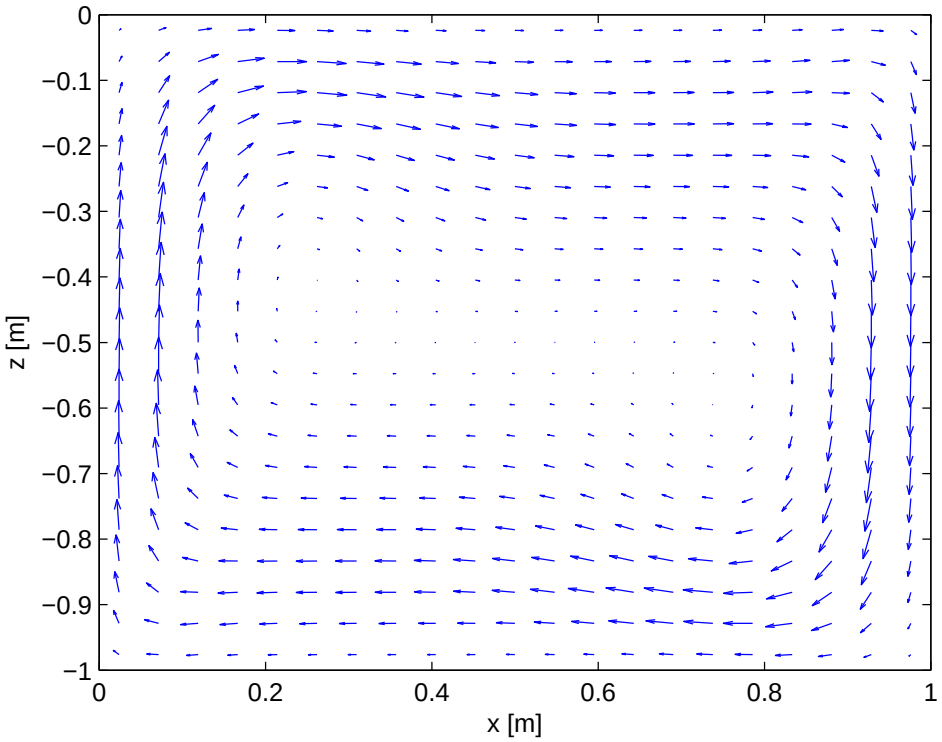


Figure 4.15: The solution after long time of the test problem on a 21x21 grid.

