NTNU
Norwegian University of
Science and Technology

Three-dimensional numerical modeling of water flow in a rock-blasted tunnel

Mari Vold

Civil and Environmental Engineering
Submission date: July 2017
Supervisor: Nils Reidar Bøe Olsen, IBM

Norwegian University of Science and Technology
Department of Civil and Environmental Engineering

Three-dimensional Numerical Modeling of Water Flow in a Rock-Blasted Tunnel

Mari Vold

July 2017

Abstract

Climate change leads to more intense precipitation in Norway. NVE requires all bypass tunnels have sufficient capacity for bypassing flood water. The hydro-power industry needs a method for establishing tunnel capacity, such as computing a friction factor. Friction factor for rock-blasted tunnels have yet to be studied thoroughly. This thesis describes a method for finding friction factors for Litjfosstunnelen using numerical models. Three turbulence models are tested and compared; k-Epsilon, k-Omega and k-Omega SST. For otherwise identical setups k-Epsilon converges better than the other two. k-Epsilon yields larger Darcy friction factors than k-Omega and k-Omega SST in general. Grids with more cells yield larger friction factors than a coarse mesh (with one exception). Grid independence is not reached.

Samandrag

Klimaendringar fører til meir intens nedbør i Noreg. NVE stiller krav om at flomavledningstunnellar skal ha tilstrekkeleg kapasitet for å ta unna flomvatnet. Vassdragsbransjen treng ein metode for å finne mål på tunnelkapasitet, eitt slikt mål er ein friksjonsfaktor. Friksjonsfaktorar for råsprengde tunnellar har enno ikkje blitt inngåande studert. Denne masteroppgåva brukar numeriske modellar for å finne friksjonsfaktorar i Litjfosstunnellen. Tre turbulensmodellar er testa og samanlikna; k-Epsilon, k-Omega og k-Omega SST. For elles like oppsett konvergerer k-Epsilon betre enn dei andre to. k-Epsilon gir også generelt høgare Darcy-friksjonsfaktorar. Finare grid gir høgare friksjonsfaktorar enn grove grid (med eitt unntak). Griduavhengigheit er ikkje nådd.

This master's thesis was written at the Department of Civil and Environmental Engineering at the Norwegian University of Science and Technology (NTNU) under supervision of Nils Reidar Olsen. The thesis is part of a research project which aims to find friction factors in rock-blasted tunnels. The research project is a collaboration between the department, Trønderenergi, BKK, the Research Council of Norway and the Norwegian Water Resources and Energy Directorate (NVE). Selected tunnels are scanned using laser technology, and the geometry data can then be used to build physical models or create meshes for numerical computations. The physical models are tested in a hydraulic laboratory, where friction loss, velocities and turbulence are measured. The main objective for this thesis was to find friction factors for Litjfossen using three-dimensional numerical models in the open source CFD program OpenFOAM.

The work started in February 2017, apart from a visit to the tunnel at Litjfossen the day it was scanned (august 30th, 2016). The work has consisted of study of relevant literature and trial and error using the CFD software. Considerable help has been provided from Radek Maca at CFD Support, Ltd. An amount of time was invested in studying C++ programming, in order to better understand the workings of OpenFOAM. A better choice would be to study turbulence modeling or numerical mathematics.

I send my thanks to all who have contributed with their knowledge, patience and enthusiasm.

Trondheim
July 17, 2017
Mari Vold

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background

Climate change is likely to lead to more extreme rainfalls and flood events in Norway. The Norwegian Water Resources and Energy Directorate represents the authorities in preventing flood events from becoming a danger to the public. NVE set requirements for a bypass tunnel's capacity and have the authority to demand expansion of a tunnel if the capacity is too small [1]. A tunnel expansion is expensive, which gives the industry an incentive to find new ways to prove that their tunnel's capacity already fulfills NVE's requirements.

Tunnel capacity depends on, among other things, the friction loss, which can be estimated by a friction factor. A popular and frequently used friction factor is the Manning Strickler value, M, known from the Manning Strickler equation:

$$Q = MAR_h^{2/3}I^{1/2} \tag{1}$$

This equation has been used for lack of a better practical alternative. Manning Strickler friction factors are empirical and have complicated dimensions, thus they must be scaled from model to prototype. CFD and lab studies produce result and measurements that can be used to compute a dimensionless friction factor, the Darcy friction factor, in addition to Manning Strickler friction factors. The Darcy friction factor, denoted f_D, is known from the Darcy Weißbach equation:

$$\frac{\Delta p}{L} = f_D \frac{\rho}{2}\frac{U^2}{D} \tag{2}$$

Numerical modeling can be much more cost effective than physical model tests, but NVE does not accept these results as proof for adequate tunnel capacity. The results must be validated. A comparison between lab model measurements and CFD results is therefore very interesting, and tunnel roughness is a field where research and data is scarce. This gives the research community motivation to initiate studies. The combined interests from hydropower developers, government and the department made this research project possible and laid the foundation for this thesis.

A physical model of Litjfosstunnelen is being built at the time of writing. It was originally meant to produce measurement results already around Easter, 2017, which would have enabled the author of this thesis to compare CFD and physical model results. Unfortunately, the lab model was delayed by several months. The assignment was therefore tweaked to comparing turbulence models instead. The turbulence models used are the most popular and well-documented two-equation models available, namely the k-Epsilon, k-Omega and k-Omega SST models. To ensure that the results are still comparable to lab model measurement in the future, the setup has been adapted to the lab model wherever possible. Details on the setup are further described in the following.

1.2 Computational Fluid Dynamics (CFD) and turbulence modeling

Only the simplest of flows can be described and solved using analytical equations, but modern day engineering includes more complex geometries and a great variety of flow conditions. These flow problems are solved numerically using the Navier Stokes equations. These equations were derived in the early 19th century, but are only recently put into extensive use. This is because of the exponential growth in computational power over the last decades.

The union of mathematics, hydraulics and computer science is called computational fluid dynamics, CFD. In other words, it is an interdisciplinary science which requires vast knowledge in each field to completely comprehend the joint methods, especially for complex flow problems with high turbulence. Turbulence modelling is difficult, because of the turbulence closure problem. Turbulence models require assumptions to close the system of governing equations (Navier Stokes and Reynolds stress equations). The complexity of turbulence has inspired statements such as Werner Heisenberg's

"*When I meet God, I am going to ask him two questions: Why relativity? And why turbulence? I really believe he will have an answer for the first.*"

With these humbling words in mind, it does not stop us from trying. There are numerous turbulence models available, the choice of which is a challenge for an engineer. Models are developed for specific flow conditions and use a series of case-specific constants that may not apply to every flow problem. The user should always be conscious of this and other sources of error and uncertainty in CFD. Models, methods and algorithms must be chosen carefully so that they suit the problem at hand, and CFD results should always be controlled, verified and validated.

The open-source CFD software used in this thesis is called OpenFOAM. It is a popular tool and affords complete control with every entry in the program. With great freedom comes great responsibility. Only more experienced CFD analysts should try to change program code, but programming skills are useful also for the novice, because it makes it easier to implement one's case and find their own errors in input. The source code also contains information about the program that is only briefly documented elsewhere, if at all. OpenFOAM has no graphical user interface, but the dictionaries/scripts are intuitively constructed and solutions can be visualized in post-processing tools. One such tool is ParaView which is included in OpenFOAM download.

1.3 Litjfossen

The tunnel in question for this thesis is called Litjfosstunnelen. It is clearly affected by its environment. The tunnel is blasted in a mountain with weakness zones in a semi-regular pattern at a skew angle relative to the tunnel axis. This contributes to irregularity and roughness which is clearly visible in Figure 1.

As the figures imply, the whole geometry is skewed because of these weakness zones, which is suspected to lead to secondary flows in the tunnel.

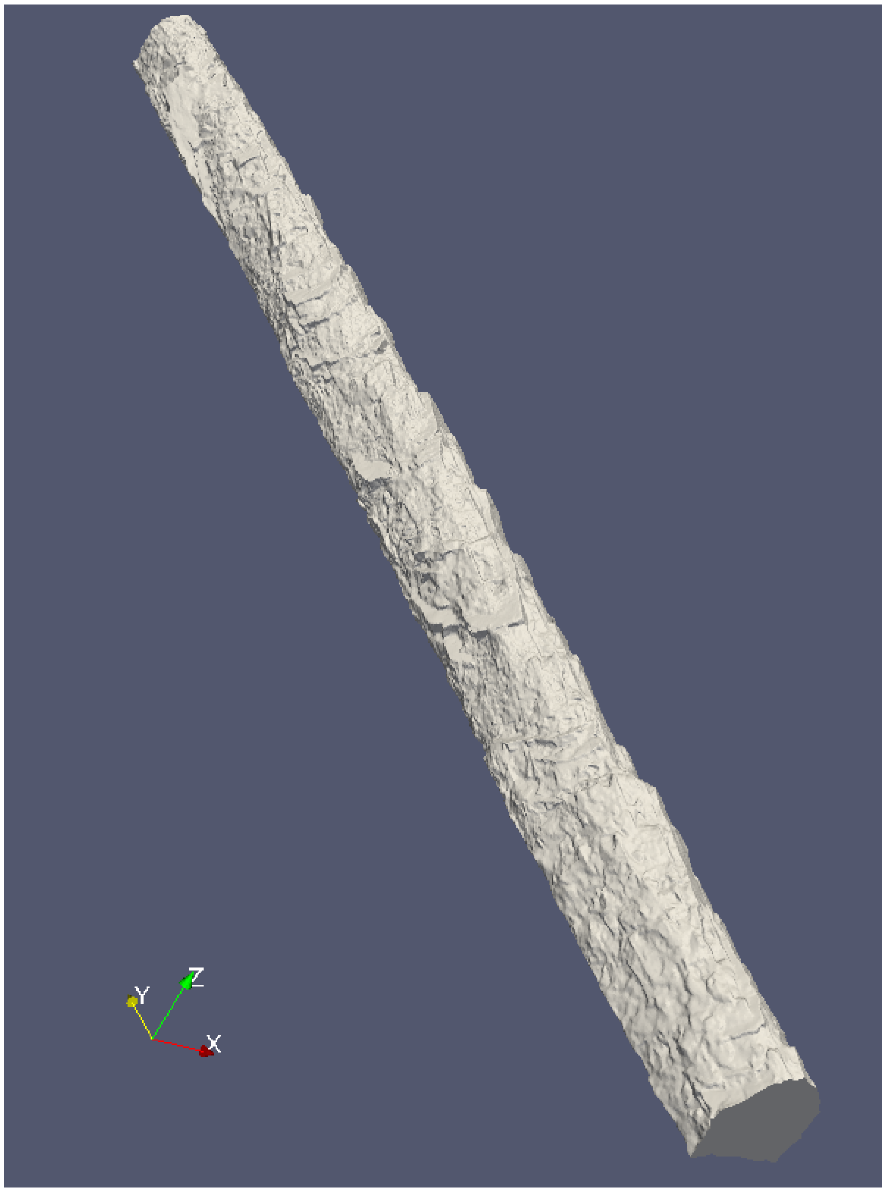


Figure 1: Litjfossen. Notice the ridge-back ceiling

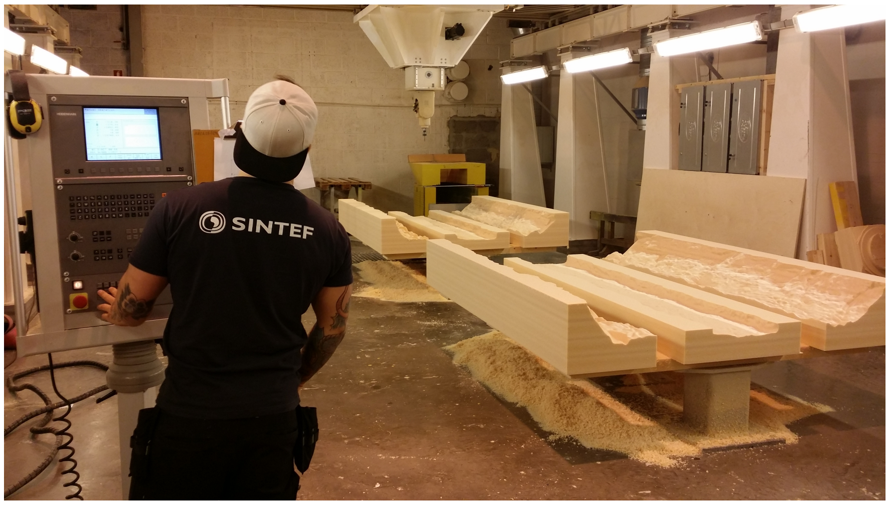


Figure 2: Lab model construction. [11][12]

1.4 Lab model as foundation for CFD setup

The lab model geometry is made from laser scanned data scaled by a factor of 1/15. The model is 8 meters long and roughly 40 centimeters in diameter. The lab model will be tested for different discharges, whereas this thesis is focused on one likely discharge ($0,1m^3/s$) within the range $[0-0,2m^3/s]$ of what the lab can supply. The temperature in the lab is approximately 20°C. Although the water temperature will initially be a bit lower, it is expected to reach room temperature after circulation through the model, therefore 20°C is used to set water viscosity, $\nu = 1,0034x10^{-6}$.

