SIMULATION OF MULTIPHASE FLOW USING NON-NEWTONIAN FLUIDS AND SAND SEGREGATION IN OPENFOAM

Peter Dobbe

July 2021

DELFT UNIVERSITY OF TECHNOLOGY

Faculty of Mechanical, Maritime and Materials Engineering

Department of Marine and Transport Technology

Section Offshore and Dredging Engineering

TU Delft

Delft University of Technology

SIMULATION OF MULTIPHASE FLOW USING NON-NEWTONIAN FLUIDS AND SAND SEGREGATION IN OPENFOAM

By Peter Dobbe

In partial fulfilment of the requirements for the degree of Master of Science Offshore and Dredging engineering at the Delft University of technology.

Student number:	4186273
Graduation committee:	Prof. dr. ir. C. van Rhee
	Dr. ir. G.H. Keetels
	Dr. ir. A.M. Talmon

Delft University of Technology

Acknowledgements

This report is the final conclusion to my master's studies. It wraps up the Master Offshore and Dredging Engineering with a specialization in Dredging Engineering, Trenching & Deepsea Mining at the faculty of Mechanical, Maritime and Materials Engineering (3mE) at Delft University of Technology.

A big thanks is owed to the graduation committee for their supervision and support throughout this research. Their feedback and ideas were of great help in our valuable discussions throughout this project. A big thank you especially goes out to Cees van Rhee, chairman of the committee and daily supervisor, for this opportunity, his continuous encouragement and his time investment in guiding me.

Additionally, I am grateful for my wife, family, friends, and colleagues for their help, moral support, and moral compass. During the entirety of my study, your encouragement is exactly what I needed.

Peter Dobbe

Delft, July 2021

Abstract

In the field of mining and dredging engineering, oftentimes slurries are involved in transport. Understanding the behaviour of these slurries in a flow is important as the energy and water consumption need to be determined. The slurries have an interesting rheology due to the presence of clay. The rheological parameters cause the flow to be non-Newtonian. Further, the presence of coarse solids (sand particles) in these slurries also influences the rheological parameters. Through Computational Fluid Dynamics (CFD) simulations, these aspects can be predicted.

Previous work in non-Newtonian CFD with coarse solids made use of a free surface flow through a rigid-lid approach (van Rhee, 2017). A shortcoming of this is that the flow needs to be uniform and the flowdepth needs to be known beforehand. In the mining and dredging fields, this is not the case and a different approach is required.

This report will focus on multiphase CFD simulations. One of the fluids will be a non-Newtonian model including sand particles, and the other fluid will just be air. The sand particles will be subjected to transport, segregation, and settling behaviour.

OpenFOAM is the weapon of choice, but as it stands it does not have a solver that's capable of including sand particles. The `interFoam` solver is chosen as a starting point and it's code is adjusted. A Bingham Plastic transport model is implemented, including sand particles. A sand transport equation is also implemented.

Using the adjusted `interFoam` solver, two sets of simulations are ran. A first set of simulations is performed using a 2D mesh for an open channel. The first simulation utilizes a Bingham Plastic fluid without any sand particles included. The resulting velocity profile is compared to the analytical solution and is found to agree well. The second open channel simulation includes sand particles. The results show a sand bed forming as well as the sand concentration to be roughly constant throughout the plug flow. This is compared to experimental work and concluded effects is captured well from a qualitative perspective.

The second set of simulations is performed using a 3D mesh for a pipe section. These results were not as satisfying as the results for the 2D open channel. Unfortunately, all 3D pipe simulations either showed the pipe to fully fill up with the Bingham Plastic fluid, or the solver crashed at some point. Many attempts have been made at this, and no conclusive reason has been pointed out as the cause for this.

It's recommended to continue research in this direction with the main focus being the simulation stability.

Contents

1. Introduction

1.1. Problem domain

In mining engineering, waste materials left over after separation processes are often referred to as mine tailings. The tailing is a fluid generally consisting of water, mud and sand. These mixtures are deposited into large basins bounded by dams. The behaviour and properties of these mixtures determines the process of deposition.

In dredging engineering similar fluid mixtures are used in land reclamation. It is essential that the fractions remain mixed while they are deposited. Segregation of the particles could be decremental to the bearing capacity of the land.

In both engineering fields the commonality is that these slurries have a high solids content. These solids in turn influence the rheology and flow behaviour. Due to the presence of clay, the rheological parameters cause the flow to become non-Newtonian. Further, the sand particles present in the mixture also influence the rheological properties.

Any fluid generates shear stresses when flowing or deforming. This shear stress in turn has its effects on the settling of suspended particles like sand. Segregation of these particles leads to an inhomogeneous concentration of solids in the mixture. This in turn leads to inhomogeneous rheological properties which affects the flow field parameters (Hanssen, 2016), (Slatter, 2011).

Different analytical, numerical and empirical models have been developed to predict non-Newtonian flow behaviour and particle segregation (Spelay, 2007), (van Rhee, 2017).

Simulating slurry flows in these two engineering fields, with these behaviours, would require software that takes into account all the factors. The work performed by (van Rhee, 2017) pertained to the adaptation of Computational Fluid Dynamics (CFD) package OpenFOAM to simulate the settling of particles under shear and the influence on the rheological properties. One of the recommendations put forward in this work was to add a free mixture surface to simulate settling behaviour in open pipes and open channel flows. This thesis intends to follow this recommendation and that recommendation can be considered the starting point of this thesis.

Having that adaptation to OpenFOAM in place would allow for simulations with the combination of non-Newtonian behaviour, the influence of suspended and segregating sand particles on the rheology, and a free mixture surface.

1.2. Objective and problem definition

The objective of this thesis is to continue the work on the development of OpenFOAM. The research problem can be defined as follows: "*How to incorporate non-Newtonian properties and sand particle segregation in combination with a free mixture surface using OpenFOAM?*".

1.3. The approach of the research

This thesis contains of two parts. Firstly, a literature study was been performed to have a high level understanding on the problem domain and related questions, this is *Part 1: Analysis and Literature Research*. This includes research into non-Newtonian models, particle segregation models and flow theory as those are factors at play in the problem domain.

Secondly, *Part 2: Implementation and simulations* goes into the implementation and simulation in OpenFOAM. Firstly, an adaptation to Open FOAM will be implemented, after which this adaptation of the code is used in various simulations. The results of these simulations will be compared against (empirical) results found in literature.

1.4. Thesis structure

The structure of this report lines up with the approach as described in section 1.3.

Within Part 1, chapter 2 explores the factors and theory at play in the problem domain and chapter 3 gives an overview of previous research in the area. This includes non-Newtonian models and particle segregation models.

In Part 2, chapter 5 presents the adjustments made to OpenFOAM. This includes customization of a solver, using custom viscosity models, and sand settling models. Next, simulations are performed and validated against existing experimental work in chapter 6. Finally, chapter 7 concludes this report by presenting conclusions and recommendations for future work.

Part 1: Analysis and Literature Research

As a starting point, this part provides a general overview of the fundamental theory of slurry transport and tailings. The factors and theory at play in the problem domain are explored. Further, experimental research in this field will be reviewed.

2. Literature & theory

2.1. Rheology

Rheology is a field of study that examines deformation and stresses of fluids under flowing conditions. In order to accurately predict the flowing behaviour it is essential to understand the relation between these factors. The relation between shear stress and shear rate can be shown in rheogram. The gradient of this relation is what's called the viscosity, and is a measure of the stickiness of the fluid. For example, honey is quite sticky and has a high viscosity whereas water has a low viscosity and is quite runny.

2.1.1. Rheological models

For all fluids, a mathematical function or rheological model can be defined. A distinction can be made between Newtonian fluids and non-Newtonian fluids. For Newtonian fluids, this model is defined as follows:

$$\tau = \mu\dot{\gamma} \tag{2.1}$$

Where τ = shear stress, μ = dynamic viscosity and $\dot{\gamma}$ = shear rate. This Newtonian model has a constant μ and thus its rheogram is simply a sloped straight line from through the origin.

Non-Newtonian rheological models however often have three physical differences. These are: yield stress, nonlinearity between stress and shear, and time-dependency.

2.1.1.1. Yield stress

If a material has a yield stress it means it will not flow or irreversibly deform as long as the stress remains under a certain threshold. For stresses that stay below the yield-stress, the material behaves elastically. This also means it will recover all applied strain once the stress is removed (Van De Ree, 2015), (Boger, Scales, & Sofra, 2006). For forces that exceed the yield stress, the fluid will show viscous behaviour.

The Bingham rheological model is the model that fits this behaviour. It has a yield stress followed by a linear relation between shear rate and viscosity. This relation is given following Equation (2.2). The elastic behaviour can be described following Equation (2.3).

$$\tau = \tau_y + \mu_B\dot{\gamma} \tag{2.2}$$

$$\tau < \tau_y \rightarrow \ \dot{\gamma} = 0 \tag{2.3}$$

Where τ_y = the yield stress, and μ_B = the plastic viscosity.

2.1.1.2. Nonlinearity

Another way a non-Newtonian material can deviate from Newtonian behaviour is if a material shows a nonlinear relation between shear rate and stress. A model often used to describe this relation is the Herschel Bulkley model. This is presented as follows:

$$\tau = \tau_y + K\dot{\gamma}^n \tag{2.4}$$

K is often referred to as the consistency index and n is the flow index. Different kinds of behaviour may occur depending on the value for n. If $0 \leq n < 1$, the fluid behaves in a pseudoplastic or shear thinning manner. This means the fluid gets thinner (viscosity decreases) as the shear rate increases. Alternatively, if $n > 1$, the fluid behaves in a dilatant or thickening manner (viscosity increases) as the shear rate increases.

This nonlinear behaviour can also occur in materials that don’t have a yield stress. We then end up with the Ostwald-De Waele power law model, described mathematically as follows:

$$\tau = \mu\dot{\gamma}^{n} \tag{2.5}$$

Then, once again, depending on the value for n, different kinds of behaviour (pseudoplastic or dilatant) may occur.

2.1.1.3. Time-dependency

Then lastly, there’s also the class of materials that show time-dependent behaviour. An example could be that under a constant shear rate, the viscosity changes over time. An increasing viscosity under shear is called thixotropic behaviour, whereas a decreasing viscosity under shear is called rheopectic behaviour. This is also where the terms remoulded and unremoulded come into play. The remoulded state is relevant for flowing fluids, whereas unremoulded is relevant for depositions of the fluids in basins and reclamation areas.