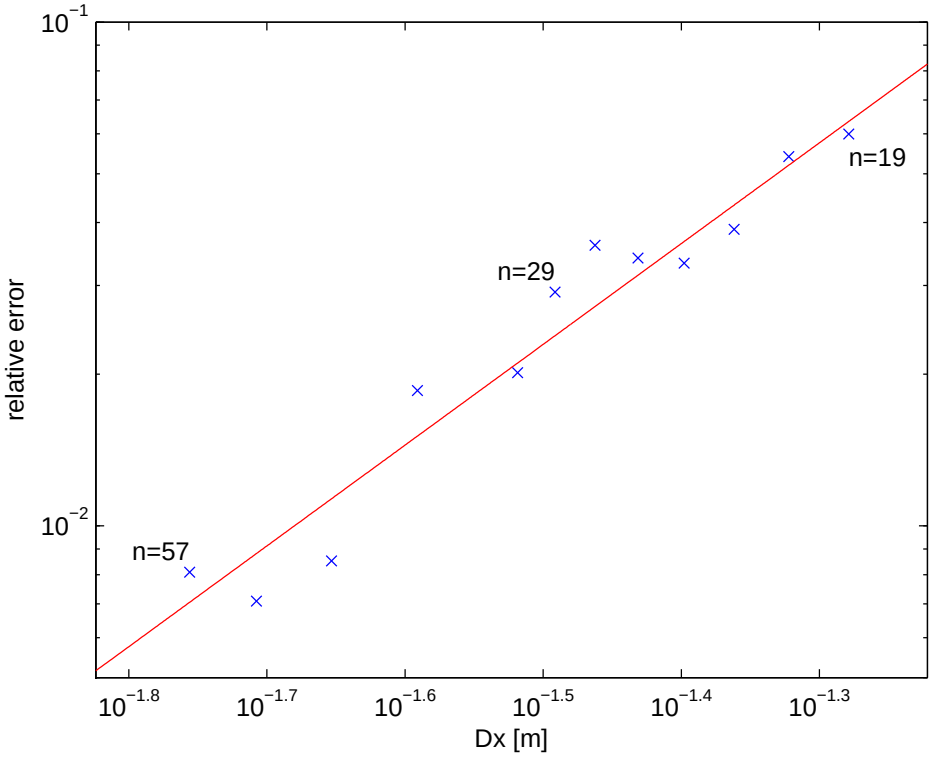


Figure 4.16: The estimated error of the algorithm for different step sizes: the rightmost point stands for $n = 19$ and the rightmost one for $n = 57$. A quadratic fit is shown as well.

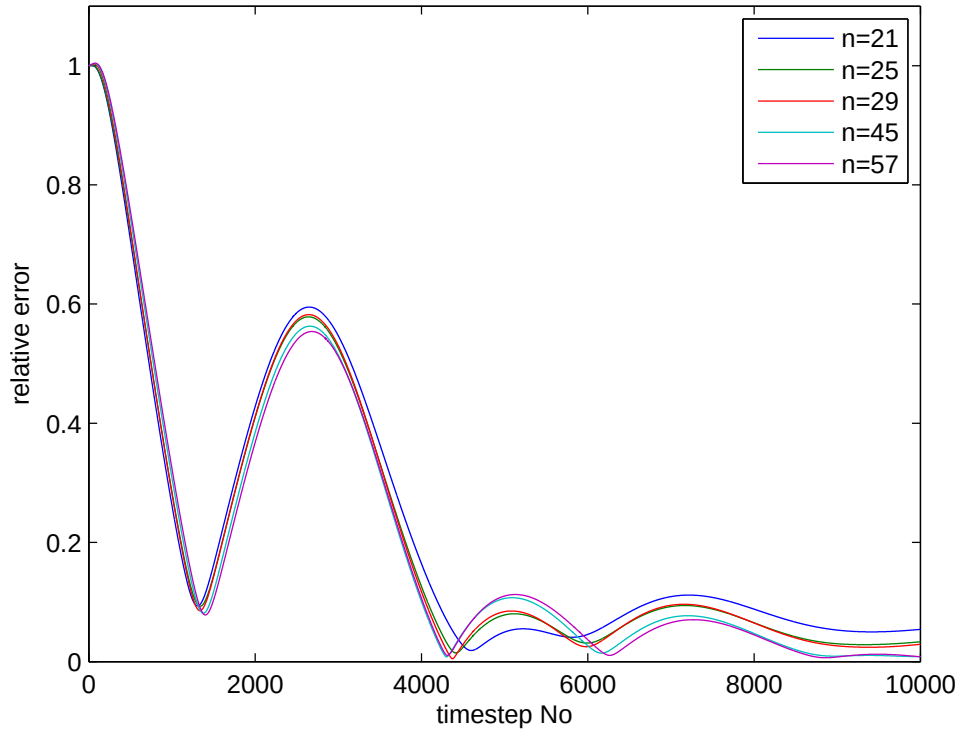


Figure 4.17: The relative distance between the simulated data and the reference data for all measured time steps and the grids with $n = 21$, $n = 25$, $n = 29$, $n = 45$ and $n = 57$.

5

Conclusions and Recommendations

5.1. Conclusions

In order to make a CFD solver for a smartphone, a JAVA application has been made, which has been tested on runtime performance and and accuracy. Our goal is to make a CFD solver that gives a real time solution with highest possible accuracy.

In order to check whether the application could gain much more speed, the JAVA code has been profiled. From the results, we found that the only way of improving the speed of the implemented algorithm is through faster matrix solvers.