To find friction factors, head loss will be measured in the lab. Pressure is measured at several locations, giving pressure loss data. Velocities will be measured using PIV technology (Particle Image Velocimetry) at a few specific locations. Other measurements are dependent on available equipment in the future and other factors that are unknown to the author, and are therefore not featured in this thesis.

The lab model is installed so that the flow field at the inlet will be as uniform as possible. The water is recirculated through a pipe (this ensures steady-state conditions), and there is a sharp bend before the inlet. Flows in sharp bends are typically pushed to the outer wall, which works against the aim of having uniform flow at the inlet. Steering wings are placed inside the pipe bend to prevent this effect. This is meant to induce very turbulent flow and thereby an even distribution of velocity immediately after the bend. The flow then enters a so-called honeycomb filter. Each cell in this filter is too small for turbulent eddies of significant size to pass, therefore the flow will be less turbulent when it leaves the filter, enters the following expansion tract and continues into the tunnel. This justifies a uniform fixed velocity boundary condition for the inlet. Since water is viscous, the BC at the tunnel wall is set to a uniform value of 0 in all directions ($\mathbf{u}=\mathbf{0}$). This is also called the no-slip condition. The outlet BC for water is set to zero gradient.

Pressure measurements will be done by drilling 20-25 small holes in the tunnel walls (including ceiling and floor) at a cross-section. The holes will be interconnected via tubes (one tube to rule them all). Consequently there is only one combined pressure measurement for a cross-section, and it serves as a main value for the pressure at this location. Corresponding mean values for what the author hopes are the same cross-sections are computed from the numerical modeling data and used when finding the Darcy friction factors.

Absolute pressures are of no interest and will not be measured in the lab. For this type of flow the head loss will remain constant over time, or close to it. In other words, head loss deviations are negligible. Pressures are also computed as gauge pressures in the numerical model, which makes it unnatural to specify absolute pressure values at boundaries. The easiest way to choose boundary conditions (abbreviated BCs for a large part of the following text) for such a case is to use a zero gradient BC at one inlet/outlet boundary and a uniform value of zero at the other. However, since the pressure distribution is unlikely to be uniform in the lab model, a boundary condition called fixedMean is used instead, which means the pressure distribution can be non-uniform, but the mean value is set to a chosen value of, say, zero. This is practical because it makes it easy to find absolute pressures by adding a reference pressure value, should it turn out to be interesting after all. BC against the tunnel wall is zero gradient.

As mentioned, velocity measurements in the lab will be done using laser

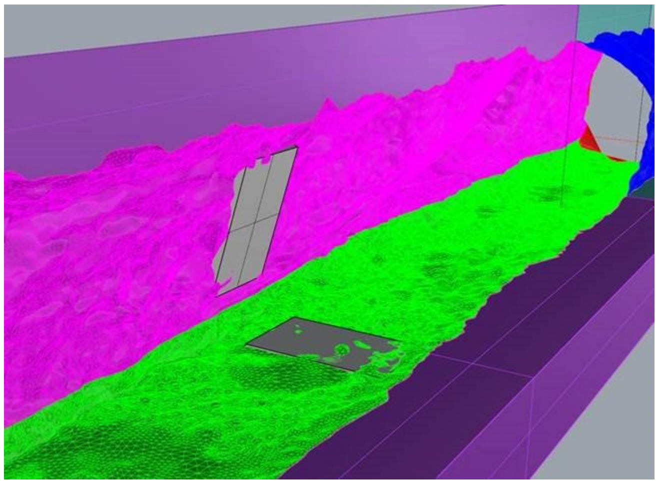


Figure 3: Concept figure for lab model windows [11]

technology called Particle Image Velocimetry (PIV). This is a technique that shoots laser beams into a flow section of interest while simultaneously taking pictures with high frequency. The lab model is not transparent, so window pairs are built into the tunnel geometry. One of the pair is located at the side/wall and the other at the bottom/floor. The one at the side is for the laser beams to shoot through, and the one at the bottom is for the camera to take pictures through. This window is smooth, while the laser beam window has the same shape as the wall which it replaces, with a few exceptions:

- There are smooth, horizontal lines cut into the window where the laser beam is meant to shoot through. This is to prevent scattering of the beams.
- The window's surface is visibly cut from a coarse triangle mesh, while the rest of the tunnel is cut using higher resolution mesh.

There are some limitations to this approach. Most important is the size of the windows, which is not large enough to map the entire section. As a result, near wall conditions fall out of scope. This does not mean that near wall conditions are not interesting in the CFD analysis, but they loose priority and tempt the author to postulate that k-Epsilon and k-Omega SST data will look very much alike, since the main difference between the two is indeed in how they model near wall conditions.

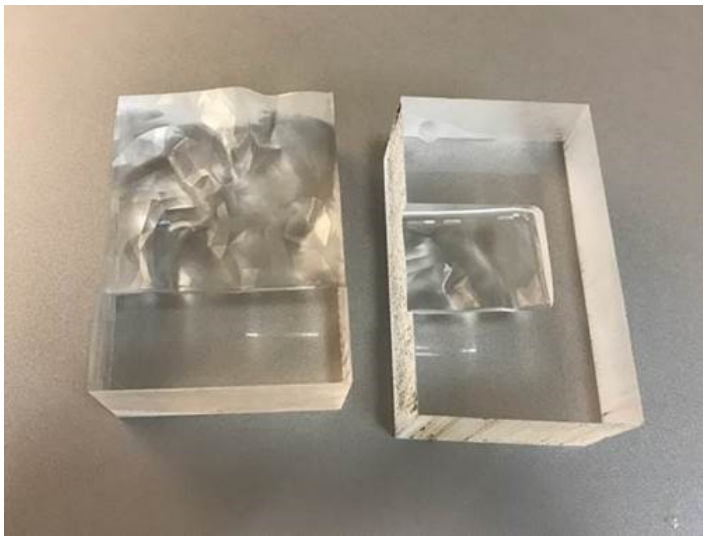


Figure 4: Lab model windows before installation [11]

Geometry Data			
Size	Symbol	Value	Source
Diameter	D	0,422m	Lab model
Hydraulic diameter	D_h	0,422m	$D_h = A_w/P_w$
Cross-section area	A	0,159m^2	$A = r^2(\pi/2 + 2)$
Length	L	8m	Lab model

Table 1: Geometry data for lab and numerical model

Some specifics from the lab geometry are show in Table 1.

2 Theory

2.1 Flow properties

To ensure that computation time is not wasted on calculating properties of immaterial value, a number of assumptions are made to help reduce the complexity of a flow problem. These assumptions are practical and common, although not entirely accurate:

- The fluid is NEWTONIAN, meaning that the shear stress is directly proportional to the rate of strain
- The fluid is ISOTROPIC
- The system is in THERMODYNAMIC EQUILIBRIUM (used in derivation of Navier Stokes)

Other assumptions are more case specific. In some cases where the geometry can be repsresented in two dimensions, for instance, the flow problem can be solved in 2D. For this case the geometry requires three-dimensional representation. Consequently, so does the flow. The following assumptions are reasonable for Litjfosstunnelen [3] [2]:

- The flow is ONE-PHASE: There is only water and no other fluid in this flow
- The flow is STEADY-STATE: There is no change in time, only space. This seems absurd for a turbulent flow, but using Reynolds averaging, where velocity is computed as the sum of a mean and fluctuating value, one can say that the flow is steady if the mean velocity is constant.
- The flow is (FULLY) TURBULENT: (There is no transition zone.) This can be established through a quick study of Reynolds numbers. With the geometry, discharge and temperatures from section 1, the Reynolds number is larger than 260000.
- The fluid is INCOMPRESSIBLE: Water density, ρ, is constant over time and space. Flow can be solved without consideration of the energy equation.

This calls for caution. All CFD results must be regarded as incorrect to a degree, and sources of error must be accounted for. As demonstrated, they are numerous. Sources of error and uncertainty are discussed in section 5.

2.2 Governing equations

The most successful way of discretising a flow problem is by using the control volume method, also called the finite volume method or Eulerian approach. As the name implies, the problem is divided into control volumes and conservation laws of physics are applied on this volume. The governing equations are

- The continuity equation (conservation of mass)
- The momentum equation (Newton's second law)
- Reynolds transport theorem

Reynolds transport theorem is applied to the governing equations to transform them into Eulerian form. [3]

Conservation of mass
Conservation of mass means that the amount of mass stays constant over time. Therefore, if there is mass flow through a volume, then the rate of accumulation within that volume equals the net efflux. In vector notation:

$$\frac{\partial \rho}{\partial t} + \mathrm{div}(\rho \mathbf{u}) = 0 \tag{3}$$

This is also called *the continuity equation* for a compressible fluid in an unsteady flow. For an incompressible fluid the continuity equation reduces to

$$\mathrm{div}(\mathbf{u}) = 0 \tag{4}$$

or, in differential form

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{5}$$

Reynolds transport theorem
The rate of change of a general property ϕ per unit volume can be expressed in a similar way as the rate of change of mass per unit volume (equation 3)

$$\frac{\partial \rho\phi}{\partial t} + \mathrm{div}(\rho\phi\mathbf{u}) = \rho\frac{D\phi}{Dt} \tag{6}$$

In words: The rate of change of property ϕ equals the sum of the rate of increase of ϕ in the control volume and the net efflux.

The momentum equation
Once Reynolds transport theorem is established, it can be applied to Newton's second law to calculate the rate of change of momentum using the control volume approach.

$$\Sigma\mathbf{F} = \frac{d(\mathbf{Mom}_{sys})}{dt} \tag{7}$$

Using

$$\mathbf{Mom}_{sys} = m\mathbf{v} = \rho V\mathbf{v} \tag{8}$$

The sum of forces on the volume is the sum of pressure, gravity (and other body forces), shear and normal stress forces. [3] Applying Reynolds transport theorem to Newton's second law on a three-dimensional infinitesimal control volume yields the differential Navier-Stokes equation in three-dimensions for laminar flow [4]:

$$\frac{\partial U_i}{\partial t} + U_j\frac{\partial U_i}{\partial x_j} = \frac{1}{\rho}\frac{\partial}{\partial x_j}\Bigg(- P\delta_{ij} - \rho\nu\bigg(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\bigg)\Bigg) \tag{9}$$

This case is turbulent, which means there are velocity fluctuations and extra shear stresses to attend to, called Reynolds stresses. Velocity is modelled as the sum of a mean value and a fluctuating value, **u=U+u'**. A slightly different version of Navier-Stokes emerges

$$\frac{\partial U_i}{\partial t} + U_j\frac{\partial U_i}{\partial x_j} = \frac{1}{\rho}\frac{\partial}{\partial x_j}\big(- P\delta_{ij} - \rho\overline{u_i u_j}\big) \tag{10}$$

using the Boussinesq approximation for the Reynolds stress (turbulent stress) term on the right side:

$$-\rho\overline{u_i u_j} = \rho\nu_T\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) - \frac{2}{3}\rho k\delta_{ij} \tag{11}$$

yielding, after insertion and rearrangement, the **Reynolds-averaged Navier-Stokes equation for turbulent flow** [4]

$$\frac{\partial U_i}{\partial t} + U_j\frac{\partial U_i}{\partial x_j} = \frac{1}{\rho}\frac{\partial}{\partial x_j}\left[-\big(P + \frac{2}{3}k\big)\delta_{ij} - \rho\nu_T\frac{\partial U_i}{\partial x_j} + \rho\nu_T\frac{\partial U_j}{\partial x_i}\right] \tag{12}$$

The transient term on the left is of course zero for a steady-state case, leaving the convective term alone on the left side. P represents pressure, k is kinetic energy, and the two remaining terms represent diffusion (turbulent stress) and viscous stresses. δ_{ij} is the Kronecker delta, which is necessary for the formula to give the right result for normal Reynolds stresses.

2.3 Numerical solution

The governing equations combined leave a closed system with an equal number of equations and unknowns. What remains now is a procedure for solving these equations numerically. OpenFOAM's solver for incompressible steady-state turbulent flow is called simpleFoam, and solves the flow using the SIMPLE algorithm.

2.3.1 The SIMPLE Method

SIMPLE, Semi-Implicit Method for Pressure-Linked Equations, starts with a (preferably qualified) guess of initial and boundary conditions for the pressure field. The guessed value is denoted with an asterisk, p*. p* is applied to the discretised momentum equations in order to calculate velocities, u_i*. These velocities are denoted and treated as guessed values.

The continuity equation is used to express a correction to the pressure, p'. Guessed and correction values are summed to obtain the actual pressure. Similarly, velocity corrections are added to u$_i$*:

$$p = p * + p' \tag{13}$$

$$u_i = u_i * + u_i' \tag{14}$$

After pressure and velocities are corrected, the other discretised transport equations are solved. This concludes one iteration. The process is repeated using the corrected pressure and velocities as initial guesses for the next iteration. This continues until some convergence criterion has been fulfilled. There may be different criteria for different flow properties. Convergence criteria for this thesis are discussed in section 2.3.3. A flowchart for the SIMPLE method from Versteeg & Malalasekera [2] is shown below.

There are several versions of this algorithm, such as the SIMPLE revised or SIMPLER method, which calculates the pressure field directly through a discretised equations for pressure. [2] Nevertheless, only the original SIMPLE method was used for this thesis as it is the foundation for the chosen OpenFOAM solver simpleFOAM.