2.1.2. Apparent viscosity

One should note that the plastic viscosity (Bingham), and consistency index (Herschel) are not the true viscosity. This viscosity is the tangent on the rheogram. However, for materials with a yield stress, or materials that show thinning or thickening behaviour, a different viscosity can also be defined. This is called the apparent viscosity and it is defined as the slope of a line from the origin to a certain shear stress on the flow curve in the rheogram. Equation (2.6) shows this relation.

$$\mu_a = \frac{\tau}{\dot{\gamma}} \tag{2.6}$$

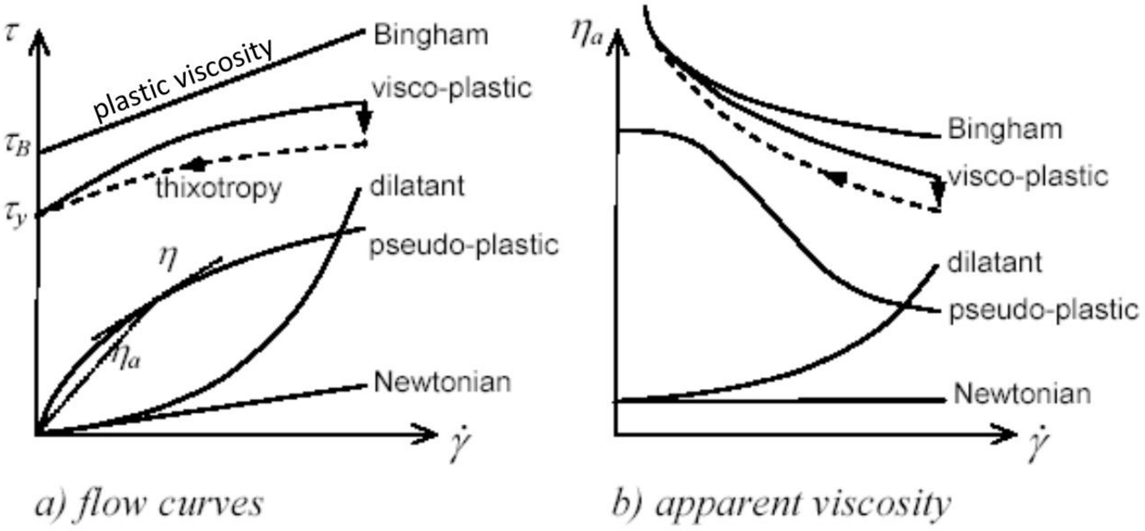

Figure 2.1: Rheograms for shear stress (a) and apparent viscosity (b) showing different fluid types (Newtonian and Bingham included). τ_y (and τ_B for Bingham) represents the yield stress, $\dot{\gamma}$ is the strain rate, and η_a is the apparent viscosity $\mu_a$as shown in Equation (2.6). Source: (Talmon, 2016)

2.1.3. Dynamic and kinematic viscosity

Besides a definition for viscosity, we can also define viscosity relative to the density. This is called the kinematic viscosity. In OpenFOAM, solvers and material models all calculate using a kinematic viscosity. So, all inputs that we define should be relative to the density. The kinematic viscosity is defined as follows:

$$\nu = \frac{\mu}{\rho} \tag{2.7}$$

2.2. Sand influence on viscosity and yield stress

Due to the presence of sand, the behaviour of the mixture changes in two ways: 1) it increases internal friction, and 2) it introduces non-cohesive particles (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016). The influence of the presence of the particles on both the plastic viscosity and yield stress are defined as follows:

$$\mu_p = \mu_{p,cf} e^{\beta\lambda} \tag{2.8}$$

$$\tau_y = \tau_{y,cf} e^{\beta\lambda} \tag{2.9}$$

Where $\mu_{p,cf}$ and $\tau_{y,cf}$ are rheological parameters of the carrier fluid alone, without particles (hence the *cf* subscript). β is a constant, which has been set at a value of 0.27 in (van Rhee, 2017) and (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016).

It should be noted that we define a slightly different nomenclature for the parameters $\mu_{p,cf}$ and $\tau_{y,cf}$ than in (van Rhee, 2017). The definition of the parameters is not different, though. $\mu_{p,c}$ and $\tau_{y,p}$ in (van Rhee, 2017) is the same as $\mu_{p,cf}$ and $\tau_{y,cf}$ in this report, respectively.

Further, the linear concentration λ is defined:

$$\lambda = \frac{1}{\left(\frac{c_{max}}{c_s}\right)^{\frac{1}{3}} - 1} \tag{2.10}$$

In which c_{max} is a maximum sand concentration. In (van Rhee, 2017) and (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016) set to 0.6.

2.3. Sand particles settling

As sand particles have a positive submerged weight in water, we expect them to settle towards the bottom. We should therefore review the theory on the particle settling velocity. This is relevant here because as particles settle, they create a non-uniform (non-homogeneous) field of the particle density. In turn this will also lead to non-uniform rheological parameters of the fluid mixture. We need that to be able to compute the viscosity as was shown in section 2.2.

2.3.1. Newtonian carrier fluid

In Newtonian fluids, the Stokes formula describes the forces on submerged particles. It is derived from the force balance on a particle between the (submerged) weight and the upward force the fluid exerts on the particle. Equation (2.11) shows this relation.

$$\frac{1}{6}\pi d^3 \left(\rho_s - \rho_{cf}\right) g = \frac{1}{8} C_D \pi d^2 w_0^2 \rho_{cf} \tag{2.11}$$

The left hand side represents the force due to gravity on the submerged weight. And the right hand side represents the drag force on a sphere moving through a fluid. d is the diameter of the particle, ρ_s and ρ_{cf} are the densities of the settling particles and fluid respectively, g is the gravitational acceleration, C_D is a drag coefficient, and w_0 is the settling velocity. If we rewrite for w_0 we arrive at equation (2.12).

$$w_0 = \sqrt{\frac{4}{3} \frac{\left(\rho_s - \rho_{cf}\right)}{\rho_{cf}} \frac{gd}{C_D}} \tag{2.12}$$

The drag coefficient depends on the regime of the flow and is a function of the particle Reynolds number:

$$Re_p = \frac{\rho_{cf} w_0 d}{\mu_{cf}} \tag{2.13}$$

Where μ_{cf} is the viscosity of the surrounding (carrier) fluid.

Now, for the laminar flow regime, the relation between the drag coefficient C_D and particle Reynolds number Re_p is defined as follows:

$$C_D = \frac{24}{Re_p} \tag{2.14}$$

Both equation (2.13) and equation (2.14) can be substituted into equation (2.12). This will yield equation (2.15) which gives a definition for the resulting settling velocity on the particle, $w_{s,0}$.

$$w_{s,0} = \frac{1}{18}\frac{(\rho_s - \rho_{cf})gd^2}{\mu_{cf}} \tag{2.15}$$

2.3.2. Hindered settling

The derivation shown in section 2.3.1 only holds for a low concentration of particles submerged in a fluid. To be more exact, the derived equations only hold for a single particle. Then, one might ask, what particles in higher concentrations? When many particles settle in the same area, those will hinder each other. The physical processes at play here have already been listed by (Winterwerp & van Kesteren, 2004):

1. Return flow and wake formation. A settling particle generates a return flow and wake. Particles in the wake will be subject to increased settling velocities. Particles caught in the return flow will see a lower settling velocity.
2. Dynamic or flow effects causing velocity gradients
3. Particle-particle collisions causing additional stresses
4. Particle-particle interaction through electrical charge
5. Einstein effect causing an increase in apparent viscosity due to an increased strain rate
6. Buoyancy effect due to increased density of the mixture
7. Cloud formation. Particles in the wake of another particle will settle faster and catch up, thus forming a (settling) cloud of particles.

According to (Van De Ree, 2015) the main contributions for hindered settling come from 1, 5, 6 for Newtonian fluids. According to (Talmon, van Kesteren, Sittoni, & Hedblom, 2013), for non-Newtonian fluids or mixtures, processes 2 and 5 are important (also mentioned by (Van De Ree, 2015)).

(Dankers & Winterwerp, 2007) provides formulations for the buoyancy effect and return flow effect factors in terms of the particle concentration c_s, shown by equations (2.16) and (2.17) respectively. (van Es, 2017) also refers to the same formulations.

$$(1 - c_s) \tag{2.16}$$

$$\left(1 - \frac{c_s}{c_{s,max}}\right)^2 \tag{2.17}$$

The hindered settling effects for buoyancy and return flow can be applied to the unhindered settling velocity $w_{s,0}$ (equation (2.15)). This gives equation (2.18).

$$w_s = (1 - c_s)\left(1 - \frac{c_s}{c_{s,max}}\right)^2 w_{s,0} \tag{2.18}$$

2.3.3. Non-Newtonian carrier fluid

In a static (not flowing) non-Newtonian fluid that has a yield stress, particles will not settle as long as the gravitational force (corrected for buoyancy) is lower than the force induced by the yield stress. The force balance on a single particle is defined as follows:

$$a_{form}\frac{1}{6}\pi(\rho_s - \rho_{cf})gd^3 > \tau_y\beta_{form}\pi d^2 \tag{2.19}$$

Here, the left hand side of the equation symbolizes the force on the submerged particle and the right hand side symbolizes the upward force caused by the yield stress. a_{form} and β_{form} are parameters that describe the shape of a particle. According to (Chhabra, 2007) for static situations (no flow, thus also no shear), this reduces to the following criterion for settling:

$$\tau_y \leq a_{cr}(\rho_s - \rho_{cf})gd \tag{2.20}$$

Where a_{cr} is an empirical parameter ranging from 0.048 to 0.2 (van Es, 2017) and (Chhabra, 2007).

However, in a non-static situation (flowing and shearing fluid), the force balance is different. The particles are co-rotating with the shearing of the fluid. (Talmon & Huisman, 2005) goes into the details of this co-rotation and presents equation (2.21):

$$w_{s,0} = a\frac{1}{18}\frac{(\rho_s - \rho_{cf})gd^2}{\mu_{cf}} \tag{2.21}$$

Where a is an empirical parameter. (Winterwerp & van Kesteren, 2004) has stated that for spherical particles, $a = 1$. Equation (2.21) looks to be quite similar to equation (2.15) except for this empirical parameter a.

Equation (2.21) can easily be substituted into equation (2.18) following (van Es, 2017). This yields equation (2.22).

$$w_s = (1 - c_s)\left(1 - \frac{c_s}{c_{s,max}}\right)^2 a\frac{1}{18}\frac{(\rho_s - \rho_{cf})gd^2}{\mu_{cf}} \tag{2.22}$$

This once again only holds if the shear stress is bigger than the yield stress. Otherwise, shear settling will not occur. This is summarized in this requirement:

$$If\ \tau < \tau_y, then\ w_s = 0 \tag{2.23}$$

2.4. Open channel flow

Tailings deposits flowing over a beach are often modelled as open channel flows. It’s main characteristics: gravity-based driving force on an angled channel. Further, these channel flows have a free water surface as opposed to closed channel or pipe flow.

2.4.1. Froude number

The Froude number can be used to quantify whether the flow is sub-critical or super-critical. The is a dimensionless parameter and for open channel flows this number is commonly calculated (Spelay, 2007). The definition of the Froude number shown in equation (2.24).

$$Fr = \frac{u}{\sqrt{gL}} \tag{2.24}$$

Where Fr is the Froude number, u is a characteristic flow velocity like the average velocity in a channel. L is a characteristic length scale like flowdepth or hydraulic radius.

The flow is considered sub-critical if the Froude number is lower than 1. If it is higher than 1, it is considered super-critical, if it is equal to 1 it is critical. A hydraulic jump might occur whenever a flow switches over from sub- to super-critical.

It should be noted that (2.24) is the Froude number for Newtonian fluids. The application of the Froude number for non-Newtonian fluids remains uncertain (Spelay, 2007).

2.4.2. Reynolds number

The flow regime can be turbulent or laminar depending on the Reynolds number. If a flow is laminar, the streamlines in the flow are parallel to each other. For Newtonian fluids, the Reynolds number is defined as follows:

$$Re = \frac{\rho u L}{\mu} \tag{2.25}$$

Where ρ is the density, u is the flow velocity, L is a characteristic length and μ is the kinematic viscosity.