For the test problem, we were able to simulate real time on a Nexus 5X on a 21×21 grid with an accuracy of 5.4%, when using the SIMPLE algorithm solving the physical parameters. We also found an optimal time step $\Delta t_{rt} = 0.1$ s for the test problem. For bigger grids, a real time solution is not longer possible, but the relative error decreases with second order.

Comparing the runtime of the JAVA script (for the test problem on a 21×21 grid) on desktop to code converted to MATLAB, we found that the algorithms in JAVA and MATLAB have about the same solving speed. MATLAB even had higher overhead. We concluded that the speed of a JAVA script cannot be improved much, since the bottleneck in the solving speed is in the matrix solvers. These solvers have already been optimized in MATLAB. Thus, a significantly faster CFD solver implementing the SIMPLE algorithm does probably not exist.

For other problems with the test fluid and the obtained time step, real time solutions can be found as well. The results have been extrapolated to the case of water and air. For air, finding real time solutions were predicted to be impossible. For water, there should be problems that give real time solutions. Nevertheless, when using the result extrapolated to laminar problems for water and air, we cannot find any real time results when using the test geometries. Thus, we have not found a correct view of the set of problems which are real time solvable.

Concluding, we have made a mobile CFD solver that can give real time results for a very specific problem. The CFD solver works quite optimal for the SIMPLE algorithm. However, we did not succeed in generalizing the result to water and air systems. The restrictions guaranteeing real time solutions remain unkown.

5.2. Recommendations

In further research, the restrictions for obtaining real time results can be explored. Instead of solving the equations for physical parameters (Equations 2.6), a non-dimensional form should be considered. The set of problems defined by the Ra and Pr numbers could be investigated for obtaining real time solutions. This will give a more general view on real time solvability which could result in a better view on arbitrary fluids being able to give real time results.

A

Calculation of the ordered sets

Let A and B be ordered sets, sorted from smallest to biggest element, and $n \in \mathbb{N}$. In the following equations the operations below are defined as follows:

1. $C = A \cup B = (c_1, c_2, ..., c_m)$ where $c_j \in A$ or $c_j \in B$ and $c_{j-1} < c_j < c_{j+1}$;
2. $C = A \cap B = (c_1, c_2, ..., c_m)$ where $c_j \in A$ and $c_j \in B$ and $c_{j-1} < c_j < c_{j+1}$;
3. $C = A \setminus B = (c_1, c_2, ..., c_m)$ where $c_j \in A$ and $c_j \notin B$ and $c_{j-1} < c_j < c_{j+1}$;
4. $A \circ n = (a_1 \circ n, a_2 \circ n, ..., a_m \circ n)$, where $\circ \in \{+, -\}$;
5. Assume $|A| = |B|$, then $A \circ B = (a_1 \circ b_1, a_2 \circ b_2, ..., a_m \circ b_m)$, where $\circ \in \{+, -\}$;
6. Assume $B \subseteq A$, then $C = \text{Loc}(A, B) = (c_1, c_2, ..., c_m)$ where $a_{c_j} = b_j$ and $c_{j-1} < c_j < c_{j+1}$;
7. $\left\lfloor \frac{A}{n} \right\rfloor = (\left\lfloor \frac{a_1}{n} \right\rfloor, \left\lfloor \frac{a_2}{n} \right\rfloor, ..., \left\lfloor \frac{a_m}{n} \right\rfloor)$.

The standard grid sets

The sets for the grid points

Consider the regular grid as in Figure A.1 (copied from Section 3.3). In general, the standar grid sets of interest can be calculated through

$$I = \{\, i \mid i \text{ is an internal boundary point}\}, \tag{A.1a}$$
$$G = \{1, 2, ..., n_x \cdot n_z\} \setminus I, \tag{A.1b}$$
$$G_N = (G \cap (I + n_x)) \cup (\{1, 2, ..., n_x\} \setminus I), \tag{A.1c}$$
$$G_E = (G \cap (I - 1)) \cup (\{n_x, 2 \cdot n_x, ..., n_x \cdot n_z\} \setminus I), \tag{A.1d}$$
$$G_S = (G \cap (I - n_x)) \cup (\{n_x \cdot (n_z - 1) + 1, n_x \cdot (n_z - 1) + 2, ..., n_x \cdot n_z\} \setminus I), \tag{A.1e}$$
$$G_W = (G \cap (I + 1)) \cup (\{1, n_x + 1, ..., n_x \cdot (n_z - 1) + 1\} \setminus I). \tag{A.1f}$$