2.3.2 Under-relaxation

The SIMPLE method is prone to instabilities. One way to prevent this is by introducing under-relaxation coefficients, α_ϕ, for the pressure and velocity corrections. An under-relaxation coefficient is a factor between 0 and 1 that is used to dampen corrected values.

$$p^{new} = p * + \alpha_p p' \tag{15}$$

$$u^{new} = \alpha_u u + (1 - \alpha_u) u^{(n-1)} \tag{16}$$

Now, even if the solution oscillates the amplitude is reduced, which may stop the solution from oscillating or diverging. However, this will also lead to slower convergence. The optimum relaxation coefficient in dependent on mesh, flow problem and iteration method and is hard to predict. [2]

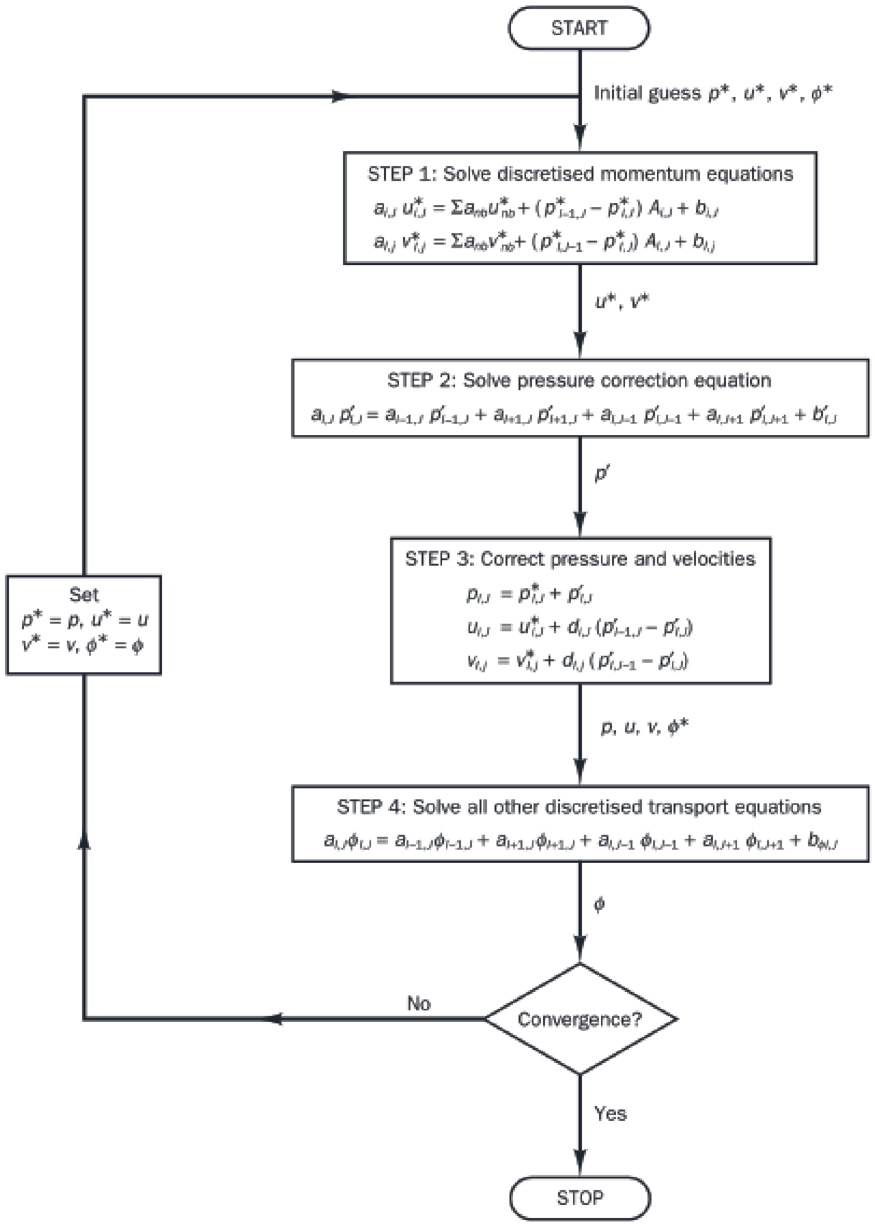

START
Initial guess p^*, u^*, v^*, ϕ^*
STEP 1: Solve discretised momentum equations
$a_{i,J}u^*_{i,J} = \Sigma a_{nb}u^*_{nb} + (p^*_{I-1,J} - p^*_{I,J})A_{i,J} + b_{i,J}$
$a_{I,j}v^*_{I,j} = \Sigma a_{nb}v^*_{nb} + (p^*_{I,J-1} - p^*_{I,J})A_{I,j} + b_{I,j}$
u^*, v^*
STEP 2: Solve pressure correction equation
$a_{I,J}p'_{I,J} = a_{I-1,J}p'_{I-1,J} + a_{I+1,J}p'_{I+1,J} + a_{I,J-1}p'_{I,J-1} + a_{I,J+1}p'_{I,J+1} + b'_{I,J}$
p'
STEP 3: Correct pressure and velocities
$p_{I,J} = p^*_{I,J} + p'_{I,J}$
$u_{i,J} = u^*_{i,J} + d_{i,J}(p'_{I-1,J} - p'_{I,J})$
$v_{I,j} = v^*_{I,j} + d_{I,j}(p'_{I,J-1} - p'_{I,J})$
p, u, v, ϕ^*
STEP 4: Solve all other discretised transport equations
$a_{I,J}\phi_{I,J} = a_{I-1,J}\phi_{I-1,J} + a_{I+1,J}\phi_{I+1,J} + a_{I,J-1}\phi_{I,J-1} + a_{I,J+1}\phi_{I,J+1} + b_{\phi I,J}$
ϕ
Convergence?
No
Yes
STOP
Set
$p^* = p$, $u^* = u$
$v^* = v$, $\phi^* = \phi$

Figure 5: The SIMPLE algorithm flowchart

2.3.3 Numerical schemes

The most important fundamental properties of discretisations schemes are:

- Conservativeness: The amount of a property entering a cell face from one side is the same as the amount leaving the same face on the other side.
- Boundedness: Property value at a point is within the range spanned by its boundaries

$$boundaryValue_{MIN} < valueAtNode < boundaryValue_{MAX} \tag{17}$$

- Transportiveness: The influence on a cell's value of a property ϕ from its neighbouring cells depends on the relation between convection and diffusion in the flow. This relation is called the Peclet number, Pe. Influence from a neighbouring cell should be biased according to the Peclet number.

$$Pe = \frac{F}{D} = \frac{\rho u}{\Gamma/\delta x} \tag{18}$$

  For pure diffusion, Pe$->$0, for pure convection, Pe$- > \infty$

To use an everyday comparison: Shopping for numerical schemes is just like any other shopping: It is difficult to get it all in one product. Desirable characteristics can be mutually exclusive, such as second-order accuracy and boundedness. Also, quality costs. Sometimes the best is simply too expensive, in this context meaning that the time of computation cannot be afforded. That being said, a scheme can perform well enough as long as it suits the problem in question, even if it is only first-order accurate or has poor transportiveness. For instance, the central differencing scheme is adequate for a pure diffusion case, but has poor transportiveness when Pe increases. [2]

The schemes in this case a chosen to try to reach second-order accuracy. Schemes are specified in fvSchemes, as described in section 3.3.5.

2.4 Turbulence modeling

Turbulence is generated by shear in the flow, which creates eddies. This typically happens near a solid boundary, like a wall. Eddies transport mass, momentum and energy across the flow, resulting in a rapid exchange of these properties and an even distribution of them. Compared to laminar flow equations there are additional shear stresses in turbulence equations (turbulent shear stress or Reynolds stresses), due to the the mixing of high and low velocity flow. Similar to laminar flow, velocities decrease from the main flow to the wall, but the main flow makes up more of the flow cross-section and the gradient is steeper towards the wall for turbulent flow. Turbulence production is high for these steep gradients. It is common to separate the flow into layers characterized by their dominating stresses. In the main flow or outer region, turbulence stresses dominate. Closer to the wall, viscous and turbulence stresses both affect the flow, while viscous stresses dominate and kill turbulence in the viscous sub-layer. Dimension analysis gives relations between flow velocity and distance from the wall that are shown below.

$$u^+ = y^+ \tag{19}$$

using definitions for dimensionless u^+ and y^+ from the law of the wall:

$$u^+ = \frac{U}{u_\tau} = f\left(\frac{\rho u_\tau y}{\mu}\right) = f(y^+) \tag{20}$$

where U is velocity in meters per second, u_τ is shear velocity and y is distance from the wall. This linear relation holds for the innermost part of the viscous sub-layer. Adjacent to this layer is a buffer region, where viscous and turbulence stresses both affect the flow. The buffer region is a transition before the so-called log-law layer where turbulent stresses take over:

$$u^+ = \frac{1}{\kappa} ln(Ey^+) \tag{21}$$

where κ is the Von Karman constant (=0.4) and E=9.8 for smooth walls. The relations above are illustrated in figure 6. Notice that, at the wall, velocities relative to the wall are zero for viscous flows. This is called the no-slip condition and is used in election of boundary conditions for velocity at the wall.

Turbulence eddies near the wall are generally anisotropic, because of the wall's restrictions on flow velocity direction. Farther from the wall, or where

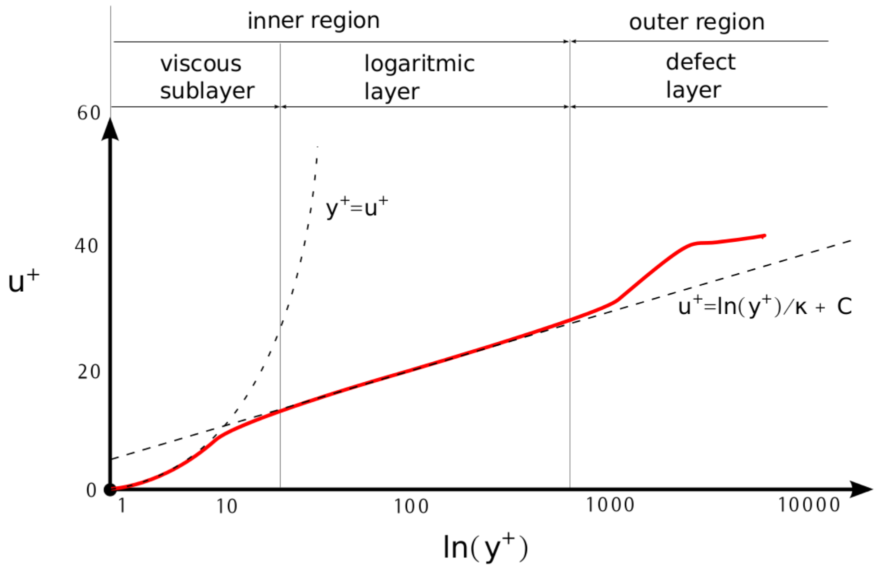


Figure 6: Layering near wall in turbulent flow [5]

there is less shear, turbulence is more isotropic. Many turbulence models are unable to catch anisotropic turbulence. This is a challenge in CFD. Nevertheless, mean flow is often of most interest, therefore these models are still widely used.

A problem in turbulence modelling is to get a closed set of equations, and turbulence models are often categorized according to the number of extra transport equations they solve to close the set. Reynolds stress models, solving as many as seven extra equations, are considered more physically correct than Reynolds averaged models, which solve two extra equations. Nevertheless, two-equation models are popular for their adequate performance paired with low computational costs. In general, turbulence models should be chosen that are developed for a similar type of flow to the problem at hand. Unfortunately, there is not one in particular for rock-blasted tunnels. The choice of turbulence models for this thesis was therefore based on popularity, documentation, existing OpenFOAM implementation and simplicity so as to keep the scope within reasonable bounds for a master's thesis. All models used are Reynolds averaged models. [2]

2.4.1 k-ϵ

The k-Epsilon model focuses on turbulent kinetic energy and how it is affected by turbulence mechanisms. Analogous to the derivation of Navier-Stokes for turbulent flow using the sum of a mean and a fluctuation velocity, turbulent kinetic energy is modelled as the sum of a mean (K) and turbulent (k) kinetic energy:

$$k(t) = K + k \tag{22}$$

The k-Epsilon model solves two partial differential equations, one for the turbulent kinetic energy, k, and one for the rate of dissipation of turbulent kinetic energy, ϵ. The Reynolds stresses are solved using Boussinesq (see derivation of Navier-Stokes equation in chapter 2.2 Governing equations). Model equations as shown in Versteeg & Malalasekera [2] are listed below .