For non-Newtonian fluids, the Reynolds number is more ambiguous (Haldenwang, Slatter, & Chhabra, 2010), (Van De Ree, 2015). Most definitions in literature are based on the friction factor (Van De Ree, 2015). The dimensionless shear stress is expressed using the fanning friction factor:

$$f = \frac{\tau_b}{\frac{1}{2}\rho v^2} \tag{2.26}$$

For Newtonian flow, in the laminar flow regime, the friction factor-Reynolds number is defined as follows:

$$f = \frac{16}{Re} \tag{2.27}$$

Then, equations (2.26) and (2.27) can be combined; this gives us the definition for the non-Newtonian Reynolds number Re_{nN}:

$$Re_{nN} = \frac{8\rho v^2}{\tau_b} \tag{2.28}$$

Where τ_b is the bed or bottom shear stress.

2.4.3. Fully developed, steady, uniform flow

The open-channel flow from this point forward is considered fully developed, steady and uniform. Realistically, the characteristics of non-Newtonian tailings lead to laminar flow behaviour (Van De Ree, 2015). When looking at tailings beach flows, there are end-effects in the transition from channel to sheet flow. Basically, we are (only) looking at a flow down an inclined plane. The theory in this section only looks at the fully developed, steady and uniform flow behaviour. Furthermore, an infinite width allows us to disregard edge effects.

2.4.3.1. Bingham Plastic

Bingham Plastic fluids have a yield stress. We know that the calculation of the shear stress in a Bingham Plastic is calculated according to equation (2.2). Further, we know that for a shear stress

$\tau < \tau_y$, the fluid will not shear. The velocity profile of a non-Newtonian fluid with non-zero yield stress will show plug flow behaviour near the free surface. This plug represents an unsheared layer. This section will show the associated formulas.

The shear stress distribution of a fluid in an open-channel flow is defined as follows:

$$\tau = \rho g y sin\theta \tag{2.29}$$

Where θ is the slope of the channel, ρ is the density of the fluid, and y is the coordinate along the flowdepth.

This leads to the calculation of the bed shear stress τ_b by substituting flowdepth h_0 for y following equation (2.30).

$$\tau_b = \rho g h_0 sin\theta \tag{2.30}$$

The plug height h_p is constructed following from $\tau = \tau_y$ at $y = h_p$:

$$\tau_y = \rho g h_p \sin\theta \tag{2.31}$$

This can be reduced by combining equations (2.30) and (2.31), effectively creating a function of channel depth, yield stress and bed shear stress:

$$h_p = \frac{\tau_y}{\tau_b} h_0 \tag{2.32}$$

The height of the plug shearing layer between the and the inclined plane (h_s) can be defined as:

$$h_s = h_0 - h_p = h_0 - \frac{\tau_y}{\tau_b} h \tag{2.33}$$

2.4.3.2. Velocity profile

An interesting flow feature to look at for flow down an inclined plane is the velocity profile along the flowdepth. An analytical solution for the velocities at different depths has been derived by (De Kee, Chhabra, Powley, & Roy, 1990). This solution holds for a flow with a free water surface and a no-slip condition on the bottom.

A more simplified version of the same solution has been given by (Haldenwang, Kotzé, & Chhabra, 2012).

Adjusting the nomenclature to align with earlier mentioned variables, we end up with equation (2.34) for the velocity $V_{x,shear}$ in the shearing layer ($h_p \leq y \leq h_0$):

$$V_{x,shear} = \left(\frac{n}{n+1}\right)\left(\frac{K}{\rho g sin\theta}\right)\left(\frac{\tau_b}{K}\right)^{\frac{n+1}{n}}\left(1 - \frac{\tau_y}{\tau_b}\right)^{\frac{n+1}{n}}\left(1 - \left(\frac{\frac{\tau}{\tau_y} - 1}{\frac{\tau_b}{\tau_y} - 1}\right)^{\frac{n+1}{n}}\right) \tag{2.34}$$

And the velocity $V_{x,plug}$ of the plug ($0 \leq y \leq h_p$):

$$V_{x,plug} = \frac{nK}{(n+1)\rho g sin\theta}\left(\frac{\tau_b}{K}\right)^{\frac{n+1}{n}}\left(1-\frac{\tau_y}{\tau_b}\right)^{\frac{n+1}{n}} \tag{2.35}$$

It should be noted that equations (2.34) and (2.35) include cases for power-law fluids ($\tau_y = 0, 0 \leq n \leq 1 \; or \; n > 1$), Bingham Plastic fluids ($n = 1, K = \mu_B, \tau_y \neq 0$), and Newtonian fluids ($n = 1, \tau_y = 0$).

The equation for the average velocity $V_{x,avg}$ has been given by (Haldenwang, Kotzé, & Chhabra, 2012):

$$V_{x,avg} = \frac{nK}{(2n+1)\rho g sin\theta}\left(\frac{\tau_b}{K}\right)^{\frac{n+1}{n}}\left(1-\frac{\tau_y}{\tau_b}\right)^{\frac{n+1}{n}}\left(1+\left(\frac{n}{n+1}\right)\frac{\tau_y}{\tau_b}\right) \tag{2.36}$$

And a more simplified version by (Van De Ree, 2015) applicable to Bingham Plastic fluids specifically, provided the flowdepth h_0 is known:

$$V_{x,avg} = \left(\frac{1}{3}\tau_b - \frac{1}{2}\tau_y + \frac{1}{6}\frac{\tau_y^2}{\tau_b^2}\right)\frac{h_0}{\mu_B} \tag{2.37}$$

3. Survey on previous research

This chapter will give an overview of relevant previous research in the field of tailings and slurry flow. Experimental research has been done with regards to velocity profile development and concentration profiles of sand-mud mixtures. More recent research focusses on numerical modelling and simulation of these flows.

Reviewing this previous research will allow for comparison of the performed simulations in OpenFOAM. In light of that, the results found by others could serve as a basis for validation.

3.1. Spelay (2007)

(Spelay, 2007) performed experiments with sand-mud mixtures in a half open pipe. The pipe had a diameter $D = 156.7$mm. The mixtures used were four different Bingham Plastics with properties quite typical for the mine tailing or oil sands industry. With the experimental setup used, the flow rate and flume angle were varied. At 14.8m from the flume inlet a traversing gamma ray densitometer was placed to measure the sand concentration in the mixture at that location.

Further, the flume was fitted with two depth gauges located 7.5 and 13.3 m from the flume inlet. These were used to measure the flowdepth of the slurry. The depth gauges are described as “height measurement” points 1 and 2 in Figure 3.1.

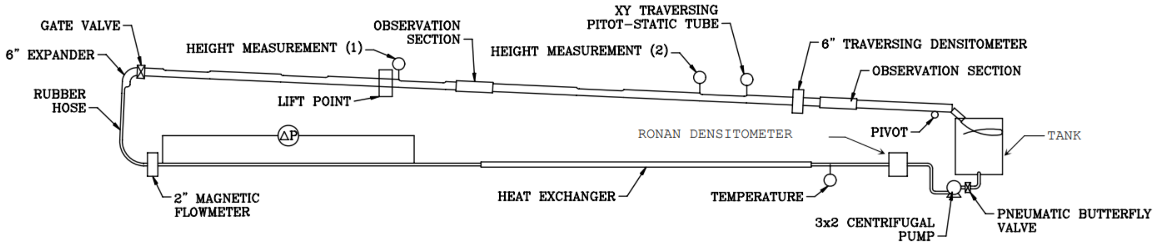

Figure 3.1: Saskatchewan Research Council’s 156.7 mm flume circuit used in the experimental program (Spelay, 2007)

One class of mixtures used in the experiment was identified as thickened tailings slurries. This is a sand, clay & water mixture and the composition was 15.4:15.1:69.5 (sand:clay:water v/v). This resulted in a bulk density ρ_{mix} = 1510 kg/m^3, calculated according to equation (3.1). Due to the high clay fraction, these slurries had quite high yield stress between 30 and 50 Pa. Due to this high yield stress, the thickened tailings mixture flowed in the laminar regime. The carrier fluid alone, without the sand particles added, had a yield stress τ_y = 47.3 Pa and plastic viscosity μ_p = 0.0214 Pa.s and density ρ_{cf} = 1303 kg/m^3. The density of the sand particles was ρ_s = 2650 kg/m^3.

$$\rho_{mix} = 0.154\, \rho_s + (1 - 0.154)\rho_{cf} \tag{3.1}$$

(Spelay, 2007) performed multiple measurements regarding the flow in the flume circuit. These measurements included a concentration profile measurement and velocity profile measurements. Section 3.1.1 and 3.1.2 present these.

3.1.1. Concentration profile measurements

Concentration profile measurements have been obtained. For the thickened tailings slurry mentioned in section 3.1, Table 3.1 lists the measurement data on the sand and solids concentration.

y/D	C_{solids} **(v/v)**				C_{sand} **(v/v)**			
0.95	-	-	-	-	-	**-**	-	-
0.90	-	-	-	-	-	**-**	-	-
0.85	-	-	-	-	-	**-**	-	-
0.80	-	-	-	-	-	**-**	-	-
0.75	-	-	-	-	-	**-**	-	-
0.70	-	-	-	-	-	**-**	-	-
0.65	-	-	-	0.247	-	**-**	-	0.083
0.60	-	-	0.277	0.263	-	**-**	0.120	0.104
0.55	-	0.272	0.280	0.262	-	***0.114***	0.124	0.101
0.50	0.272	0.270	0.280	0.261	0.114	***0.112***	0.124	0.101
0.45	0.275	0.267	0.287	0.253	0.117	***0.108***	0.132	0.091
0.40	0.276	0.273	0.287	0.252	0.119	***0.115***	0.132	0.089
0.35	0.273	0.276	0.285	0.256	0.115	***0.118***	0.129	0.094
0.30	0.270	0.263	0.281	0.250	0.111	***0.103***	0.125	0.087
0.25	0.272	0.267	0.281	0.253	0.114	***0.108***	0.125	0.091
0.20	0.258	0.252	0.271	0.252	0.097	***0.089***	0.112	0.090
0.15	0.256	0.255	0.267	0.255	0.095	***0.093***	0.108	0.093
0.10	0.277	0.296	0.316	0.284	0.120	***0.143***	0.168	0.129
0.05	0.332	0.360	0.400	0.333	0.187	***0.221***	0.270	0.189
h **(m)**	0.0861	0.0968	0.1039	0.1077	0.0861	***0.0968***	0.1039	0.1077
θ **(°)**	5.4	4.5	4	4.5	5.4	***4.5***	4	4.5
Q **(L/s)**	5	5	5	2.5	5	***5***	5	2.5

Table 3.1: Solids and sand concentration (C_{solids} and C_{sand} respectively) profile measurements for a model thickened tailings slurry in the 156.7 mm flume; ρ_{mix} = 1510 kg/m^3. h represents the flowdepth at the measurement point, θ the inclination of the flume and Q the inlet flow rate, Table D.24 in (Spelay, 2007).

The values for C_{sand} in Table 3.1 that are presented in bold and italics have been plotted against the non-dimensional y/D in Figure 3.2. This shows a profile of the sand concentration along the depth of the flow. It can be seen that near the bottom, close to the pipe wall, a bed with a higher sand concentration forms. Just above that, there is a zone with a dip in the sand concentration.