The resulting locations are the sets

$$lG = \{1, 2, ..., |G|\},$$

$$\begin{aligned}
lG_N &= \text{Loc}(G, G_N), & lG_N^c &= lG \setminus lG_N, & lG_N^{c+} &= \text{Loc}(G, G_N^c - n_x),\\
lG_E &= \text{Loc}(G, G_E), & lG_E^c &= lG \setminus lG_E, & lG_E^{c+} &= lG_E^c + 1,\\
lG_S &= \text{Loc}(G, G_S), & lG_S^c &= lG \setminus lG_S, & lG_S^{c+} &= \text{Loc}(G, G_S^c + n_x),\\
lG_W &= \text{Loc}(G, G_W), & lG_W^c &= lG \setminus lG_W, & lG_W^{c+} &= lG_W^c - 1.
\end{aligned}$$

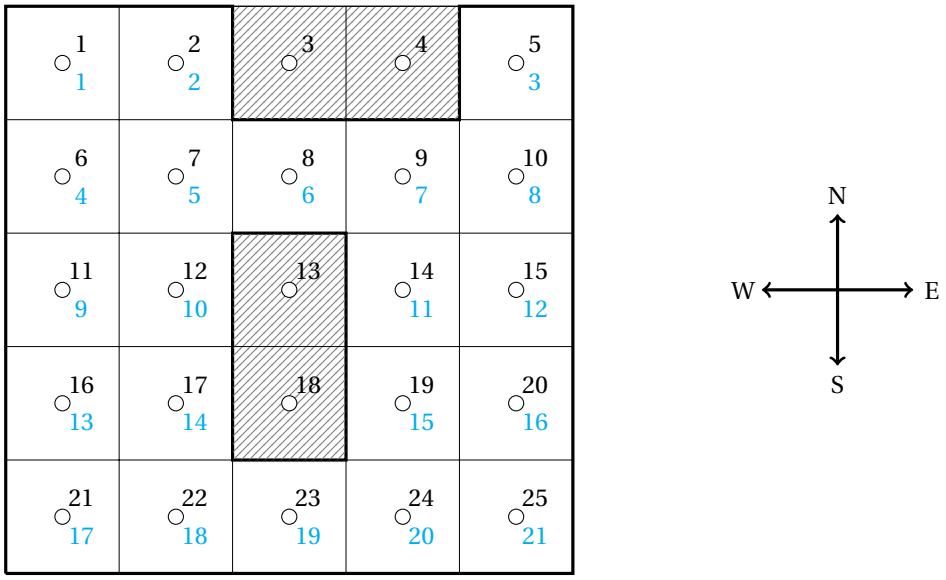


Figure A.1: A 5x5 grid with 4 striped interior boundary cells and 21 interior grid cells. The north, east, south and west directions are also denoted.

The sets for the horizontal boundary faces

Consider the horizontal faces as in A.2a. The sets of interest, the horizontal faces, can be calculated through

$$F_n = G_N, \tag{A.2}$$
$$F_s = G_S + nx, \tag{A.3}$$
$$F_{hb} = F_n \cup F_s. \tag{A.4}$$

The resulting locations are the sets

$$lF_n = \text{LOC}(F_{hb}, F_n), \qquad lF_s = \text{LOC}(F_{hb}, F_s).$$

The sets for the vertical boundary faces

Consider the vertical faces as in A.2b. The sets of interest, the vertical faces, can be calculated through