Turbulent kinetic energy equation

$$\frac{\partial(\rho k)}{\partial t} + div(\rho k\mathbf{U}) = div\left[\frac{\mu_t}{\sigma_k} gradk\right] + 2\mu_t S_{ij}.S_{ij} - \rho\epsilon \tag{23}$$

Dissipation rate

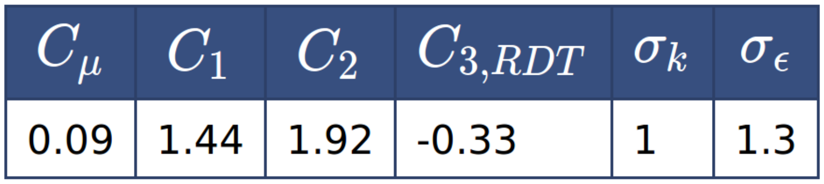
$$\frac{\partial(\rho\epsilon)}{\partial t} + div(\rho\epsilon\mathbf{U}) = div\left[\frac{\mu_t}{\sigma_\epsilon} grad\epsilon\right] + C_{1\epsilon}\frac{\epsilon}{k}2\mu_t S_{ij}.S_{ij} - C_{2\epsilon}\rho\frac{\epsilon}{k} \tag{24}$$

Eddy viscosity

$$\mu_t = \rho C_\mu \frac{k^2}{\epsilon} \tag{25}$$

divided by density:

$$\nu_t = C_\mu \frac{k^2}{\epsilon} \tag{26}$$



C_μ	C_1	C_2	$C_{3,RDT}$	σ_k	σ_ϵ
0.09	1.44	1.92	-0.33	1	1.3

Figure 7: Constants used for turbulent kinetic energy and dissipation rate equations

OpenFOAM implementation
Turbulent kinetic energy equation

$$\frac{D}{Dt}(\rho k) = \nabla(\rho D_k \nabla_k) + G_k - \frac{2}{3}\rho(\nabla \mathbf{u})k - \rho\epsilon + S_k \tag{27}$$

Dissipation rate

$$\frac{D}{Dt}(\rho\epsilon) = \nabla(\rho D_\epsilon \nabla_\epsilon) + \frac{C_1 G_k \epsilon}{k} - (\frac{2}{3}C_1 + C_{3,RDT})\rho(\nabla \mathbf{u})k - C_2\rho\frac{\epsilon^2}{k} + S_\epsilon \tag{28}$$

where D_k represents dissipation and G_k represents generation of turbulent kinetic energy.

Initialization
Initial and boundary conditions for k and ϵ are set using these initialization equations:

$$k = \frac{3}{2}(I|\mathbf{u}_{ref}|)^2 \tag{29}$$

$$\epsilon = \frac{C_\mu^{0.75}k^{1.5}}{L} \tag{30}$$

where I is turbulence intensity, $\mathbf{u}_{ref}$ is a reference velocity (typically flow mean velocity), C$_\mu$ is a constant of 0.09 and L is a reference length scale. According to Versteeg and Malalasekera [2], the reference length scale L is $L = 0.07l$, where l is a characteristic length. The characteristic length for

Litjfosstunnelen is the hydraulic diameter: $l = D_h$.
There is no standard approach for initializing turbulence intensity for a rough tunnel, unless there are measurements available. The value is therefore guessed, based on a qualitative categorization from CFD-online wiki [9]. This source suggests a turbulence intensity of 5-20% for high-turbulence cases, which Litjfosstunnellen is likely to be. A value of 10% was chosen, quite arbitrarily. This might be a bit low, especially for k-Omega. This will be discussed later. k and ϵ are initialized

$$k = 0.00593331[m^2/s^2]$$
$$\epsilon = 0.00254224[m^2/s^3]$$

Weaknesses
A drawback for this model is its underlying assumption that the eddy viscosity, ν_T is isotropic, which makes it fail in complex flows due to its inacurate predictions.[2]

Boundary conditions
The inlet, internal field and outlet boundary conditions for k and epsilon are relatively straight-forward. At the inlet, a distribution based on initialization equations are given. At the outlet there should be a zero gradient boundary condition. An initial distribution in the internal field/free stream is given if available, otherwise a zero gradient condition is given here too. One option is to use the same free stream distribution as inlet distribution for a uniform field, and that is the approach used in this case. Initialization is based on available data or guesswork. The experience of the CFD analyst will make a difference. Initialization values for k and epsilon are typically small, but it is important they not be set to zero, or the denominator in the eddy viscosity equation is zero.
The most complicated boundary conditions are at the wall, and these are Reynolds number dependent. For high Reynolds numbers, the log-law is valid and the following wall functions relating shear stress, mean velocity, k and epsilon, are useful for a smooth wall:

$$k = \frac{u_\tau^2}{\sqrt{C_\mu}} \tag{31}$$

$$\epsilon = \frac{u_\tau^3}{\kappa y} \tag{32}$$

in addition to equation 21. [2] There are also wall functions available for rough walls, but as previously mentioned, the roughness of the lab model was undecided when this thesis was written. To make a random roughness guess was deemed no better than using smooth wall functions for the tunnel. It would only bring the solution closer to lab results by a lucky strike, and increased accuracy should not come for the wrong reasons. A third reason for using smooth wall functions is the roughness element size. The roughness elements are large, and will probably affect the flow much more than the (probably small) surface roughness would.

2.4.2 k-*omega* and k-ω SST (shear stress transport)

The k-Omega model has much in common with the k-Epsilon model. Like k-Epsilon it focuses on turbulent kinetic energy and solves an equation for k. The eddy viscosity, however, is defined a little differently, using a different length scale and the turbulence frequency $\omega = \epsilon/k$:

$$\mu_t = \frac{\rho k}{\omega} \tag{33}$$

dividing by density

$$\nu_t = \frac{k}{\omega} \tag{34}$$

The second equation solves for turbulence frequency or turbulence specific dissipation rate, ω. As the equations show, a larger omega leads to a lower eddy viscosity, which might enhance solution stability.

Boundary conditions
k-Omega's best feature compared to k-Epsilon is better behaviour for near wall flow. The solution is also not very sensitive to wall boundary conditions. Free stream, on the other hand, can prove problematic when $\omega- >0$, see equation 33 and 34. The solution is also assumption dependent, which is a source of uncertainty.

The third model tested in this thesis is the k-Omega SST model, a hybrid of k-Epsilon and k-Omega. The intention was to combine the best features from k-Epsilon and k-Omega:

- assumption insensitivity for internal field
- near wall robustness

k-Omega SST uses k-Omega equations near the wall and k-Epsilon equations in the inertia dominated flow farther from the wall. Model equations from OpenFOAM for k-Omega SST are shown below.

Turbulence specific dissipation rate

$$\frac{D}{Dt}(\rho\omega) = \nabla(\rho D_\omega \nabla_\omega) + \frac{\rho\gamma G}{\nu_t} - \frac{2}{3}\rho\gamma\omega\nabla\mathbf{u} - \rho\beta\omega^2 - \rho(F_1 - 1)CD_{k\omega} + S_\omega \quad (35)$$

Turbulent kinetic energy

$$\frac{D}{D_t}(\rho k) = \nabla(\rho D_k \nabla_k) + \rho G - \frac{2}{3}\rho k\nabla\mathbf{u} - \rho\beta^*\omega k + S_k \quad (36)$$

Turbulent viscosity, limiter

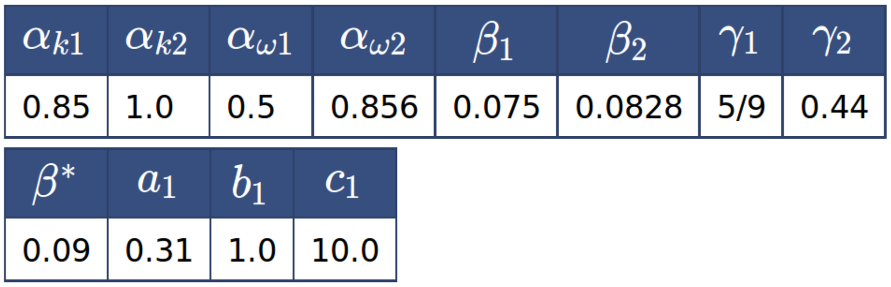
$$\nu_t = a_1 \frac{k}{max(a_1\omega, b_1 F_{23}\mathbf{S})} \quad (37)$$

The turbulent viscosity is a limiter to prevent accumulation of turbulence in stagnation regions. it limits the production of k. For details on this, see OpenFOAM documentation online. [7]

Initialization

$$k = \frac{3}{2}(I|\mathbf{u}_{ref}|)^2 \quad (38)$$

$$\omega = \frac{k^{0.5}}{C_\mu L} \quad (39)$$



α_{k1}	α_{k2}	$\alpha_{\omega 1}$	$\alpha_{\omega 2}$	β_1	β_2	γ_1	γ_2
0.85	1.0	0.5	0.856	0.075	0.0828	5/9	0.44

β^*	a_1	b_1	c_1
0.09	0.31	1.0	10.0

Figure 8: k-Omega SST in OpenFOAM: Constants for turbulent kinetic energy and specific dissipation rate equations

The initialization for k is the exact same as for k-Epsilon. Using the same reference length scale L as for k-Epsilon, the following value for ω is found:

$$\omega = 28.9731599[m/s^2]$$

3 Implementation

With data from the lab model and a basic theory foundation, the case is now ready to be implemented in OpenFOAM. The recommended procedure is to copy a tutorial case with flow conditions similar to one's problem and adapt the necessary files. Within tutorials there is a folder for incompressible flow problems, categorized by solver. Flow properties, assumptions and numerical method lead to the simpleFoam folder, containing flow problems solved by the SIMPLE method.

tutorials − > incompressible − > simpleFoam − > **caseFolder**

There are three folders in a case, **0**, **constant** and **system**. Case file structure can be seen in table 2. Bold face indicates folder, normal text indicates text-file. Examples of text-files are included in the attachments.

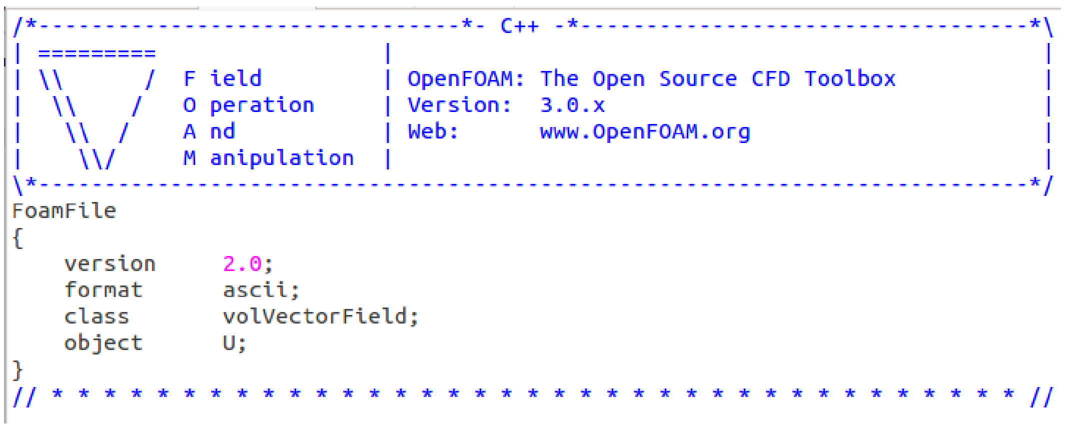
Initial and boundary conditions are defined in the 0/fileName files. Each transport property has a file with a logical name. The boundaries themselves are defined in the mesh dictionaries, blockMeshDict and snappyHexMeshDict. Numerical schemes are defined in fvSchemes, solver controls and under-relaxation are specified in fvSolution. Time and write control are specified in controlDict. Details are described under case folder sub-sections.

Case folder		
0	**constant**	**system**
U	transportProperties	blockMeshDict
p	turbulenceProperties	controlDict
k	**polyMesh**	decomposeParDict
epsilon	**triSurface**	fvSchemes
omega	- Litjfossen.stl	fvSolution
nut		meshQualityDict
		snappyHexMeshDict

Table 2: Case file structure

3.1 0

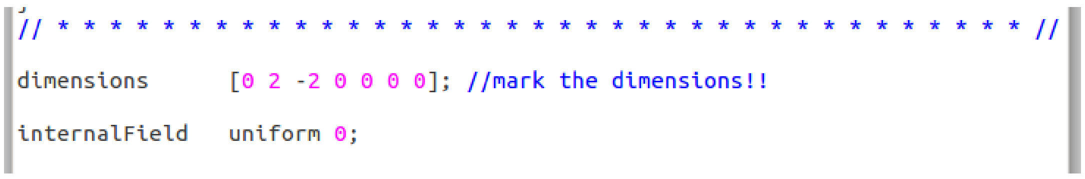
Boundary and initial conditions are already found, and need only be implemented in OpenFOAM. As previously stated, this is done in each flow property's own file in the 0 folder. This file belongs to a given class, like volVectorField for vectors (velocity) and volScalarField for scalars (pressure, k, etc), denoted in the header of the file.



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

Figure 9: Header in velocity file U

After the file header, the property dimensions are defined. Below is an example from a pressure (p) file, where the dimensions are m^2/s^2. Two slashes are used to insert comments. The internal field is then specified and given a uniform value of zero.