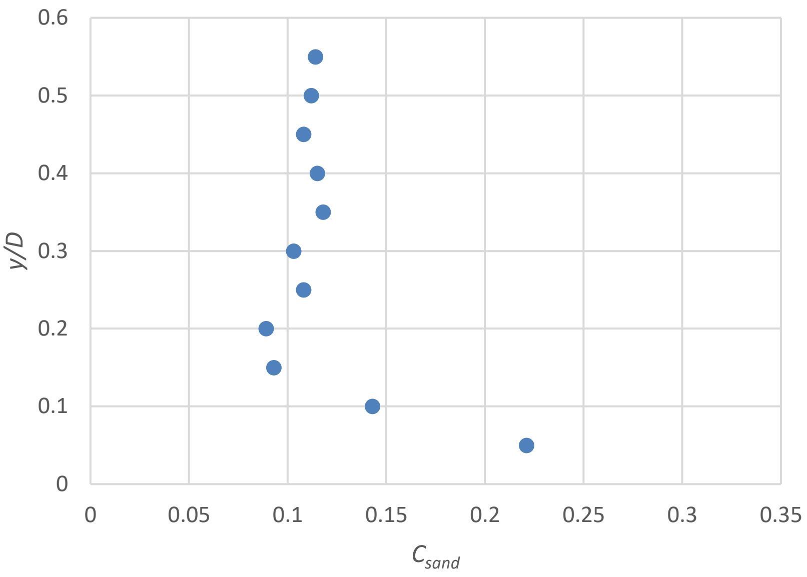

Figure 3.2: Sand concentration measurement profile C_{sand} against non-dimensional y/D for θ = 4.5° and Q = 5 L/s

3.1.2. Velocity profile measurements

(Spelay, 2007) also performed velocity profile measurements. Local velocities were measured at different points in the flume using a XY traversing pitot-static tube. Figure 3.3 shows the positions of these measurement points. Table 3.2 shows both the exactly (non-dimensional) locations of the measurement points as well as the results of these velocity measurements for the thickened tailings slurry mentioned in in section 3.1.

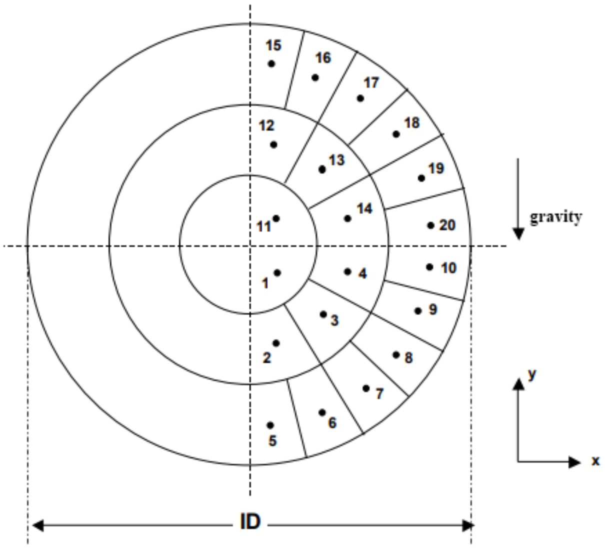

Figure 3.3: Flume two-dimensional mixture velocity profile measurement positions in the 156.7mm flume (corresponding to Table 3.2 in this report and Table D.32 in (Spelay, 2007). Gravity works in negative y-direction. Source: Figure D.1 in (Spelay, 2007).

Point	x/R	y/R	v/V	v/V	v/V	v/V
1	0.11	0.89	1.75	2.37	2.37	2.41
2	0.12	0.54	1.80	2.29	2.24	2.94
3	0.34	0.66	1.71	2.35	2.19	2.45

4	0.46	0.88	1.72	2.41	2.27	2.96
5	0.11	0.19	0.82	1.70	1.40	1.96
6	0.31	0.25	0.89	1.76	1.53	2.27
7	0.50	0.35	1.00	1.71	1.67	2.04
8	0.65	0.50	0.98	1.84	1.77	1.51
9	0.75	0.69	1.17	1.78	1.80	0.59
10	0.81	0.89	1.11	1.87	1.85	1.21
11	0.11	1.11	--	2.33	2.38	2.95
12	--	--	--	--	--	--
13	0.34	1.34	--	--	--	2.11
14	0.46	1.12	--	2.30	2.37	3.09
15	--	--	--	--	--	--
16	--	--	--	--	--	--
17	--	--	--	--	--	--
18	--	--	--	--	--	--
19	0.75	1.31	--	--	1.87	1.54
20	0.81	1.11	--	2.02	1.92	2.39
Q (L/s)			5	5	5	2.5
θ (°)			5.4	4.5	4	4.5
h (m)			0.0873	0.0970	0.1048	0.1077
V (m/s)			0.46	0.40	0.36	0.18

Table 3.2: Mixture velocity profile measurements for a model Thickened Tailings slurry in the 156.7mm flume; ρ_{mix} = 1510 kg/m^3. Table D.32 in (Spelay, 2007)

3.1.3. Frictional loss measurements

Frictional loss measurements were also performed. This data also includes measurements for flowdepths h_1 and h_2, at different locations in the flume. These were performed using Vernier caliper gages, located 7.5m (location 1) and 13.3m (location 2) from the flume inlet. These measurements can later be used to compare to.

Q (m^3/s)	**T (°C)**	**θ (°)**	**h_1 (m)**	**h_2 (m)**	**V (m/s)**	**τ_w (Pa)**	**Re_{Zhang}**	**f**
0.00497	21.6	4.00	0.0909	0.0993	0.41	41.0	55	0.3275
0.00498	22.4	4.50	0.0849	0.0931	0.44	44.7	48	0.3027
0.00510	26.6	5.41	0.0805	0.0834	0.50	54.6	59	0.2892
0.00253	25.0	4.50	0.0998	0.1054	0.19	49.7	8	1.8379

Table 3.3: Frictional loss measurements for a model Thickened Tailings slurry in the 156.7mm flume; ρ_{mix} = 1510 kg/m^3. Table D.17 in (Spelay, 2007)

3.1.4. Different inlet

(Spelay, 2007) tested with two different inlet conditions. These were only tested in the sand-water tests and not in the tests with the thickened tailings. With the original inlet condition, the mixture was transferred from the feed line to the flume through a flexible rubber hose which was orientated parallel (in-line) to the flume. With this setup there is a serious risk of blocking the channel system due to sand being deposited at the inlet of the flume.

A new inlet condition was created so that the slurry was transferred from the feed line to the flume through a flexible rubber hose which was orientated perpendicular to the flume. The purpose of these modifications was to prevent this sand deposition close to inlet. However, these experiments were not performed using the thickened tailings.

3.2. Hansen (2016)

In the work by (Hanssen, 2016), non-Newtonian settling slurries have been implemented in Delft3D, a numerical modelling package developed by Deltares. This work goes into the very specifics of the 1DV numerical model Delft3D-Slurry.

Three rheological models were used that were based on the shear stress τ as follows:

$$\tau = \tau_y + \mu\dot{\gamma}^n \tag{3.2}$$

The different models differ in the formulation of yield stress τ_y, viscosity μ and flow index n. The definitions of these three models were following Winterwerp & Kranenburg, Jacobs & Van Kesteren, and Thomas.

On first look, these definitions don't match earlier theory encountered in section 2.2. However, upon further inspection, also in (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016), it is found that the models are in fact matching with the earlier theory. The difference comes from the definition of the yield stress and viscosity of the carrier fluid. In equation (2.8) and (2.9) this is encapsulated into 1 parameter for both, μ_p and τ_y.

(Hanssen, 2016) and (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016) show the more fundamental origin of these parameters. They include for instance: water content W, plasticity index PI, and empirical parameters K_y, B_y, K_μ, B_μ, β. For completeness, the rheological models are shown below.

Equations (3.3) and (3.4) show the first model following Winterwerp and Kranenburg. It should be noted that for this model, the flow index n: $0 < n \leq 1$.

$$\tau_y = A_y \left(\frac{\phi_{cl}}{1-\phi_{sasi}}\right)^{\frac{2}{3-n_f}} e^{\beta\lambda}, \tag{3.3}$$

$$\mu = \left[\mu_w + A_\mu \left(\frac{\phi_{cl}}{1-\phi_{sasi}}\right)^{\frac{2(a+1)}{3}} \left[\frac{1}{\dot{\gamma}}\right]^{\frac{(a+1)(3-n_f)}{3}}\right] e^{\beta\lambda} \tag{3.4}$$

Equations (3.5) and (3.6) show the second model following (Jacobs, Le hir, Van Kesteren, & Cann, 2011). This follows a Bingham model, so flow index n = 1.

$$\tau_y = K_y \left(\frac{W}{PI}\right)^{B_y} e^{\beta\lambda} \tag{3.5}$$

$$\mu = \mu_w \left[1 + K_\mu \left(\frac{W}{PI}\right)^{B_\mu}\right] e^{\beta\lambda} \tag{3.6}$$

Equation (3.7) and (3.8) show the third model following (Thomas, 1999). This also follows a Bingham model, so flow index n = 1.

$$\tau_y = C_y \left(\frac{\phi_{fines}}{\phi_{water}+\phi_{fines}}\right)^p \left[1 - \frac{\phi_{sa}}{k_{yield}\phi_{sa,max}}\right]^{-2.5} \tag{3.7}$$

$$\mu = e^{D\frac{\phi_{fines}}{\phi_{water}}}\left[1-\frac{\phi_{sa}}{K_{visc}\phi_{sa,max}}\right]^{-2.5} \tag{3.8}$$

It can be noted that the first 2 models (equations (3.3), (3.4), (3.5), and (3.6)) contain the term $e^{\beta\lambda}$, in which β is an empirical value set to 0.27. The linear concentration λ is defined as in equation (2.10).

The other terms in these equations can be collapsed into 2 carrier fluid parameters $\mu_{p,cf}$ and $\tau_{y,cf}$, as has been done by (van Rhee, 2017). This can be useful if the carrier fluid parameters are known.

(Hanssen, 2016) concluded Delft3D's 1DV model is capable of capturing the rheological and sand settling processes for laminar slurry mixtures. They fall in line with the theoretical predictions. Further, plug-flow behaviour and sand bed formation was reproduced. Further, flow velocity was reversely proportional to sand bed concentration, also as expected.

3.2.1. Regularization of the flow curve

For materials that have a yield stress τ_y calculating the apparent viscosity is a challenge at low shear rates. The apparent viscosity is very high for low shear rates $\dot{\gamma}$. Following equation (2.6), the apparent viscosity will become impossible for a shear rate equal to 0. This occurs if shear stresses are smaller than the yield stress. This happens in the plug zone of the flow.

(Hanssen, 2016) has made a modification has been made following (Papanastasiou, 1987). The proposed model for this is an exponential regularization like so:

$$\tau = \tau_y(1-e^{-m\dot{\gamma}}) + \mu\dot{\gamma} \tag{3.9}$$

The same formulation was also found in (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016), and it is stated that equation (3.9) creates a finite viscosity at low shear rates. The constant m should be chosen in such a way that the numerical solution approaches the analytical solution. In (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016) it is stated that this holds for high values of m. A value of m = 5000 has been chosen.

For a sensitivity analysis on this parameter I turn to (Hanssen, 2016). It is shown that parameter m has significant influence on the steepness of the flow curve for low shear rates as well as the velocity profile of flowing fluids with a yield stress. From a comparison with the analytical solution it is shown that m = 5000 is required to minimize numerical mutation.