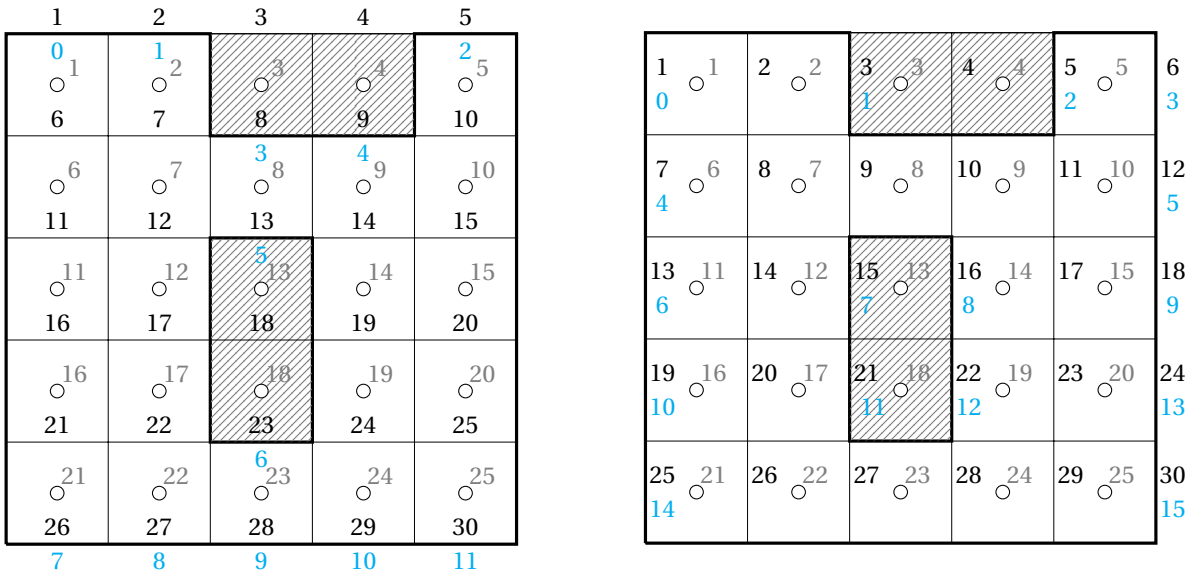
$$F_e = G_E + \left\lfloor \frac{G_E - 1}{n_x} \right\rfloor + 1, \tag{A.5}$$
$$F_w = G_W + \left\lfloor \frac{G_W - 1}{n_x} \right\rfloor, \tag{A.6}$$
$$F_{vb} = F_e \cup F_w. \tag{A.7}$$

The resulting locations are the sets

$$lF_e = \text{LOC}(F_{vb}, F_e), \qquad lF_w = \text{LOC}(F_{vb}, F_w).$$

The u-grid sets

For the u-grid, only the grid sets have to be known (and not the faces), because the no slip boundary conditions let every image to a face direct to a zero-velocity value on this face. In general, the sets of interest can be calculated through



(a) The horizontal faces

(b) The vertical faces

Figure A.2: A 5x5 grid with 4 striped interior boundary cells and 21 interior grid cells. The horizontal faces are numbered left and the vertical faces are numbered right.

$$G_u = \left(\left(G + \left\lfloor \frac{G-1}{n_x} \right\rfloor\right) \cup \left(G + \left\lfloor \frac{G-1}{n_x} \right\rfloor + 1\right)\right) \setminus F_{vb}, \tag{A.8}$$

$$G_{uN} = G_u \cap \left(\left(F_n + \left\lfloor \frac{F_n-1}{n_x} \right\rfloor + 1\right) \cup \left(F_n + \left\lfloor \frac{F_n-1}{n_x} \right\rfloor\right)\right), \tag{A.9}$$

$$G_{uE} = G_u \cap (F_e - 1), \tag{A.10}$$

$$G_{uS} = G_u \cap \left(\left(\left(F_s + \left\lfloor \frac{F_s-1}{n_x} \right\rfloor + 1\right) \cup \left(F_s + \left\lfloor \frac{F_s-1}{n_x} \right\rfloor\right)\right) - (n_x+1)\right), \tag{A.11}$$