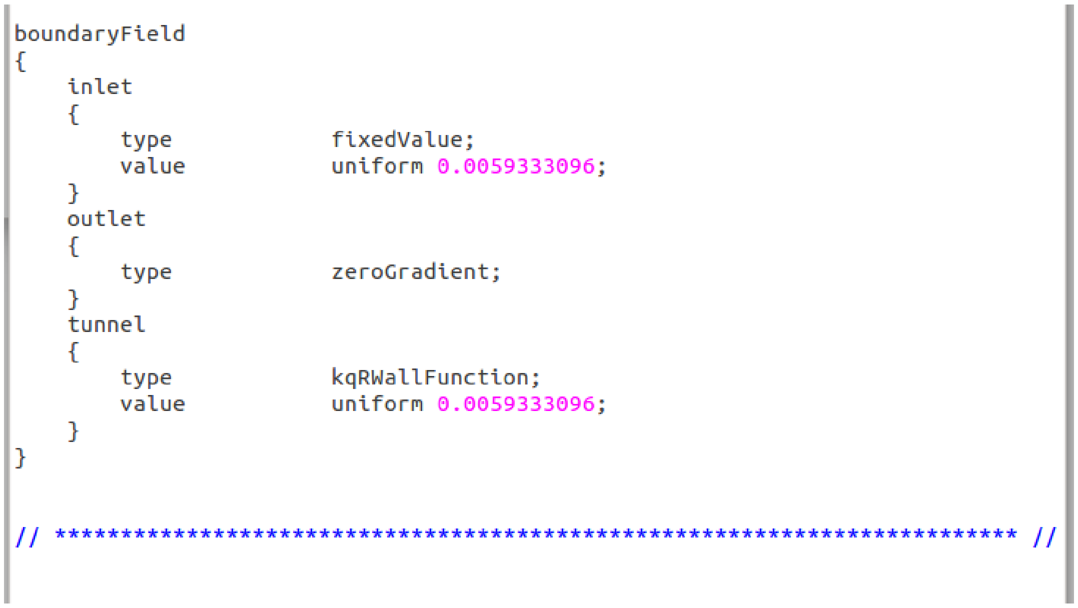


```
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0]; //mark the dimensions!!

internalField   uniform 0;
```

Figure 10: Dimensions and internal field, p

If this were a velocity file, the number would be replaced by a parenthesis with separate values for u, v and w: (u v w).

Below the internal field, the boundary fields are listed. An example from a k file is shown below. All boundaries, namely inlet, outlet and tunnel are listed and given a boundary condition type and, if suitable, a value. Boundary conditions have logical type names in OpenFOAM, so the meaning is intuitive. A complete overview of boundary conditions for all properties is shown in table 3.



```
boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.0059333096;
    }
    outlet
    {
        type            zeroGradient;
    }
    tunnel
    {
        type            kqRWallFunction;
        value           uniform 0.0059333096;
    }
}


// ************************************************************************* //
```

Figure 11: Boundary fields, k

Wall functions

OpenFOAM offers pre-defined wall functions for smooth and rough walls. The functions used in this case are kqRWallFunction for k, epsilonWallFunction for epsilon, omegaWallFunction for omega and nutWallFunction for turbulence viscosity.

OpenFOAM Boundary Conditions			
Property	Inlet patch	Outlet patch	Tunnel wall
U $[m/s]$	fixedValue (0 0.6 0)	inletOutlet	fixedValue (0 0 0)
p $[m^2/s^2]$	zeroGradient	fixedMean (0.0)	zeroGradient
k	fixedValue (0.0059)	zeroGradient	kqRWallFunction
ϵ	fixedValue (0.00254)	zeroGradient	epsilonWallFunction
ω	fixedValue (28.973)	zeroGradient	omegaWallFunction
ν_T	calculated	calculated	nutWallFuction

Table 3: Boundary conditions in OpenFOAM

3.2 constant

3.2.1 polyMesh

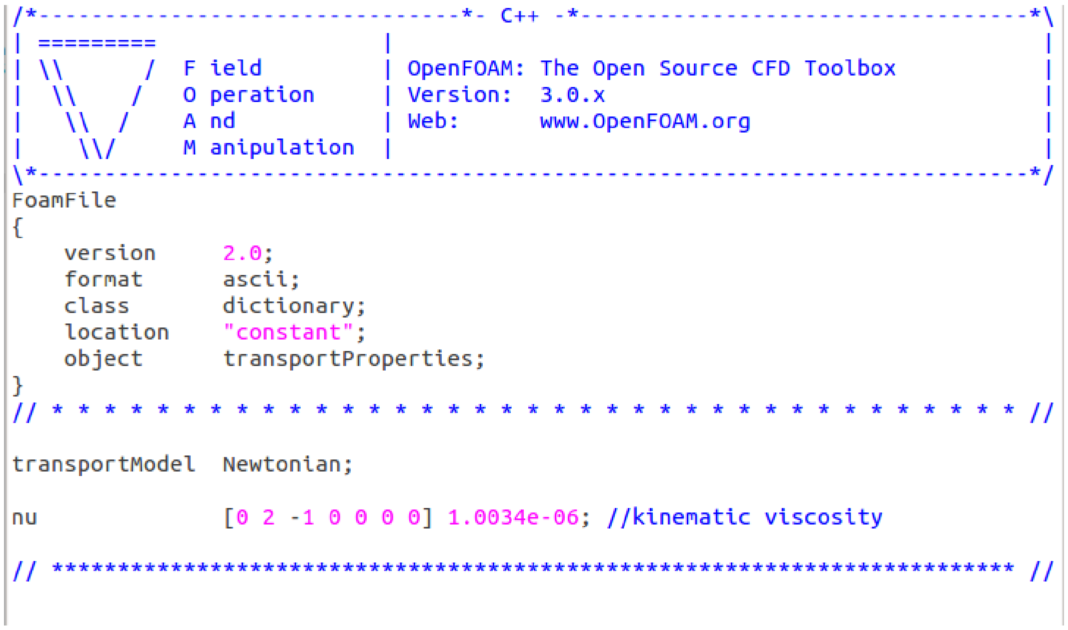
This folder is initially empty. When mesh is generated, mesh files are written and stored here. Most of there files are not as readable as other files, as they are filled with mesh data and not words: The largest files contain separate digits for each and every cell. For this case, that means more than 2 million entries. For checking mesh process and quality, it is better to write a log-file for a utility called checkMesh. This outputs number of cells, faces, checks for skewness and more.

3.2.2 triSurface

This folder is only needed for cases with complex geometries. This is where the complex geometry file is stored, in this case an STL file with scanned data from Litjfossen. Other geometry files can also be used, the reader is referred to OpenFOAM documentation for details on permissible file types.

3.2.3 transportProperties

There are two entries in transportProperties. The first is a transport model, which for our case is the Newtonian transport model. Below is the kinematic viscosity, ν, denoted nu, with dimensions and value. Given the lab water temperature of 20° C, the kinematic viscosity is set to 1.0034e-06 $[m^2/s]$.

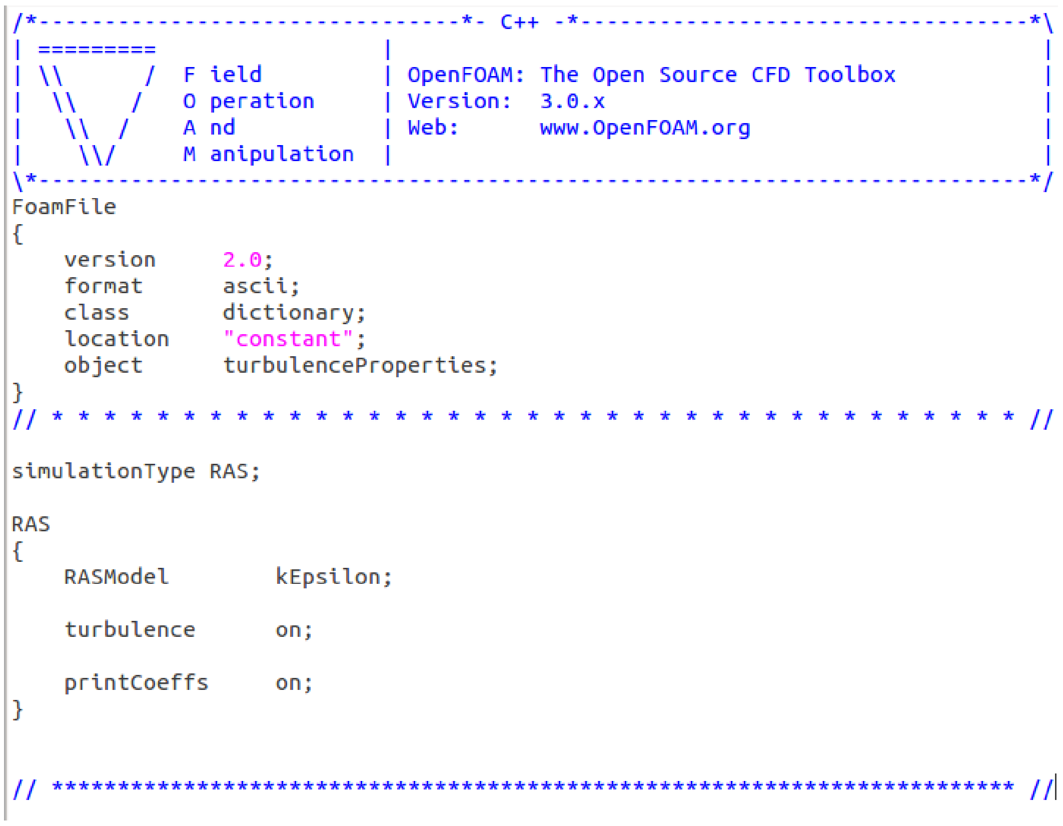


```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

transportModel  Newtonian;

nu              [0 2 -1 0 0 0 0] 1.0034e-06; //kinematic viscosity

// ************************************************************************* //
```

Figure 12: Dimensions and internal field, p

3.2.4 turbulenceProperties

Turbulence model implementation in OpenFOAM is surprisingly simple once the initialization of turbulence properties is done. First, a simulation type is specified, like Reynolds Average Simulation (RAS). A subdivision corresponding to the chosen simulation type follows, including name of turbulence model and on/off switches for turbulence and coefficient printing. Figure x shows turbulenceProperties for k-Epsilon for our case.



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType RAS;

RAS
{
    RASModel        kEpsilon;

    turbulence      on;