3.3. Van Rhee (2017)

(van Rhee, 2017) investigated OpenFOAM's capability to simulate non-Newtonian fluids and coarse solid mixture flow. An adaptation of the already existing `icoFoam` application has been created. Non-Newtonian models were already available in `icoFoam`, but the influence of coarse particles on the rheology hadn't been implemented before.

The influence of the particles on the plastic viscosity and yield stress has been implemented following to (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016). Equations (2.8), (2.9) and (2.10) were followed. It should be noted here that the source code showed regularization as shown in equation (3.9) has been incorporated too. The simulation case files showed that the regularization parameter m has been set to 50. The source code can be seen in Appendix A.

Further, the settling velocity of the particles have also been implemented following (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016) which can be seen in equation (2.15).

Lastly, a transport equation for the sand fraction has been implemented using the drift flux approach. This will ensure that the particles can 1) settle with reference to the carrier fluid and 2) get dragged along with the carrier fluid as the mixture flows through the domain. Equations (3.10) and (3.11) show this for a given control volume:

$$\frac{\partial c_s}{\partial t} + \nabla . (U_s c_s) = 0 \tag{3.10}$$

$$U_s = U + w_s \tag{3.11}$$

Here, c_s is the sand fraction, U_s is the sand particle velocity, w_s is the settling velocity and ∇ is the divergence operator.

In the work by (van Rhee, 2017), only the buoyancy effect (equation (2.16)) has been implemented following equation (3.12).

$$w_s = (1 - c_s) w_{s,0} \tag{3.12}$$

Where $w_{s,0}$ is the unhindered settling velocity, w_s is the hindered settling velocity and $(1 - c_s)$ represents the hindered settling effect in due to buoyancy.

The source code files accompanying (van Rhee, 2017) also showed traces of code following equation (2.22), having both the buoyancy and return flow effects implemented. However, these lines of code had been commented out, so I don't believe this has actually been used. Appendix A.2 shows this code being commented out.

(van Rhee, 2017) concluded the implementation still showed quite okay agreement between numerical computations and experimental findings in (Spelay, 2007). One should note that in the paper, on equation 7, a power 2 operator on the particle diameter (d^2) is missing, though this has been implemented correctly.

(van Rhee, 2017) validated the implementation in 2 parts. Part 1 was performed using 2D open-channel (section 3.3.1), and part 2 was performed using a 3D open pipe (section 3.3.2).

3.3.1. 2D open-channel

The first part of the validation was a comparison against the analytic solution of a 2D open-channel flow with a fixed flowdepth. In a sense this could be considered a 2D closed-channel flow with a top lid that allows for full slip. It should be noted here that no sand particles are added to the simulation just yet.

A simulation using a Bingham Plastic viscosity model has been performed. The analytical solution was formulated using the pressure gradient taken from the numerical solution.

The following parameters were applied: flowdepth h_0 = 0.1 m, average velocity $\bar{u}$ = 0.4 m/s, yield stress τ_y = 10 Pa, plastic viscosity μ_p = 0.0214 Pa.s, channel length L = 20 m. The velocity in x direction U_x has been extracted at x = 15m from the inlet zone. Figure 3.4 shows these velocity as calculated by the simulation and the velocity following the analytical solution.

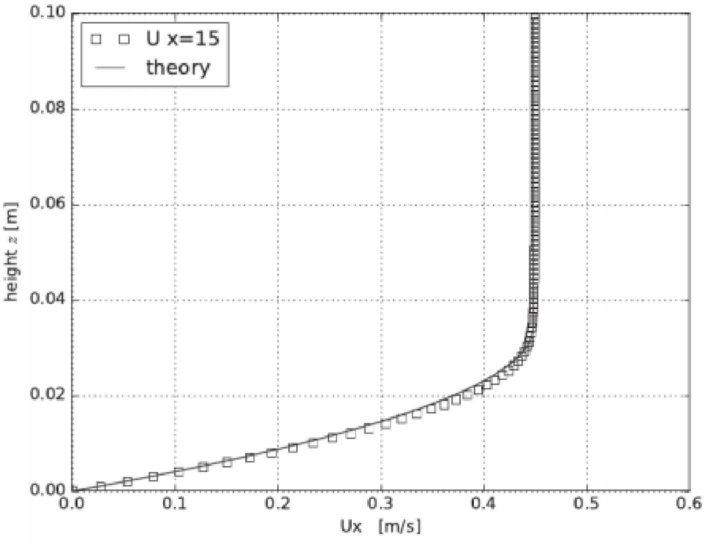

Figure 3.4: Velocity U_x for both the numerical and analytical solution for the 2D channel flow at x=15. Source: (van Rhee, 2017)

3.3.2. 3D open pipe

The second part of the validation was a comparison against experiments performed by (Spelay, 2007) in a 3D open pipe flow with a fixed flowdepth. In a sense this could be considered a 3D half round closed-pipe flow with a top lid that allows for full slip.

The flowdepth was chosen in such a way that it matched with the flowdepth found in the experiments from (Spelay, 2007). A simulation using a Bingham Plastic viscosity model has been performed. The sand concentration profile has been compared to the profile found in the experiments.

The following parameters were applied: flowdepth h_0 = 0.0968 m, inlet discharge Q = 5 L/s, yield stress of the carrier fluid $\tau_{y,cf}$ = 47.3 Pa, plastic viscosity of the carrier fluid $\mu_{p,cf}$ = 0.0214 Pa.s, inflow concentration of sand c_s = 0.12, sand particle diameter d = 188 micron, pipe length L = 15 m, pipe diameter D = 0.1567 m.

Figure 3.5 shows the extracted sand concentration c_s and flow velocity in the z-direction along the length of the pipe U_z at z = 15 m from the inlet zone. The sand concentration has been compared to the sand concentration as found by (Spelay, 2007). Figure 3.6 and Figure 3.7, respectively, show the sand concentration along the vertical symmetry plane after 300 seconds of simulation and at the xy-plane at z = 15 m.

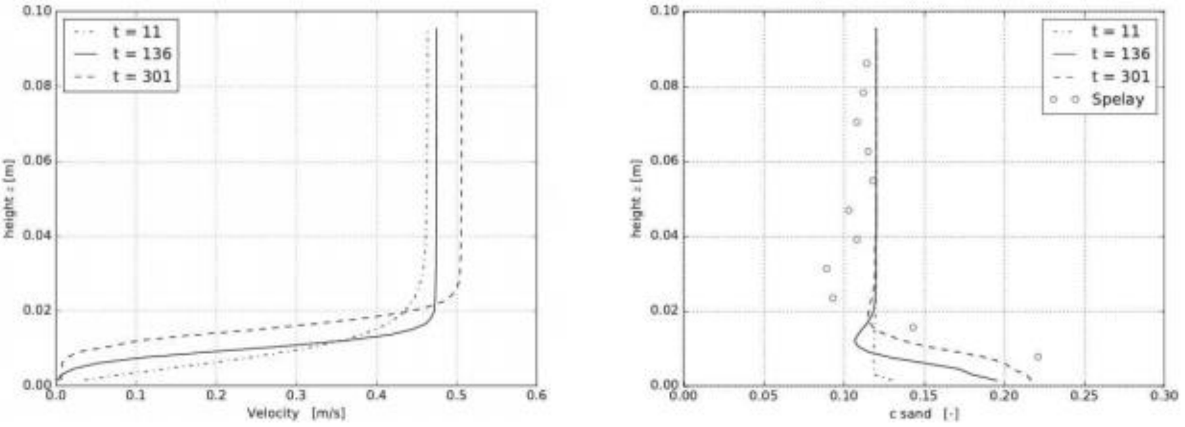

Figure 3.5: Velocity profile and sand concentration at z = 15 m at different times from the start of the simulation. Sand concentration has been compared to results found by (Spelay, 2007). Source: (van Rhee, 2017)

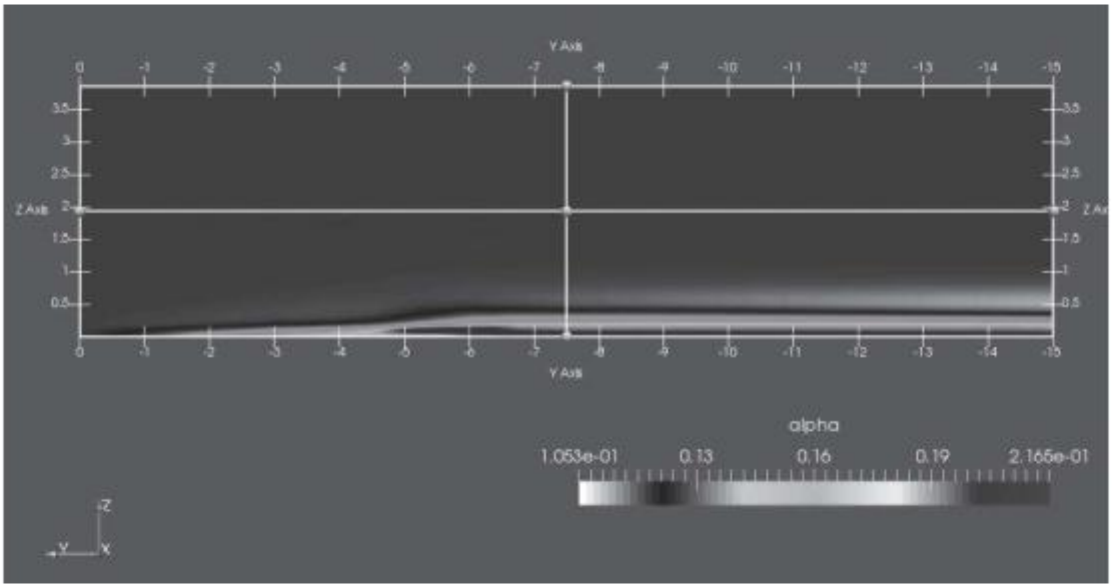

Figure 3.6: Sand concentration along vertical symmetry plane in 3D pipe. Source: (van Rhee, 2017)

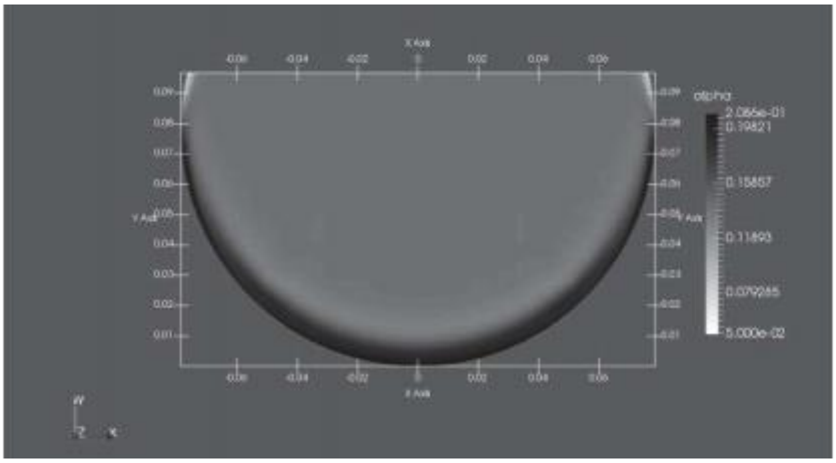

Figure 3.7: Sand concentration at z = 15 m from the inlet zone. Source: (van Rhee, 2017)

Sand settles at the bottom of the pipe and this forms a sand bed. (van Rhee, 2017) concludes the concentration profile of the sand is similar to the experiments, noting that the dip in sand concentration just above the bed in the shearing layer is found to be smaller in the simulations; the experiments showed a larger dip in sand concentration in the shearing layer. The final conclusion is that `icoFoam` is capable to capture solids settling processes. However, to truly mimic open pipe and open channel flows, a free mixture surface is required.