$$G_{uW} = G_u \cap (F_w + 1). \tag{A.12}$$

The resulting locations are the sets

$$lG_u = \{1, 2, \ldots, |G_u|\},$$

$$lG_{uN} = \textsc{Loc}(G_u, G_{uN}), \quad lG^c_{uN} = lG_u \setminus lG_{uN}, \quad lG^{c+}_{uN} = \textsc{Loc}(G_u, G^c_{uN} - (n_x+1)),$$

$$lG_{uE} = \textsc{Loc}(G_u, G_{uE}), \quad lG^c_{uE} = lG_u \setminus lG_{uE}, \quad lG^{c+}_{uE} = lG^c_{uE} + 1,$$

$$lG_{uS} = \textsc{Loc}(G_u, G_{uS}), \quad lG^c_{uS} = lG_u \setminus lG_{uS}, \quad lG^{c+}_{uS} = \textsc{Loc}(G_u, G^c_{uS} + (n_x+1)),$$

$$lG_{uW} = \textsc{Loc}(G_u, G_{uW}), \quad lG^c_{uW} = lG_u \setminus lG_{uW}, \quad lG^{c+}_{uW} = lG^c_{uW} - 1.$$

The v-grid sets

As for the u-grid, only the v-grid sets have to be known. The sets of interest can be calculated through

$$G_v = (G \cup (G + n_x)) \setminus F_{hb}, \tag{A.13}$$

$$G_{vN} = G_v \cap (F_n + n_x), \tag{A.14}$$

$$G_{vE} = G_v \cap \left(\left(F_e - \left\lfloor \frac{F_e-1}{n_x+1} \right\rfloor - 1\right) \cup \left(F_e - \left\lfloor \frac{F_e-1}{n_x+1} \right\rfloor - 1 + n_x\right)\right), \tag{A.15}$$

$$G_{vS} = G_v \cap (F_s - n_x), \tag{A.16}$$

$$G_{vW} = G_v \cap \left(\left(F_w - \left\lfloor \frac{F_w-1}{n_x+1} \right\rfloor\right) \cup \left(F_w - \left\lfloor \frac{F_w-1}{n_x+1} \right\rfloor + n_x\right)\right). \tag{A.17}$$

(A.18)

The resulting locations are the sets

$$lG_v = \{1,2,...,|G_v|\},$$

$$lG_{vN} = \text{LOC}(G_v, G_{vN}), \quad lG^c_{vN} = lG_v \setminus lG_{vN}, \quad lG^{c+}_{vN} = \text{LOC}(G_v, G^c_{vN} - n_x),$$
$$lG_{vE} = \text{LOC}(G_v, G_{vE}), \quad lG^c_{vE} = lG_v \setminus lG_{vE}, \quad lG^{c+}_{vE} = lG^c_{vE} + 1,$$
$$lG_{vS} = \text{LOC}(G_v, G_{vS}), \quad lG^c_{vS} = lG_v \setminus lG_{vS}, \quad lG^{c+}_{vS} = \text{LOC}(G_v, G^c_{vS} + n_x),$$
$$lG_{vW} = \text{LOC}(G_v, G_{vW}), \quad lG^c_{vW} = lG_v \setminus lG_{vW}, \quad lG^{c+}_{vW} = lG^c_{vW} - 1,$$
$$lG_{vW1} = \text{LOC}(G_v, G_{vW1}), \quad lG^c_{vW2} = \text{LOC}(G_v, G_{vW2}), \quad lG_{vW3} = \text{LOC}(G_v, G_{vW3}).$$

Images between the u-grid and the v-grid

Images from the u-grid to the v-grid

In general, the sets representing the images from the u-grid to the v-grid at the north (denoted by an a and a b) and the south (denoted by an a and a b as well) can be calculated through

$$G_{uvna} = G_u \setminus G_{uN} - \left\lfloor \frac{G_u \setminus G_{uN} - 1}{n_x + 1} \right\rfloor - 1, \tag{A.19}$$

$$G_{uvnb} = G_u \setminus G_{uN} - \left\lfloor \frac{G_u \setminus G_{uN} - 1}{n_x + 1} \right\rfloor, \tag{A.20}$$

$$G_{uvsa} = G_u \setminus G_{uS} - \left\lfloor \frac{G_u \setminus G_{uS} - 1}{n_x + 1} \right\rfloor - 1 + n_x, \tag{A.21}$$

$$G_{uvsb} = G_u \setminus G_{uS} - \left\lfloor \frac{G_u \setminus G_{uS} - 1}{n_x + 1} \right\rfloor + n_x. \tag{A.22}$$