    printCoeffs     on;
}


// ************************************************************************* //
```

Figure 13: Dimensions and internal field, p

3.3 system

3.3.1 blockMeshDict

Mesh generation in OpenFOAM is controlled through mesh dictionaries. It starts with a background mesh defined in blockMeshDict. The background mesh envelopes the complex geometry. blockMeshDict lets the user specify

- scaling factor
- vertices
- blocks
- edges

- boundaries
- cell size

The background mesh for Litjfosstunnelen consists of one single block with hexahedral cells of uniform size (except the finest mesh, where lengths in x, y and z direction differ by a fraction of a millimeter). The mesh is orthogonal.

3.3.2 snappyHexMeshDict

blockMeshDict is enough for simple geometries, but a rock-blasted tunnel is not such a one. As mentioned in section 3.2.2, the complex geometry file is placed in **constant/trisurface**. The mesh dictionary for this geometry is on the other hand placed in **systemt**, and is called snappyHexMeshDict. This dictionary dictates the utility snappyHexMesh, which snaps the existing background mesh cells to the tunnel surface and throws out unnecessary cells. snappyHexMeshDict also controls features such as

- layering
- refinement
- patch type
- expansion ratio
- meshing iterations
- mesh quality controls
- ... and much more

The snappyHexMesh utility costs quite a lot of computation power/time for fine mesh, but finishes quickly for coarser mesh. See table 4 below. Once the mesh in finished, it should be studied in paraview, a post-processing tool that is described more thoroughly later. A visual presentation is very helpful when assessing mesh quality. An example of a fine mesh with one refinement layer at the wall can be seen in figure 14. Notice that cells in the refinement layer are not necessarily hexahedral. For the second finest mesh with 6898414 cells, about 1.6 million of them were prisms. Using the checkMesh utility will give detailed mesh quality data.

snappyHexMesh details			
Cell size	Number of cells	Time	CheckMesh
50	138257	60	OK
25	719579	250	OK
12,5	3930613	1216	Failed 1
10	6898414	2160	Failed 1
8	12481472	4000	Failed 1

Table 4: Size, cells, time and quality data for tunnel mesh

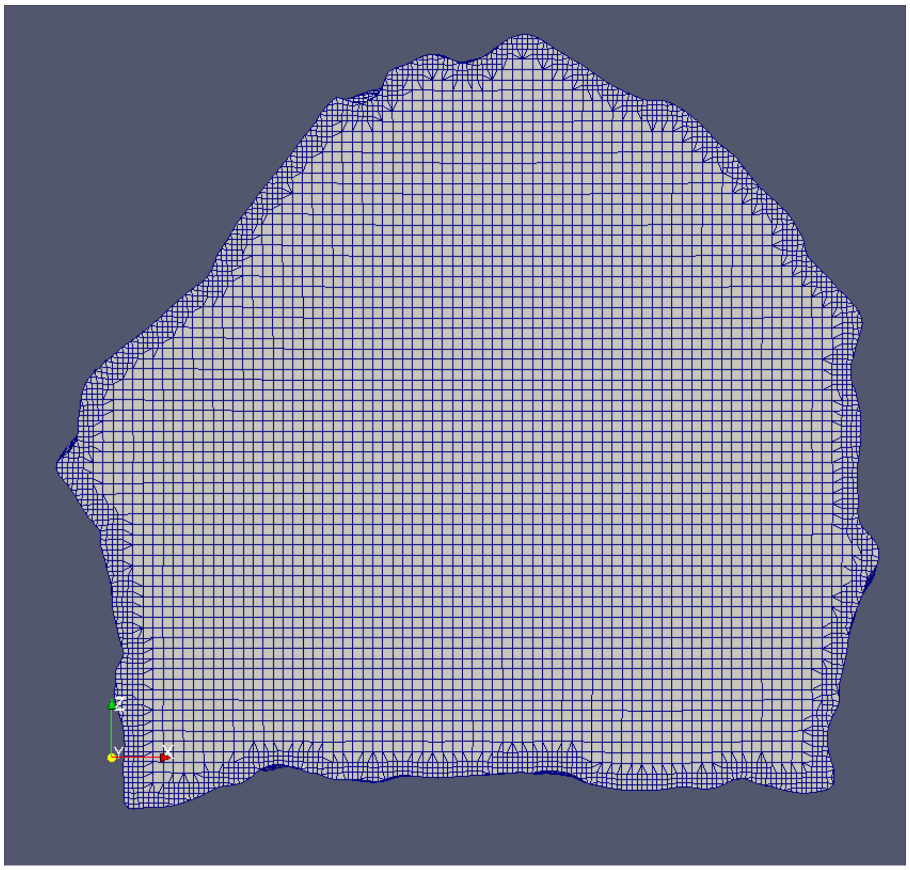


Figure 14: Mesh at inlet

If near-wall conditions were of particular interest, several refinement layers could have been added. The characteristic cell length is halved for each refinement layer. If possible, snappyHexMesh will avoid making skew faces. For such a large mesh it is however hard to avoid. The failed mesh checks for finer meshes are all because of a single face with max skewness approximately 0.1 above the limit. This is within reasonable deviation, particularly since it is only one face in a mesh of 3.9 million cells or more.

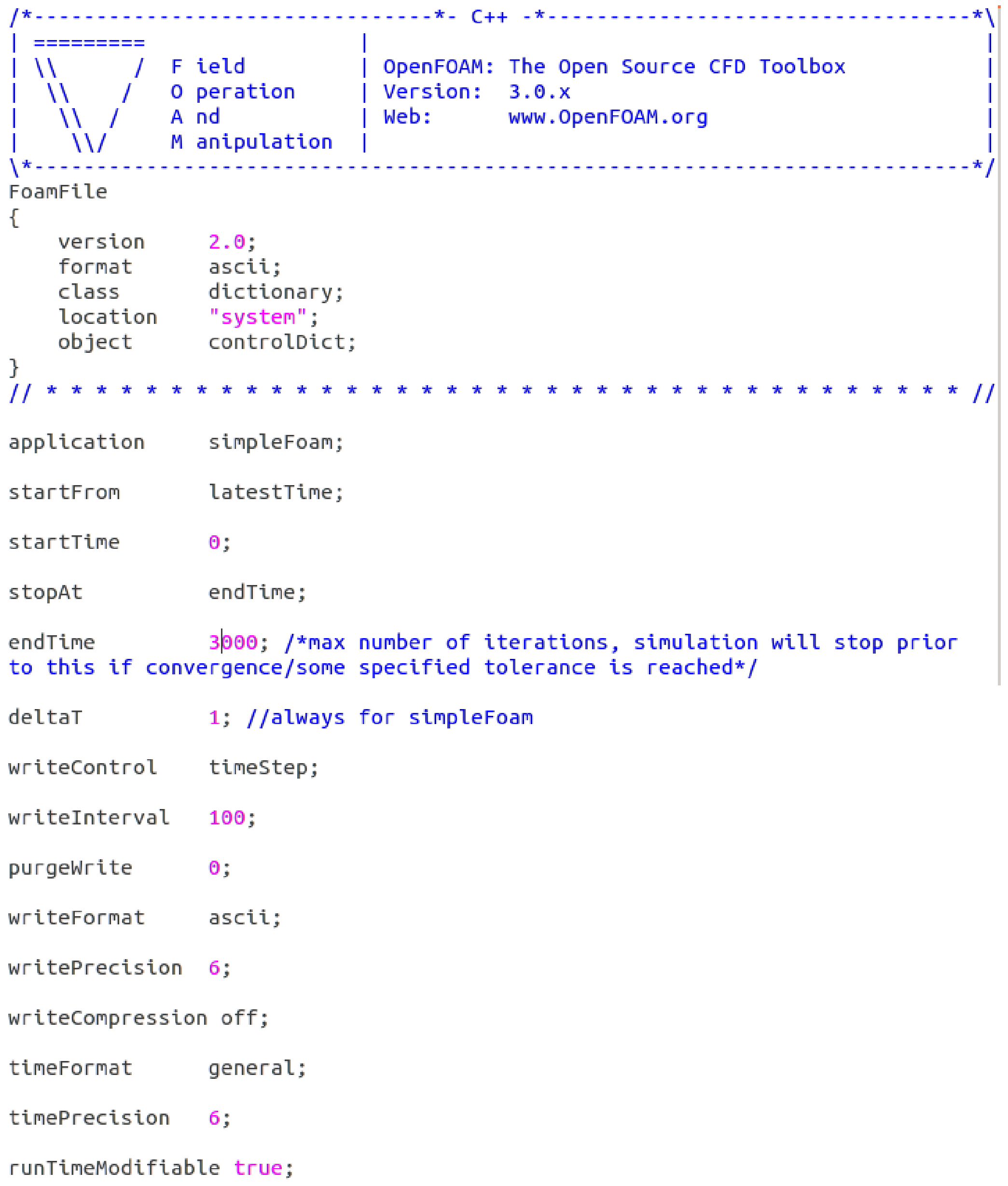
3.3.3 Time, iteration and write control

To define where to start and stop a solution, when to output data and in which format and with what precision, the user modifies the controlDict fie. An example from a k-Omega test run is shown in Figure 15. Entries are made for transient solvers and include words like startTime and deltaT. For the steady-state solver simpleFoam, the time step deltaT is not really a time step. If deltaT is 1 and endTime is 2000, this means that the simulation will stop after 2000 iterations (unless convergence criteria are met prior to this). A deltaT of 0.1 and endTime of 200 is also a setup for 2000 iterations. The only difference between the two is the time step continuity error. In all other respects, the solutions are identical.

If a solution oscillates and needs more iterations than expected to converge, it is easy to change the setup without having to restart the entire simulation. Set runTimeModifiable to true, increase endTime value and make sure that the entry for startFrom is latestTime. Then, when the solver utility is entered in terminal, it will pick up from the latest iteration.

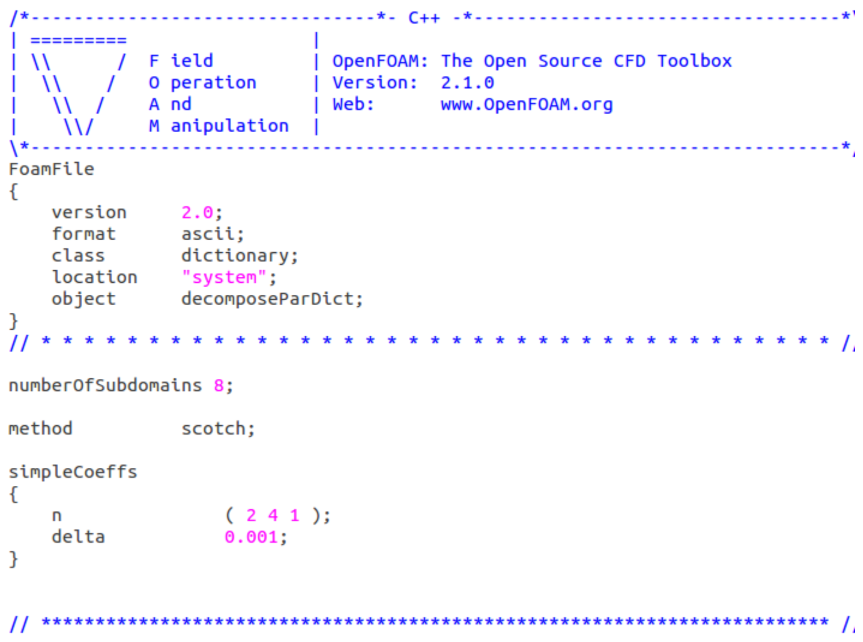
3.3.4 decomposeParDict

To speed up simulations a case can be split up into sub-domains and run on several cores in parallel, as many as the computer offers or as many is available to the user through a cluster. This process is called decomposition. There are many ways to do this too, the method used is called scotch and is



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     simpleFoam;

startFrom       latestTime;

startTime       0;

stopAt          endTime;

endTime         3000; /*max number of iterations, simulation will stop prior
to this if convergence/some specified tolerance is reached*/

deltaT          1; //always for simpleFoam

writeControl    timeStep;

writeInterval   100;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;
```

Figure 15: Iteration and write control in controlDict



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
|  \\    /   O peration     | Version:  2.1.0
|   \\  /    A nd           | Web:      www.OpenFOAM.org
|    \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 8;

method          scotch;

simpleCoeffs
{
    n               ( 2 4 1 );
    delta           0.001;
}


// ************************************************************************* //
```

Figure 16: decomposeParDict

recommended by OpenFOAM. Decomposition method and number of available CPU's are specified in deomposeParDict. The numberOfSubdomains entry is the number used in the utility command. it is important to remember the code-word parallel, or the problem will run on just one core even if it is decomposed. A decomposeParDict is show in Figure 16. When the simulation is converged, the case can be reconstructed using the reconstructPar utility.

3.3.5 fvSchemes

In OpenFOAM, numerical schemes are specified in the fvSchemes file. It contains the following subcategories:

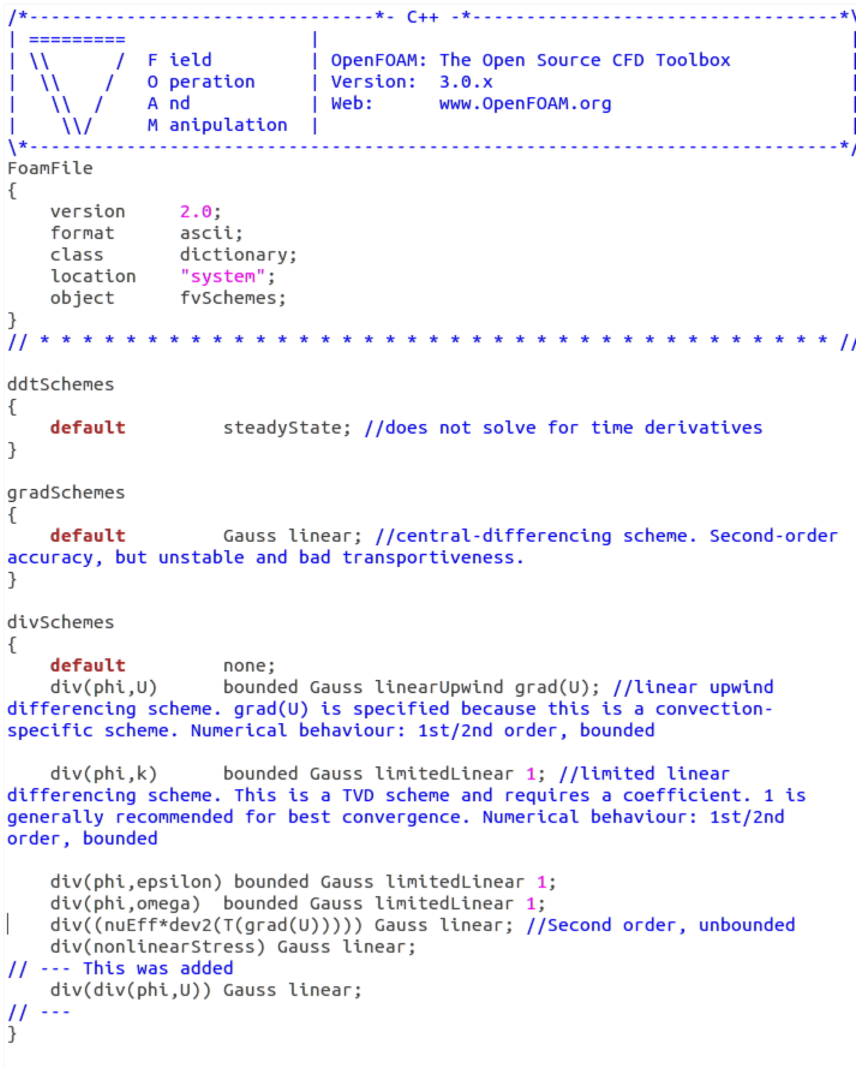
- timeScheme: first and second time derivatives.
- gradSchemes: gradient ∇ schemes

- divSchemes: divergence $\nabla\bullet$ schemes
- laplacianSchemes: Laplacian ∇^2 schemes
- interpolationSchemes: cell to face interpolations of values
- snGradSchemes: surface normal schemes: component of gradient normal to a cell face
- wallDist: distance to wall calculation, where required

Schemes can be specified differently for each property, or default schemes can be chosen to apply to all properties. Regardless, there is only one discretisation scheme available, that is the standard for finite volume discretisation: The Gaussian integration, for which an **interpolation scheme** must be specified. There are many interpolation schemes to chose from, both general and convection-specific. The convection-specific schemes require a flux field entry and sometimes (for TVD schemes) a coefficient, but the general schemes do not.
Interpolation schemes greatly affect the numerical behaviour of the solution. used in this case:

- Gauss linear. The linear interpolation scheme is equivalent to the central differencing scheme. It is second-order accurate, but is unbounded, can 'wiggle' and has poor transportiveness for high Peclet numbers. In this case it is used for all gradient schemes
- Guass linearUpwind. This is the linear upwind differencing scheme (LUD). Numerical behaviour: 1st/2nd order, bounded.
- Gauss limitedLinear. This is the limited linear differencing scheme. It is a TVD (Total Variation Diminishing) scheme and requires a coefficient. 1 is generally recommended for best convergence. Numerical behaviour: 1st/2nd order, bounded. This group of schemes is specially developed to avoid oscillation and unphysical values for turbulent flow. It is used for turbulence properties k, epsilon and omega.

Examples are shown in figure 17. Note that this is not a complete fvSchemes file!