4. Summary

In Part 1 of this study, an overview of relevant theory and literature has been presented. We have seen different rheological models and implementation techniques in CFD packages OpenFOAM and Delft3D.

Different aspects have been compiled: the foundational modelling of rheological models (section 2.1), the influence (suspended) particles have (section 2.2), and particle settling behaviour (section 2.3). Further, specifically for open channel flow with a non-Newtonian fluid, an analytical solution for the velocity profile (section 2.4.3.2)and equations for the shear stress distribution (section 2.4.3.1) have been presented.

(Hanssen, 2016) has shown how for low strain rates the apparent viscosity can't be calculated. This poses a problem in the plug flow region when dealing with non-Newtonian fluids. A solution for this has been presented following (Papanastasiou, 1987) in section 3.2.1.

(Spelay, 2007) has performed experiments with a mixture of non-Newtonian fluid and solids in a half open pipe. Measurements for sand concentration profiles as well as velocity profiles are performed. (van Rhee, 2017) used these measurements for validation of the adaptation of `icoFoam`.

(van Rhee, 2017) has implemented capability to simulate non-Newtonian fluids with coarse solids mixture flow. This has done by adapting the `icoFoam` solver in OpenFOAM. Simulations have been performed with a Bingham Plastic fluid model excluding and including solids. The relevant theory for the influence of the solids on the viscosity and yield stress of the mixture have been presented in section 2.2 (van Rhee, 2017) concludes `icoFoam` is capable to capture solids settling processes. However, to truly mimic open pipe and open channel flows, a free mixture surface is required. The results themselves can be interesting for comparison for the work presented in this study.

Part 2: Implementation and simulations

Following up on the literature study that has been performed in Part 1 of this report, Part 2 zooms in on the practical side of things. The goal is to be able to run simulations in OpenFOAM using a non-Newtonian fluid with sand transport, segregation, and settling. To be able to do this, first an analysis of OpenFOAM is performed, then the adaptation is implemented, and finally simulations are ran and compared to previously presented results.

5. OpenFOAM & implementation

This chapter will give an introduction to OpenFOAM, the weapon of choice for this research. Further, this chapter will show what adjustments have been made to fulfil the requirements set by the research objective.

5.1. Intro to Computational Fluid Dynamics

For simple flow problems, explicit analytical solutions have been formulated. But for more complex problems, more complex geometries, more complex fluid behaviours, these don't exist. That's where computational fluid dynamics (CFD) comes into play.

CFD software is used for approximating Partial Differential Equations (PDEs) numerically in for example flow problems. These equations come from the Navier-Stokes equations, and the equations for mass and energy conservation. Space is discretized to small control volumes, and time is discretized into small timesteps. Using that principle the flow can be numerically approximated. For each of the control volumes the equations can be solved. Calculations are performed of the interaction of the fluid with the boundary conditions that are set.

5.2. Introduction to OpenFOAM

OpenFOAM is a library written in C++. It is a library build of many different components: applications, utilities and tools. For this research, OpenFOAM 5.0 (foundation variant) is used (Build: 5.x-68e8507efb72). FOAM is a shorthand for Field Operations and Manipulation. It can be used for flow problems, finite element problems and financial computations. Figure 5.1 graphically shows the structure of the components.

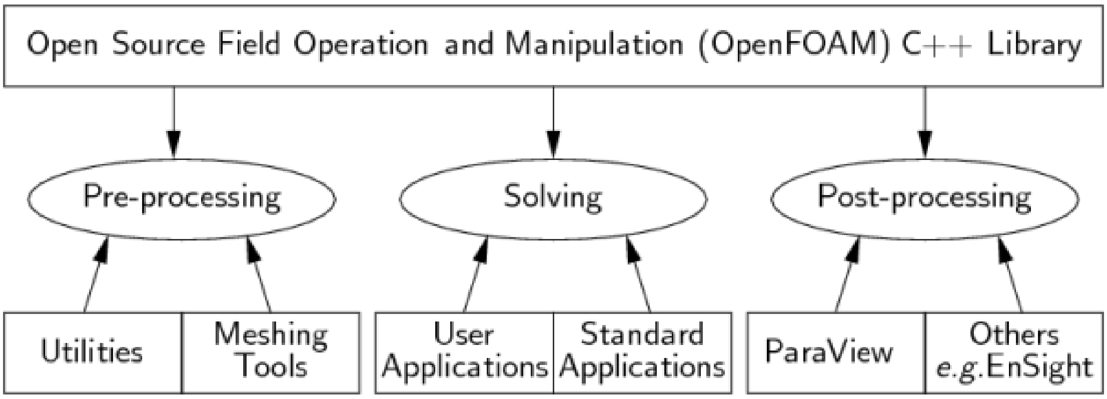

Figure 5.1: Overview of OpenFOAM components. Source: cfd.direct

Each simulation case in OpenFOAM gets its own folder in which a certain structure must be adhered to. 3 main folders are required: `0`, `constant`, and `system`. The `0` folder contains separate files for each of the values one is interested in, for example: pressure, velocity, volume fraction, etc. These have to be initialized with an initial value on the internal domain. Further, the boundary conditions have to prescribed as well. Constant values, both in space and time, are to be put in the `constant` folder. Examples are: the mesh geometry, viscosity, external force fields like gravity, and possibly turbulence model settings. The `system` folder contains all information pertaining to the discretization options, solver options like the time step and maximum courant number, differential schemes, residual control, and algorithm choice. This folder structure is shown graphically in Figure 5.2.

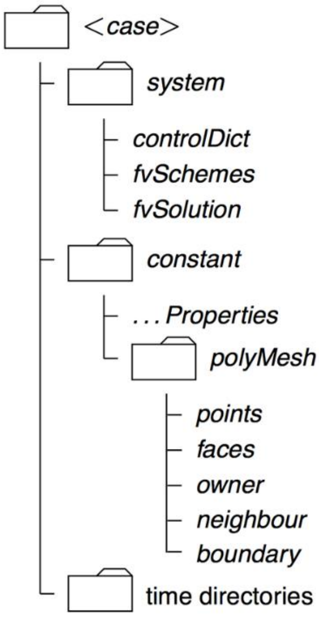

Figure 5.2: Case folder structure. Source: cfd.direct

5.3. Challenge with OpenFOAM

OpenFOAM doesn't have a graphical user interface like some other CFD software packages have. Also, some parts of OpenFOAM's code aren't as *well-documented* as one might hope. At best, the documentation is spread out over multiple places: coding guides, user manuals, in the source code, or online forums.

This is a challenge for when a simulation fails for some reason. If proper documentation is lacking, figuring out why something fails easily results in a trial and error process. With that I mean that simulation case files must be configured, the simulation must be ran, and then visual inspection can commence using postprocessing tools like ParaView. If anything fails during that process, it's back to square one.

5.4. Solver requirements

To satisfy the research objective, it's important to understand what existing solvers are capable of. OpenFOAM consists of a number readily-available solvers; but not all requirements are currently built into OpenFOAM's solvers. A starting point can be chosen in such a way that most requirements are already fulfilled, or such that development efforts are minimized. A capability matrix of existing solvers can be found on openfoam.com[1].

The requirements needed to simulate tailing slurries can be formulated as follows: gravitational forces have to be taken into account; a free mixture surface should be trackable; non-Newtonian viscosity models are to be used; particle segregation and transport model are to be supported, and the influence of particles on the viscosity should be taken into account.

Luckily, the source code as used by (van Rhee, 2017) has been made available to me and this should save time in the development phase. Also, to follow along the recommendations put forward by (van

[1] https://www.openfoam.com/documentation/guides/latest/doc/openfoam-guide-applications-solvers.html#sec-applications-solvers-capability-matrix

Rhee, 2017), OpenFOAM's `interFoam` seems the logical choice. It is a solver for 2 incompressible fluids using the VOF method. It will allow for the tracking of an interface, which is needed to allow for a true free mixture surface. According to OpenFOAM's capability matrix, `interFoam` is the only multi-phase solver. It should be noted that many solvers are still missing from this matrix. To take an example, `multiphaseEulerFoam` is missing. That solver also includes compressible fluids and heat transfer, both of which are unnecessary for our research.

As it stands, `interFoam` offers support for non-Newtonian models (e.g.: Power Law, Herschel-Bulkley), but none include the influence of sand particles, so this is a part that will need to be implemented. Further, it doesn't have additional transports equations for sand. To allow for sand settling, segregation, and transport, an additional transport equation will have to be added.

5.5. interFoam

`interFoam` is an incompressible, 2 phase solver in OpenFOAM with VOF method. Like the VOF method described in section 5.5.2, `interFoam` utilizes VOF averaging for properties like density, viscosity and velocity. So, only one momentum equation is solved.

5.5.1. Governing equations

The fundamental governing equations are the Navier Stokes equations for an incompressible, viscous fluid. From this, the momentum equation is shown in equation (5.1):

$$\frac{\partial U}{\partial t} + \nabla.(UU) - \nabla.\frac{\mu}{\rho}\nabla U - g - \frac{F_s}{\rho} = -\frac{1}{\rho}\nabla p \tag{5.1}$$

Where U is the flow velocity, t is time, p is the pressure, μ is the viscosity, g is the acceleration due to gravity, ρ is the density of the fluid and F_s is the surface tension, which is not considered an important aspect for this research.

The continuity equation in the differential form states:

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho U) = 0 \tag{5.2}$$

If incompressible flow may be assumed, which `interFoam` does, this reduces to:

$$\nabla.U = 0 \tag{5.3}$$

The density for the 2 phases, ρ, is defined through the VOF method:

$$\rho = \rho_1\alpha + \rho_2(1-\alpha) \tag{5.4}$$

Further, the equation for the volume fraction α has to be solved:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\alpha U) + \nabla \cdot (\boldsymbol{u_c}\alpha(1-\alpha) = 0 \tag{5.5}$$

Where $\boldsymbol{u_c}$ is the artificial normal compression velocity of the interface. This is done in `interFoam` to prevent interface dispersion (Afshar, 2010).

5.5.2. Volume of Fluid method in multiphase flow

One of the fundamentals of multiphase flow is the interface capturing. This essentially comes down to calculating where the interface between the different phases lies. If interface capturing methods are omitted entirely, numerical diffusion will occur, and non-physical results are bound to arise (Peeters, 2016).

A well-known example of an interface capturing method is the Volume of Fluid (VOF) method. This is a method in which each of the control volumes (CV) in the grid get assigned a volume fraction α. The following condition is set:

$$0 \leq \alpha \leq 1 \tag{5.6}$$

In this, the values α = 1 and α = 0 correspond to the CV only containing 1 single phase. If α takes a value lower than 1 and higher than 0, the CV is partially filled with both fluids, and the interface between the fluids lies within that CV.