The resulting locations are the sets

$$lG_{uvna} = \text{LOC}(G_v, G_{uvna}), \quad lG_{uvnb} = \text{LOC}(G_v, G_{uvnb}),$$
$$lG_{uvsa} = \text{LOC}(G_v, G_{uvna}), \quad lG_{uvsb} = \text{LOC}(G_v, G_{uvsb}).$$

Images from the v-grid to the u-grid

The sets representing the images from the v-grid to the u-grid at the east (denoted by an a and a b) and the west (denoted by an a and a b as well) can be calculated through

$$G_{vuea} = G_v \setminus G_{vE} + \left\lfloor \frac{G_v \setminus G_{vE} - 1}{n_x} \right\rfloor + 1, \tag{A.23}$$

$$G_{vueb} = G_v \setminus G_{vE} + \left\lfloor \frac{G_v \setminus G_{vE} - 1}{n_x} \right\rfloor - n_x, \tag{A.24}$$

$$G_{vuwa} = G_v \setminus G_{vW} + \left\lfloor \frac{G_v \setminus G_{vW} - 1}{n_x} \right\rfloor, \tag{A.25}$$

$$G_{vuwb} = G_v \setminus G_{vW} + \left\lfloor \frac{G_v \setminus G_{vW} - 1}{n_x} \right\rfloor - (n_x + 1). \tag{A.26}$$

The resulting locations are the sets

$$lG_{vuea} = \text{LOC}(G_v, G_{vuea}), \quad lG_{vueb} = \text{LOC}(G_v, G_{vueb}),$$
$$lG_{vuwa} = \text{LOC}(G_v, G_{vuwa}), \quad lG_{vuwa} = \text{LOC}(G_v, G_{vuwa}).$$

Bibliography

[1] Relative performance solve. `https://github.com/optimatika/ojAlgo/wiki/Solve`. Accessed: 2017-06-05.

[2] Processing power compared. `http://pages.experts-exchange.com/processing-power-compared/`. Accessed: 2017-07-02.

[3] Stuart W Churchill and Humbert HS Chu. Correlating equations for laminar and turbulent free convection from a vertical plate. *International journal of heat and mass transfer*, 18(11):1323–1329, 1975.

[4] Donald D Gray and Aldo Giorgini. The validity of the boussinesq approximation for liquids and gases. *International Journal of Heat and Mass Transfer*, 19(5):545–551, 1976.

[5] K. Hanjalic, S. Kenjeres, M.J. Tummers, and H.J.J. Jonker. *Analysis and Modelling of Physical Transport Phenomena*. VSSD, 2007.

[6] Ifan Hughes and Thomas Hase. *Measurements and their uncertainties: a practical guide to modern error analysis*. Oxford University Press, 2010.

[7] Léon Peter Bernard Marie Janssen and Marinus Maria Cornelis Gerardus Warmoeskerken. *Transport phenomena data companion*. Hodder Arnold, 1987.

[8] Suhas Patankar. *Numerical heat transfer and fluid flow*. CRC press, 1980.

[9] Olga Shishkina, Richard JAM Stevens, Siegfried Grossmann, and Detlef Lohse. Boundary layer structure in turbulent thermal convection and its consequences for the required numerical resolution. *New Journal of Physics*, 12(7):075022, 2010.

[10] Cornelis Vuik, P Van Beek, F Vermolen, and J Van Kan. *Numerical Methods for Ordinary differential equations*. VSSD, 2007.

[11] DC Wan, BSV Patnaik, and GW Wei. A new benchmark quality solution for the buoyancy-driven cavity by discrete singular convolution. *Numerical Heat Transfer: Part B: Fundamentals*, 40(3):199–228, 2001.