```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  3.0.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         steadyState; //does not solve for time derivatives
}

gradSchemes
{
    default         Gauss linear; //central-differencing scheme. Second-order
accuracy, but unstable and bad transportiveness.
}

divSchemes
{
    default         none;
    div(phi,U)      bounded Gauss linearUpwind grad(U); //linear upwind
differencing scheme. grad(U) is specified because this is a convection-
specific scheme. Numerical behaviour: 1st/2nd order, bounded

    div(phi,k)      bounded Gauss limitedLinear 1; //limited linear
differencing scheme. This is a TVD scheme and requires a coefficient. 1 is
generally recommended for best convergence. Numerical behaviour: 1st/2nd
order, bounded

    div(phi,epsilon) bounded Gauss limitedLinear 1;
    div(phi,omega)  bounded Gauss limitedLinear 1;
    div((nuEff*dev2(T(grad(U))))) Gauss linear; //Second order, unbounded
    div(nonlinearStress) Gauss linear;
// --- This was added
    div(div(phi,U)) Gauss linear;
// ---
}
```

Figure 17: Numerical schemes in fvSchemes

3.3.6 fvSolution

Specifics for the numerical solution (solver controls) are found in fvSolution. This file contains solvers for all properties of interest, namely all that have their own file in **0**. Residual controls for the case specific algorithm are also specified here, in addition to relaxation factors for each property. For this thesis, velocity relaxation factors of 0.9 were used for the coarsest mesh and most well-behaved turbulence model, k-Epsilon. All other properties were given a relaxation coefficient of 0.6. For finer meshes using k-Omega, the solution did not converge in 2000 iterations and velocity relaxation factors were lowered to 0.8. This adjustment was not successful. k-Omega SST was even less convergent, which initially lead to an adjustment of boundary conditions to prevent recirculation at the outlet (from zeroGradient to inletOutlet). A lowering of the velocity relaxation coefficient from 0.9 to 0.7 was also tested.

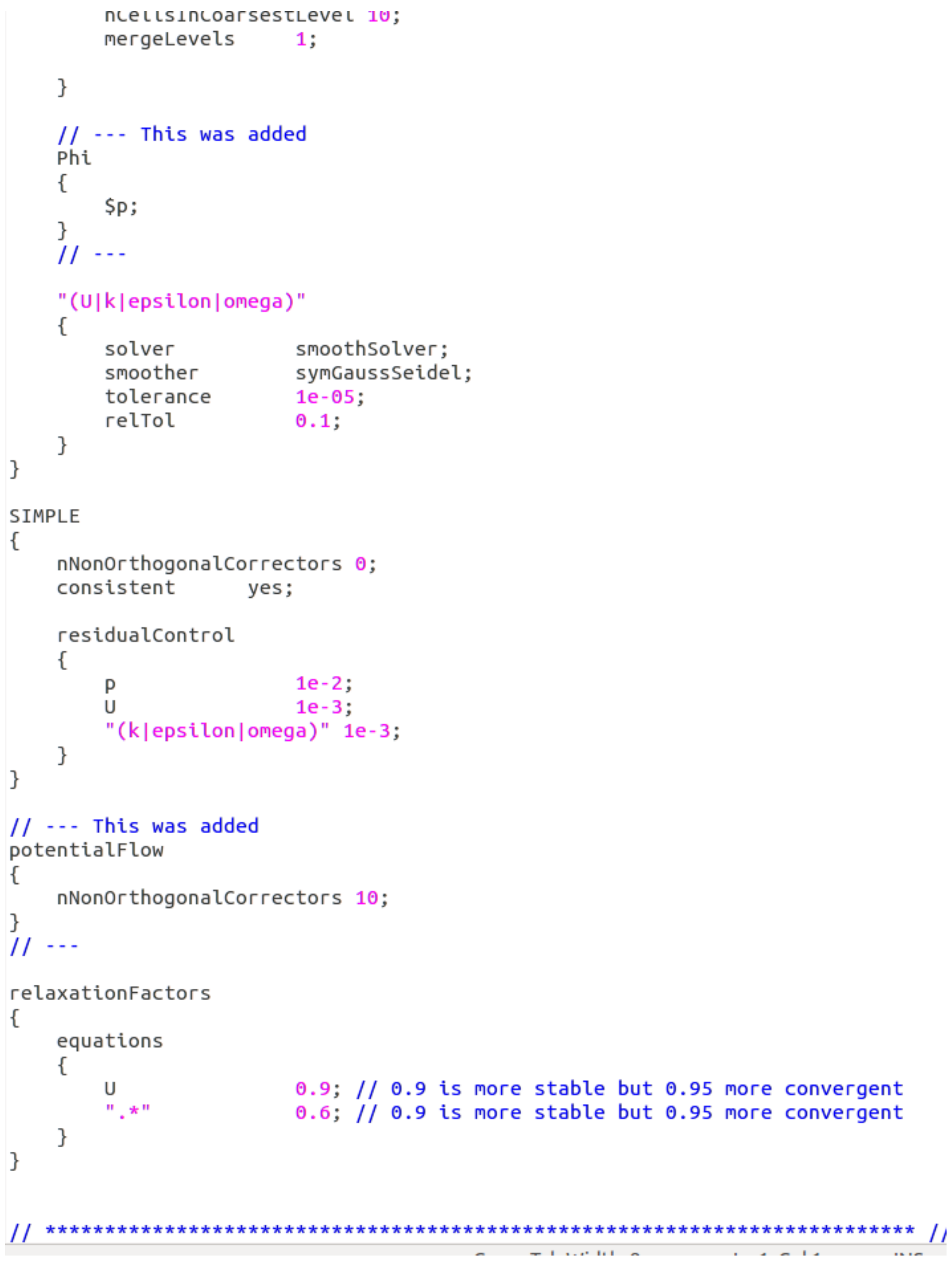
For most setups, residuals were as shown in Figure 18. The header and some solver controls were excluded from this figure, but can be studied in detail in the attachments. For some of the later setups, the residualControl for p was set to the same value as for the other properties. These simulations did not converge, but that is more likely because of other problems (oscillation) in k-Omega and k-Omega SST.

[6]

3.4 Workflow

When the implementation is complete, the case is ready to be run. Each step is executed using OpenFOAM utilities, commands in terminal with names corresponding to their funtions in OpenFOAM. It is always good practice to make sure a case is "clean" before running it. There are two important cleaning functions, one that deletes all mesh files from **constant/polyMesh** and one that deletes output folders. (Output folders appear when a simulation is running, according to specified write controls in controlDict. Their names correspond to a time step or iteration number.)

The next step is to create the mesh. First the background mesh defined in blockMeshDict, then the final mesh for the tunnel geometry. When the mesh-



```
        nCellsInCoarsestLevel 10;
        mergeLevels     1;

    }

    // --- This was added
    Phi
    {
        $p;
    }
    // ---

    "(U|k|epsilon|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-05;
        relTol          0.1;
    }
}

SIMPLE
{
    nNonOrthogonalCorrectors 0;
    consistent      yes;

    residualControl
    {
        p               1e-2;
        U               1e-3;
        "(k|epsilon|omega)" 1e-3;
    }
}

// --- This was added
potentialFlow
{
    nNonOrthogonalCorrectors 10;
}
// ---

relaxationFactors
{
    equations
    {
        U               0.9; // 0.9 is more stable but 0.95 more convergent
        ".*"            0.6; // 0.9 is more stable but 0.95 more convergent
    }
}