This holds for two fluid systems, but also for multiple (n) fluid systems. In that case, a fluid fraction is defined for all fluids. Equation (5.7) shows the fluid fraction for the m-th fluid for each CV:

$$0 \leq a_m \leq 1 \tag{5.7}$$

And the sum of all n fractions in each CV should be equal to one:

$$\sum_{m=1}^{n} \alpha_m = 1 \tag{5.8}$$

Using the volume fractions, for each CV, the physical properties are calculated as a weighted averages. In a system with 2 fluids, typically only 1 fluid fraction (α) is defined. For example, the density ρ and viscosity μ are then averaged for each CV following equations (5.9) and (5.10):

$$\rho = \rho_1 \alpha + \rho_2 (1 - \alpha) \tag{5.9}$$

$$\mu = \mu_1 \alpha + \mu_2 (1 - \alpha) \tag{5.10}$$

The spatial derivative of the value for α can be used to compute the orientation of the interface if that's needed. An example of this is the Least Squares Volume of Fluid Interface Reconstruction Algorithm (LVIRA). We will not go into that here, (Peeters, 2016) explains a lot more about it.

5.5.3. Pressure momentum coupling

The equation for momentum contains a term for pressure; however, there is no equation for pressure. So when solving the momentum and continuity equations, a guessed value for pressure is initially used. Using the guessed value for pressure, a velocity can be computed but it will most likely not satisfy the continuity equation. This is called the pressure momentum coupling and multiple pressure correction methods have been developed.

Some examples of methods are: PISO, SIMPLE and PIMPLE (PISO-SIMPLE). The difference between these three lies in the correction equations and the number of correctors, and whether they are inner or outer correctors. (Peeters, 2016) noted SIMPLE has been designed with steady-state flow problems in mind (Patankar, 1980). The nature of the flow problems in this thesis are transient start-up behaviour and bed formation are both time dependent. OpenFOAM has multiple solvers (including `interFoam`) which can operate all three methods.

The PISO algorithm uses multiple prediction and correction steps to solve the pressure momentum coupling. (Issa, 1986) developed this method and explains the inner workings. In OpenFOAM, the settings to drive the PISO algorithm are `nCorrectors` and additionally settings for non-orthogonal correction can also be set. `nCorrectors` is set to 2 throughout this thesis.

We are not planning on using the PIMPLE algorithm, the solver operates the PIMPLE method in PISO mode by not applying any field or equation relaxation.

5.5.4. Discretization / differential schemes

Each of the governing equations should be discretized. For spatial discretization this should be following the mesh or grid definition. In OpenFOAM, this is controlled by user-defined differential schemes. These schemes describe and instruct how interpolation should be performed. This is for instance needed when calculating a cell face flux based on a value that's stored on cell centres.

For time discretization, the Euler implicit temporal scheme (first order accurate) is used. Equation (5.11) shows the Euler scheme for scalar ϕ.

$$\frac{\partial \phi}{\partial t} = \frac{\phi - \phi^0}{\Delta t} \tag{5.11}$$

Where ϕ is the scalar at the current time step and ϕ^0 is the value of the same scalar at the previous time step.

The spatial discretization schemes used to interpolate values from cell-centres to their respective faces must also be defined. (Jasak, 1996) shows a Central Differencing scheme in equation (5.12). This interpolation assumes linear variation of value ϕ between nodes P and N.

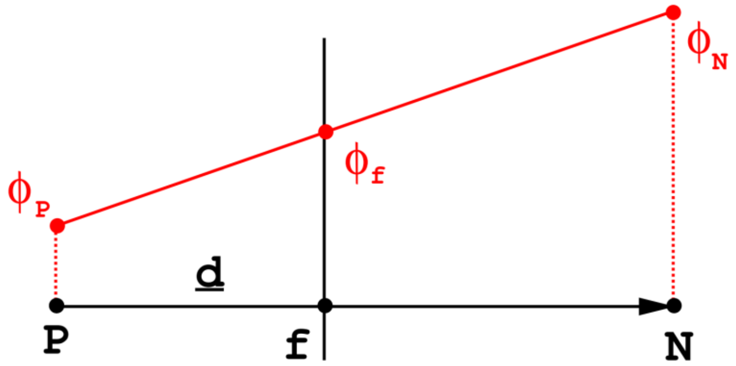

Figure 5.3: Cell centre to face interpolation for nodes P and N. Source: (Jasak, 1996)

$$\phi_f = f_x \phi_P + (1 - f_x)\phi_N \tag{5.12}$$

Where ϕ_P and ϕ_N are the values of scalar ϕ at node P and node N respectively, f_x represents the ratio of the distances $\overline{fN}$ and $\overline{PN}$:

$$f_x = \frac{\overline{fN}}{\overline{PN}} \tag{5.13}$$

In OpenFOAM, this interpolation method can be used by setting `Gauss linear` in the `fvSchemes` dictionary.

(Jasak, 1996) also shows a scheme for Upwind Differencing. This is given following equation (5.14).

$$\phi_f = \begin{cases} \phi_f = \phi_P \;\; for\, F \geq 0 \\ \phi_f = \phi_N \;\; for\, F < 0 \end{cases} \tag{5.14}$$

In which the value for ϕ_f is determined by the direction of the flow.

For the divergence terms, a combination of `Gauss linear`, `Gauss linearUpwind`, and `Gauss vanLeer` is used. Again, each of the schemes used for the simulations performed later are shown in Appendix D.

Non-orthogonality in the mesh is handled in the corrected scheme for the Laplacian schemes only; they are computed using `Gauss linear corrected`; the `snGradScheme` is also set to corrected.

5.5.5. Matrix solvers

The equations that are solved are reduced to the linear algebraic problem in equation (5.15).

$$Ax = b \tag{5.15}$$

The algorithms used to solve these are a combination of `smoothSolver`, `GAMG`, and `PCGGAMG`. Each of these solvers has different methods for solving the equations. Each has settings for residual control and (relative) tolerances. For all simulations we've ran, these settings are also included in Appendix D.

5.5.6. MULES correction on α

For the volume fraction α, additional corrections can be made. For each timestep, a number of sub-cycles can be performed to calculate α. This can be put in place to increase stability when the solver is operating at larger Courant numbers, (Peeters, 2016), (Damián, 2013). This sub-cycling is provided by the MULES algorithm. Further, this method has been put in place to maintain boundedness of the phase fraction α. That is then independent of the underlying numerical schemes and it is mesh independent. This in turn leads to a more free choice in schemes for, for example, convection (divSchemes)[2]. Following the advice in the same reference, the discretization schemes are chosen.

Two parameters to drive MULES have to be defined: `nAlphaCorr` and `nAlphaSubcycles`. These are set to 1 and 5 respectively. Further, there's an `nLimiterIter` which is set to 3.

5.5.7. Courant-Friedrichs-Lewy condition

The Courant Friedrichs Lewy (CFL) limit is a stability criterion that needs to be satisfied. It's a measure of stability for simulations of flows based on a maximum time step Δt, (Courant, Friedrichs, & Lewy,

[2] https://www.openfoam.com/documentation/tutorial-guide/tutorialse8.php#dx14-75004

1928), (Peeters, 2016). Equation (5.16) presents the formulation for the well-known (dimensionless) Courant number Co in 3 spatial dimensions.

$$Co = \Delta t \sum_{i=1}^{3} \frac{u_i}{\Delta x_i} \leq Co_{max} \tag{5.16}$$

Where u_i represents the velocity in direction i, and Δx_i is the cell size in the same direction. A typical maximum value used is $Co_{max} = 1$. For explicit solvers, this is a hard limit (Peeters, 2016).

The criterion can also be used in the reverse manner, where the (next) timestep is computed from the maximum (user specified) value for the Courant number, Co_{max}. The `interFoam` solver can compute the timesteps from the CFL limit dynamically. This criterion is defined as follows:

$$\Delta t_{max} \leq min \left\{ \Delta t_{u,max}, \frac{Co_{max}}{\sum_{i=1}^{3} \frac{u_i}{\Delta x_i}} \right\} \tag{5.17}$$

Where Δt_{max} is the maximum timestep based on the CFL limit, and $\Delta t_{u,max}$ is the maximum user defined timestep.

5.6. Code adjustments

The `interFoam` solver will need to be adjusted such that it cope with a mixture of a non-Newtonian fluid and sand particles. We will need to implement equations for the sand transport and settling. Section 5.6.1 and 5.6.2 show the changes that are performed on the solver, and section 5.6.3 will present the changes needed to allow for non-Newtonian materials.

5.6.1. Solver adjustments

First up is the solver code for `interFoam` itself. This section describes the adjustments made to allow for sand transport and segregation in the `interFoam` solver.

5.6.1.1. Sand transport

The adjustments required to add the sand transport equation to the solver is highlighted in this section. The actual source code for the solver can be found in Appendix B. Large parts of the code were provided by Cees van Rhee as these were also used in his work (van Rhee, 2017).

The sand transport equation has been implemented following the drift flux method as described by (van Rhee, 2017). Equations (3.10) and (3.11) show that. In code, this looks as follows:

```
Us = U + wsvol;

surfaceScalarField phised = fvc::interpolate(Us) & mesh.Sf();

fvScalarMatrix csandEqn
(
      fvm::ddt(csand)
    + fvm::div(phised, csand)
);
csandEqn.solve();
```

`U` represents the velocity field of the fluid mixture. `Us` symbolizes the velocity field of the sand particles and `wsvol` is the settling velocity of the sand parties.

`phised` symbolizes the flux of the sand particles between the cells of the mesh. It is a `surfaceScalarfield` because the flux is a scalar. The “`surface`“ part of that means that it’s stored on the surface of the cells in the mesh.

The velocity field `U` and `Us` are stored at the centre of each cell (`vol` keyword in `volVectorField`), but the flux between cells is the value on the faces. So to get the flux of the sand, we need to interpolate to the cell faces. The `fvc::interpolate()` method returns the interpolated velocity on the cell faces.

`mesh.Sf()` gives back the cell face area vectors. `&` is the operator for the scalar product. So the scalar product of the interpolated velocity on the cell faces and the cell face vectors gives the flux.

Further, following equation (3.10) is calculated and solved on lines 5-10 of the above code.

As far as the transport equation goes, that’s all that’s needed.

5.6.1.2. Settling velocity

In the code that calculates the transport of sand (section 5.6.1.1), the settling velocity of sand is used. The (hindered) settling velocity is implemented following equation (3.12) and includes the buoyancy effect and equation (2.15) that gives the unhindered settling velocity. In code, this looks as follows. Table 5.1 presents how the nomenclature of equation (3.12) and (2.15) translates into the code below.

```
        dimensionedScalar one("one", dimless, 1.0);

        dimensionedScalar factor("factor", dimless, 1.0/18.0);

        volScalarField muws_mixture = mixture.muws();

        wsvol  = factor * (((rhos-rho)*sqr(Diam)*g) / (muws_mixture));

        wsvol *= (one - csand);
```

Code variable in above code	**Variable in equation (3.12) and (2.15)**
`mixture.muws()`	μ_{cf}
`rhos`	ρ_s
`rho`	ρ_{cf}
`g`	g
`sqr(Diam)`	d^2
`csand`	c_s
`wsvol`	w_s

Table 5.1: Code and equation variable mapping for settling velocity

`mixture.muws()` returns the viscosity of the mixture without the influence of sand particles μ_{cf}; section 5.6.3 elaborates more on this.

5.6.2. More solver adjustments: sand particle influence on mixture density

So far, the influence of the presence of sand particles is not taken into account in the calculation of the total mixture density.

Simulation 1, 2, 3 and 4 have been performed without this influence: the density of the sand is not taken into account in the calculation of the mixture density. As has been concluded in simulation 4, this is exactly the suspected cause of the pipe filling up completely instead of partially.