// ************************************************************************* //
```

Figure 18: Numerical solver controls in fvSolution

ing is done, mesh quality is checked using checkMesh utility as mentioned in section 3.3.2. Meshing is followed by decomposition. Before simpleFoam is run it can be beneficial to use a utility called potentialFoam (potentialFlow must be included in fvSolution for this to be available), which initializes fields to a better initial condition. This can help the SIMPLE algorithm in converging faster.

The main event is next, using the utility simpleFoam to solve the flow problem. Notice the extra code for parallel simulation and log files. A log file comes in very handy in post-processing and should always be written for the solver. Log files can also be made for checkMesh and other utilities.

If the case converges, the sub-domains are reconstructed. The processing is done, and the results can be post-processed and visualized. paraView is the software used for this thesis. The last command opens the case in paraView. Another option is to type paraFoam. The author's experience is that this is slower than the alternative listed below.
List of commands in order of appearance

- foamCleanPolyMesh
- foamCleanTutorials
- blockMesh
- snappyHexMesh -overwrite
- checkMesh
- decomposePar
- mpirun -np 8 potentialFoam -writep -parallel
- mpirun -np 8 simpleFoam -parallel > log | tail -f log
- reconstructPar
- touch case.foam | paraview case.foam

Gnuplot can be used to plot results from the simulation. This has not been done to a great extent for this thesis, since this kind of presentation looses a lot of information compared to 3D visualizations. Still, it is a great tool for checking for oscillations in the solution, and qualifies for a workflow list as well:

- gedit log
- foamLog log
- gnuplot
- set logscale y
- plot 'logs/filename' using 1:2 with lines
- — OR: plot 'logs/filename' u 1:2 w l
- reset
- exit

4 Post-processing

4.1 Pressure loss

The initial goal was to find pressure loss from one cross-section to another using mean pressure values for each section. Some post-processing can be executed in Terminal using OpenFOAM utilities, but this is cumbersome compared to post-processing software with graphical user interface. Also, a visual presentation of data has several advantages, like immediate intuitive perception and effectiveness (one picture says more than a thousand words). Also it is more practical, which lowers user threshold. The list of advantages goes on, but there is reason for caution. The human mind is susceptible to accepting nonsensical results with no physical meaning, only because the colorful presentation ”looks professional” or ”looks reliable”. The user should therefore approach visual post-processing with humility and insight into human weaknesses. With the right mindset, the benefits can readily be harvested.

When the solution is loaded in paraView, the first step is to make a slice for the cross-section of interest. This is done by applying a slice filter. The user specifies orientation and origin of the slice. When the slice is applied, a flow property can be chosen. The properties must be checked off in boxes in the menu to the left, and will appear in a drop-down menu on the toolbar. For pressure p and velocity U, there are two different alternatives, one is a point value and one is a cell value. If the point value is chosen, cell values are interpolated to make a smoother visualization of the property. This can introduce some errors, but is perfectly fine for making pictures. However, if the mesh is fine, the difference will not be as clear, and one might as well chose the cell presentation instead. Cell presentation uses each cell’s actual value from the solution, and does not introduce interpolation errors. Warning: The slice itself will cut cells! If mathematics filters are applied to a slice, there will be interpolation effects on the results! This won’t have dire consequences unless there are very steep gradients present, but is an important thing to be conscious of.

4.2 Friction factors

4.2.1 Darcy friction factors

There is an OpenFOAM utility called patchAverage which calculates the average of a property on a pre-defined patch. For this case it could be inlet and outlet, for instance. This gives more accurate results than paraView methods, but with the drawback of only being applicable to patches. paraView slices can be applied anywhere. The filter used for finding cross-section/slice average values is called Integrate Variables. It does not give the average value directly, but integrates any property of the users choosing over the area of the slice (or other shape) it is applies to, and also returns the area of the cross-section. The average can then be found by dividing integrated variables by cross-section area.

$$\frac{\int p dA}{A} = p_{average} \tag{40}$$

When the average value for different cross-sections are found, the difference can be inserted into the rearranged Darcy Weißbach equation:

$$f_D = \frac{\Delta p}{\rho} \frac{2}{L} \frac{D}{U^2} \tag{41}$$

where L is the distance between the cross-sections and D is a characteristic length, in this case the hydraulic diameter. Three slices are chosen for this case, at 4, 6 and 7 meters downstream of the inlet.

There is a downside to this very simple approach, being that the hydraulic diameter is not constant, and neither is the velocity. A quick analysis using the given hydraulic diameter from the lab and the corresponding main velocity gave negative pressure loss from cross-section 6 to cross-section 7. This is also visibly in a plot with rescaled legend for pressures in paraView, see Figure 19. Cross-section at 4m has highest pressures, while the lowest pressures are at 6m. This calls for a more thorough investigation.

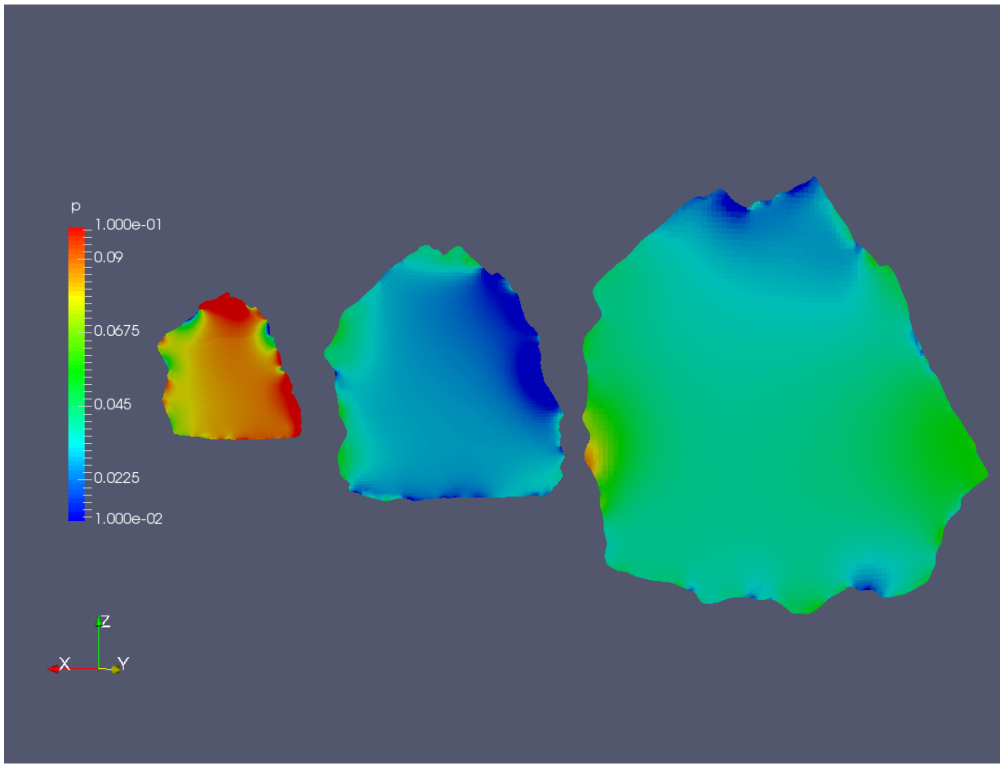


Figure 19: Pressure increase in downstream direction

Total loss is found through the energy equation [3]

$$\left(\frac{p_1}{\gamma}+\alpha_1\frac{\overline{U_1^2}}{2g}+z_1\right)+h_p=\left(\frac{p_2}{\gamma}+\alpha_2\frac{\overline{U_2^2}}{2g}+z_2\right)+h_t+h_L \tag{42}$$

Cancelling z_i, h_p and h_t and rearranging yields an equation for head-loss h_L

$$h_L=\left(\frac{p_1}{\gamma}+\alpha_1\frac{\overline{U_1^2}}{2g}\right)-\left(\frac{p_2}{\gamma}+\alpha_2\frac{\overline{U_2^2}}{2g}\right) \tag{43}$$

which relates to the head-loss Darcy Weißbach equation

$$f_D=h_L\frac{2g}{L}\frac{D}{\overline{\overline{U^2}}} \tag{44}$$

The mean pressures and velocities are found using Integrate Variables like before. Inserted into equations above, the head loss is found. There is still the matter of choosing a representative U and D_h for the head loss form of Darcy Weissbach. The mean hydraulic diameter in the lab model is used for D_h. The corresponding mean velocity U was therefore also used for the sake of consistency. The remaining input needed for the Darcy Weißbach equation is head-loss (equation 43).

The finest mesh for each turbulence model was used to find friction factors for the three cross-sections 4-6, 4-7 and 6-7 meters downstream of inlet shown in Table 5.

Darcy Friction Factors f_D			
Section	k-Epsilon	k-Omega	k-Omega SST
4-6	0,0499	0,0503	0,0477
4-7	0,0459	0,0458	0,0401
6-7	0,0379	0,0369	0,0250
$f_{D,mean}$	0,0446	0,04433	0,0376

Table 5: Friction factors

Friction factors are also found for other grids. A spread-sheet is attached with details. Averaged friction factors are shown in table 6.

Averaged friction factors			
Grid	k-Epsilon	k-Omega	k-Omega SST
3,33cm	0,04308	0,04237	0,03761
1,67cm	0,04487	0,04433	not found
0,83cm	0,04506	not found	not found
0,53cm	0,04459	not found	not found

Table 6: Friction factors averaged between sections

A more accurate approach than the one described here would be to use cross-sections in immediate proximity of each other. For a discrete case like this, they could not be less than one cell-length apart. The deviations for A and U would then be small, and the friction factor applies in that location alone.

Pressure and velocity for head-loss equation, k-ϵ, fine mesh			
	Slice 4m	Slice 6m	Slice 7m
$\overline{p}/\rho[m^2/s^2]$	0,085392	0,025957	0,038419
$\overline{U}[m/s]$	0,600874	0,621546	0,570834
A $[m^2]$	0,132192	0,127888	0,139230
$p/\gamma[m]$	0,008539	0,002645	0,003916
$U^2/2g[m]$	0,018402	0,019690	0,016608

Table 7: Head-loss data

This could also be done for a large number of cross-sections to find the Darcy friction factor throughout the tunnel as a function of y (flow direction). An implementation of this is not complicated in theory. Data from slices i paraView can be imported to a spread-sheet, and the operations are readily available given the simplicity of the mathematics.

This approach would also have been used for this thesis were it not for the memory needed to load so many slices in paraView. In an attempt to make 800 slices over the 8 meters of the tunnel, the program aborted. An image of 80 slices and integrated variables data can be found in attachments.

The more accurate approach would require a local characteristic length. The characteristic length for a cross- section is

$$D_h = \frac{A_w}{P_w} \tag{45}$$

where A_w is wetted area of cross-section and P_w is wetted perimeter. Integrate Variables filter in paraView finds cross-section area, as described, while the perimeter is found by applying a slice filter only to the wall and not the internal field. Integrate Variables (point data) will now give the circumference.

4.2.2 Manning Strickler values

Head-loss can also be used to find Manning Strickler values, see equation 1. An example is shown below.

$$I = \frac{h_L}{L} \tag{46}$$

values for I are available in the spread-sheet attachment. For k-Epsilon, they range from I=0,00205 to I=0,0215, the latter being closest to the finest resolution result of I=0,0213. I=0,0213 is used for this example. Using geometry and flow data as presented above, the Manning Strickler value becomes:

$$M_m = \frac{Q}{AR_h^{2/3} * \sqrt{I}} = \frac{0,1}{0,159 * (0,211)^{2/3} * \sqrt{0,00213}} = 36,6[m^{1/3}/s] \tag{47}$$

for the lab model.

5 Verification and validation

The two dominating criteria for a successful simulation result are

- convergence
- grid independence

Convergence is, as previously stated, reached when a pre-defined criterion for error/residuals is satisfied. As the reader can see in Table 8, convergence reached for modelling setups, not all test cases have converged. This can mean that convergence criteria are too strict, making convergence unlikely, or that the solution would converge given more time and iterations, or that the solution is oscillating/divergent. A residual plot for the first non-convergent k-Omega test case revealed that there were oscillations. The author was not able to detect why, or to stop it from happening, but these measures were tested in an unsuccessful effort to make the case more convergent:

- Changing the outlet boundary condition, from zeroGradient to inletOutlet for velocity, to prevent recirculation at the outlet. This boundary works as a zeroGradient BC if there is no recirculation.
- Increasing the number of iterations to 3-4000
- Lower the under-relaxation factor. This was described in section 3.

CFD Model Setups			
Grid cell size	k-Epsilon	k-Omega	k-Omega SST
3,33 cm	Yes	Yes	Yes
1,67 cm	Yes	Yes	Yes
0,83 cm	Yes	Yes	No
0,67 cm	Yes	No	No
0,53 cm	Yes	No	No

Table 8: Convergence reached for modelling setups

5.0.1 Errors

As previously stated, numerical solutions to complex flow problems are always erroneous, to a degree. Even if this weren't so, there is no use expecting perfect solutions from a CFD analysis. A CFD analyst must accept errors while simultaneously making an effort to reduce them. Sources of error are divided in three

- Numerical errors
- Code errors
- Human errors

Human errors are not only possible, but probable for complex cases. A decimal point deviation can make a great impact on a solution. Problem discretisation introduces errors, which may be reduced by mesh refinement and time-step reduction, but only for those who can afford the expense of it. There can be errors in the program code, and computational science is limited in that a digital number is represented by a finite number of digits, which results in round-off errors. It is unreasonable and insensible to spend time and computational power on an infinite number of iterations when the error, or residual, is small enough for the results to be useful in practice. Therefore, it is better to use some convergence criterion, a delimiter for when the solution is "close enough" to the correct solution. A small sum of absolute values of residuals is a typical convergence indicator. The change in error from one mesh to another can be estimated using GCI, see section 5.1.2.

5.0.2 Grid Convergence Indicator

To find whether or not a simulation is grid independent, one must have at least to different grids to compare, preferably more. The first method used in this thesis is described in course material for TVM4155, *Numerical Modelling and Hydraulics* by Olsen (2017) [8]. The equations below are slightly rewritten for disambiguation.

$$GCI^{CF} = \frac{1,25e^{CF}}{r^{CF}} \tag{48}$$

$$e^{CF} = \left| \frac{\Phi_F - \Phi_C}{\Phi_F} \right| \tag{49}$$

$$r^{CF} = \frac{h_c}{h_f} \tag{50}$$

where Φ is a result from the simulation. The result of interest here is the Darcy friction factor, so this is the value substituted for Φ. r^{CF} is the grid cell relation between the grids under investigation. These are preferably a mean value computed from all cells in a mesh. For heuristics case, the cell size of the background mesh was used in this case, since they are hexahedral cells of equal length in all three planes. The exception is the finest mesh, but deviations are very small and a mean value for each cell instead of for the entire mesh was used in this case.

Grid Convergence Indicator		
Grid	k-Epsilon	k-Omega
0,53-0,67 cm	0,0495	not converged
0,67-0,83 cm	0,0211	not converged
0,83-1,67 cm	0,0499	0,05526

Table 9: Grid convergence indicators

5.0.3 Uncertainties

As established in section 1, the results from this thesis cannot be validated until measurements from the lab are in place. Therefore this is merely a brief qualitative discussion on sources for uncertainty i CFD modelling. The main sources are

- input uncertainty: geometry, BCs and fluid properties
- physical model uncertainty

Input uncertainty for this case is most likely associated with turbulence properties, k, ϵ and ω, since the other properties are based on an actual setup an not poor initialization guesswork by an inexperienced turbulence analyst. Since convergence was not reached despite efforts described in section 5.1, it would be interesting to see what initialization would do to the solution. As mentioned, k-Omega is more sensitive to initialization than k-Epsilon. A more qualified choice of turbulence intensity is suggested.

The geometry of the CFD model will be slightly different from the lab model because of the windows, roughness and a general smoothing of the surface.

Fluid properties in the CFD model are simplified compared to the real world, as described in section 1.
The boundary conditions should be quite suitable for this case.

6 Conclusion

As shown in Table 8, k-Epsilon performs best by far of the three turbulence models that were teste in this thesis. Instabilities in k-Omega and k-Omega SST resulted in oscillations, and model setup tweaks did not result in convergence. That being said, the convergent k-Omega and k-Omega SST setups give results for friction factors that are in the same order of magnitude as k-Epsilon. General flow patterns are similar. Darcy friction factors and a model Manning Stricklef value are computed. Darcy friction factors in k-Epsilon are generally higher than for the other models. Mesh refinement leads to higher friction factors with one exception(Table 6). Within the tunnel, section 4-6 (meters downstream from inlet) generally yield higher friction factors than section 6-7. This is a general feature and does not depend on turbulence model. Grid independence is not reached, with a GCI of $\tilde{5}\%$ for comparison of the two finest grids.

Further investigation is advised. Suggestions to modifications include initial conditions for ω and interpolation schemes to avoid wiggles in the solution. Friction factors should be compared to physical model study for validation.

All attachments are submitted separately on a transportable storage device.

References

[1] Lovdata, 2017 *https://lovdata.no/dokument/SF/forskrift/2009-12-18-1600*

[2] H. K. Versteeg, W. Malalasekera, 2007, *An Introduction to Computational Fluid Dynamics*, Pearson Educational Limited, 2nd edition

[3] C. T. Crowe, D. F Elger, B. C. Williams, J. A. Roberson, 2010, *Engineering Fluid Mechanics*, John Wiley & Sons, Inc., 9th edition

[4] N. R. B. Olsen, 2012 *Numerical modelling and hydraulics*, Department of Hydraulic and Environmental Engineering (NTNU), 5th edition

[5] CFD Support Ltd., 2017, *OpenFOAM Training by CFD Support*

[6] OpenFOAM, 2017, *http://openfoam.com/documentation/user-guide/*

[7] OpenFOAM, 2017, *http://openfoam.com/documentation/cpp-guide/html/guide-turbulence-ras-linear-eddy-viscosity-models.html*

[8] N. R. B. Olsen, 2017 *Numerical modelling and hydraulics*, Department of Hydraulic and Environmental Engineering (NTNU), 3rd edition

[9] CFD-online, 2017, *https://www.cfd-online.com/Wiki/*

[10] Radek Maca, 2017 *Personal communication*

[11] NTNU, 2016-2017, *Lab model research associates*

[12] Sintef, 2016-2017, *Lab model production associates*