Simulation 5 will therefore take the density of the sand particles into account when calculating the mixture density. This will then be used to override the density value for later use in the momentum equation. The other adjustment required is that the settling velocity of the sand particles, following equation (2.15), is calculated based on the density of the carrier fluid (without the sand particles).

As can be seen in section 5.6.1.2, the initial implementation of the settling velocity equation would not be sufficient; it used a variable called `rho` which now represents the density of the entire mixture including particles. Instead, we should define a new variable for the density of the carrier fluid only. We call this `rho_cf`.

The following adjustment has been made in the code for the sand transport file. On line 3 we can see the calculation for the density of the carrier fluid only, line 4 shows this is being used to calculate settling velocity, and line 5 and 6 show the new mixture density is being calculated.

```
1        // Calculate carrier fluid density rho_cf for later use in calculating sand
2           particle settling velocity

3        rho_cf = alpha1*rho1 + alpha2*rho2;

         … // code omitted here for ease of reading


4        wsvol = factor * (((rhos-rho_cf)*sqr(Diam)*g) / (muws_mixture));

         … // code omitted here for ease of reading


         // Calculate rho_ws (density with sand particles included)
5        rho_ws = (1 - csand)*rho_cf + csand*rhos;

6        // Overwrite the value for rho with the newly calculated rho_ws
         rho = rho_ws
```

5.6.3. Non-Newtonian material model

In C++, abstract classes (sometimes also called interfaces) can be used to dictate each concrete inheriting class must have some (overriding) implementation of the virtual functions in the abstract class.

The abstract `viscosityModel` class provided by OpenFOAM has a virtual function that should return the apparent (kinematic) viscosity. This dictates that every descendant of this class must have a function that returns the apparent viscosity. This is the apparent (kinematic) viscosity as described in section 2.1.2 / equation (2.6).

5.6.3.1. Bingham Plastic

To be able to calculate the settling velocity based on the viscosity of only the carrier fluid (sand particles omitted), we've extended the abstract `viscosityModel` class to also have a virtual function that returns the field for viscosity without sand particles, `nuws()`.

```
        //- Return the laminar viscosity
        virtual tmp<volScalarField> nu() const = 0;

         //- Return the viscosity for settling velocity
        virtual tmp<volScalarField> nuws() const = 0;
```

This function now must have the name `calcNuws()` and it returns a `volScalarField` object. This is the apparent kinematic viscosity without the influence of the sand particles (ν_a). "`ws`" here represents "without sand".

It follows equation (2.2), (2.6), (2.7) and (3.9). Combining these equations yields equation (5.18) for the apparent kinematic viscosity ν_a without the influence of sand particles.

$$\nu_a = \frac{\tau_{y,\rho}(1 - e^{-m\dot{\gamma}}) + \mu_{b,\rho}\dot{\gamma}}{\dot{\gamma}} \tag{5.18}$$

Where $\tau_{y,\rho}$ is the yield strength of the fluid divided by the density of the carrier fluid, $\mu_{b,\rho}$ is the plastic viscosity divided by the density of the carrier fluid.

In code, this is implemented as follows. A mechanism is implemented to prevent dividing by 0, which could happen with mixture that have a non-zero yield stress, in the plug of the flow. This is done capping the strain rate at `VSMALL`, a very small value in OpenFOAM. Its value is 1e-300.

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Talmon::calcNuws() const
{
    dimensionedScalar one("one", dimless, 1.0);

    tmp<volScalarField> sr(strainRate());

    return
    (
        min(
            numax_,
            nu0_ +  ( tau0_*( one-exp(-coef_*sr()) ) )
            /(max(sr(), dimensionedScalar ("VSMALL", dimless/dimTime, VSMALL)))
        )
    );
}
```

The name Talmon is chosen for the class, because (van Rhee, 2017) used the same name. It's based on (Talmon, Hanssen, Winterwerp, Sitoni, & van Rhee, 2016), I believe.

The class also has a function called `calcNu()` to calculate the apparent kinematic viscosity *including* the influence of sand $\nu_{a,s}$ following equation (2.2), (2.6), (2.7), (2.8), (2.9), and (3.9) yielding:

$$\nu_{a,s} = \frac{\tau_{y,\rho}\left(1 - e^{-m\dot{\gamma}}\right)e^{\beta\lambda} + \mu_{b,\rho}e^{\beta\lambda}\dot{\gamma}}{\dot{\gamma}} \tag{5.19}$$

In code, this is implemented as follows:

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Talmon::calcNu() const
{
    dimensionedScalar one("one", dimless, 1.0);
    dimensionedScalar one3("onethird", dimless, 1.0/3.0);
    dimensionedScalar klein("klein", dimless, 1e-5);
    tmp<volScalarField> sr(strainRate());
    volScalarField labda_= one / (pow(cmax_/(alpha_+klein),one3)-one);
    Info<< " Max waarde van alpha_ in calcNu" << max(alpha_) << endl;
    Info<< " Min waarde van alpha_ " << min(alpha_) << endl;
    Info<< " Berekening van labda " << max(labda_) << endl;
    return(
        min(_,
            nu0_*exp(alpha0_*labda_) +
            (tau0_*exp(alpha0_*labda_) * (one-exp(-coef_*sr())) )    /
            (max(sr(),dimensionedScalar("VSMALL", dimless/dimTime, VSMALL)))
         ));
}
```

5.6.3.2. Capped exponents and λ

It was quickly found that the exponents $e^{\beta\lambda}$ runs to infinity for c_s close to c_{max}. For varying values of c_s (`csand`) the value of λ (`labda`) can be plotted, this is shown in Figure 5.4. The value for c_{max} in this plot is set at 0.6.

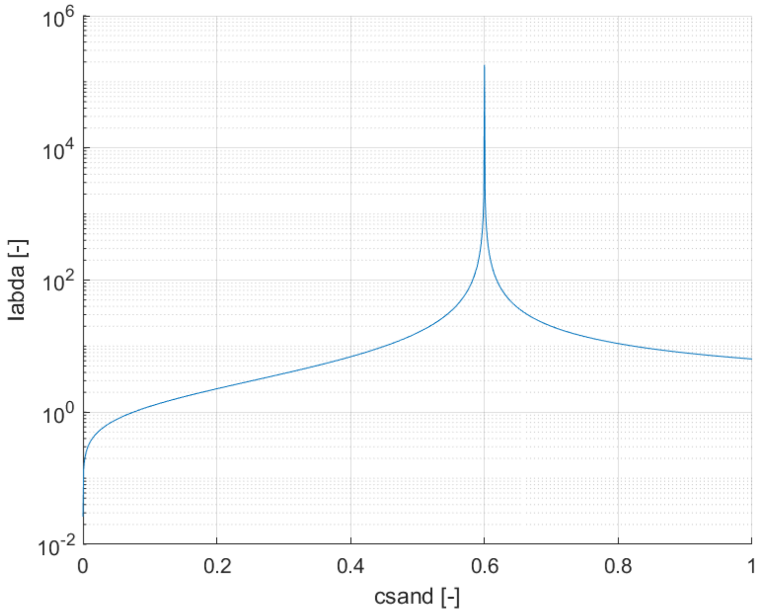

Figure 5.4: Linear sand concentration λ (`labda`, on the y-axis) for varying c_s (`csand`, on the x-axis), $c_{max} = 0.6$

We see that for c_s values really close to c_{max} (0.6 in Figure 5.4), λ becomes really large; for $c_s = c_{max}$, λ tends to infinity. This is a problem as soon as it is multiplied by β and the exponent $e^{\beta\lambda}$ is computed. Therefore, an adaptation to this has been proposed to make sure this exponent stays below a very large value in OpenFOAM. This large value is called `ROOTVGREAT`, and it's value is 1e150.

$$max\ (e^{\beta\lambda}, ROOTVGREAT) \tag{5.20}$$

Further, the value for λ is also capped to a maximum. Recall equation (2.10), the suggested adaptation looks like this:

$$\lambda = min\left(\frac{1}{\left(\frac{c_{max}}{c_s + 10^{-5}}\right)^{\frac{1}{3}} - 1}, \lambda_{max}\right) \tag{5.21}$$

In which λ_{max} is the maximum theoretical value at which the exponent $e^{\beta\lambda}$ should not exceed the literal maximum value in OpenFOAM (`std::numeric_limits<double>::max()`). This is calculated as follows:

$$\lambda_{max} = log\left(\frac{max_{OpenFOAM}}{\beta}\right) \tag{5.22}$$

Similarly, this maximizing of these values has also been added before the division by the strain rate $\dot{\gamma}$ happens; the strain rate is capped so it's always larger than a minimum small value, `VSMALL`: 1e-150. This last part was already part of the implementation as described in section 5.6.3.1.

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Talmon::calcNu() const
{
    dimensionedScalar one("one", dimless, 1.0);

    tmp<volScalarField> sr(strainRate());

    volScalarField labda_= calcLabda();

    volScalarField capped_exponent = min(exp(alpha0_*labda_), dimensionedScalar ("ROOTVGREAT", dimless, ROOTVGREAT));

    return
    (
        min(
            numax_,
            nu0_*capped_exponent + (tau0_*capped_exponent*(one-exp(-coef_*sr())) )
                /(max(sr(), dimensionedScalar ("VSMALL", dimless/dimTime, VSMALL)))
        )
    );
}
```

Following equation (5.21), the function `calcLabda()` is implemented as follows:

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Talmon::calcLabda() const
{
    dimensionedScalar one("one", dimless, 1.0);
    dimensionedScalar one3("onethird", dimless, 1.0/3.0);
    dimensionedScalar klein("klein", dimless, 1e-5);

    volScalarField labda_return = (one / (pow(cmax_/(alpha_+klein),one3)-one));

    return min(labda_return,maxlabda);
}
```

Following equation (5.22), the function for λ_{max} (`maxlabda`) is calculated once as follows in the constructor function of the `Talmon` object. It's placed inside the constructor function since it's only required to calculate this once.

```
maxlabda("maxlabda", dimless, log(std::numeric_limits<double>::max())
    /alpha0_.value()),
```

A couple of calls to the `Info` function (for logging purposes) have been omitted from the above, but are shown in Appendix C.1.

6. Simulation and validation

A number of simulations have been executed using different grids, boundary conditions and material properties. Each of the subsections in this chapter goes into detail about what has been done exactly.

All of the simulation case input files (geometry definitions, material properties, boundary conditions, and solver settings) can be found in Appendix D.

6.1. Simulation 1

First up is a rather simple simulation using the Bingham Plastic rheological model. The goal of this simulation is to recreate a simulation case comparable to the work by (van Rhee, 2017). We do this so we can compare results. Validation is done against the analytical solution for the velocity of flow down an inclined plane as presented in section 2.4.3.2.

So the goal of this simulation is to see whether the adapted `interFoam` solver is capable of recreating a 2D open channel flow using a Bingham Plastic fluid model. Most parameters are set to the same values as used by (van Rhee, 2017). For this first simulation we will not add any sand particles to the simulation.

6.1.1. Geometry

The geometry is rather simple: a 2D rectangular (open) channel with length L = 21m (x-direction), height h = 0.3m (y-direction), width b = 0.1m (z-direction). Two blocks have been defined, block 1 is there to facilitate inflow in the positive y-direction, and block 2 through which the fluid will flow and eventually exit the simulation domain.

Block 1 has dimensions 1m x 0.3m x 0.1m. It is divided into 120 x 80 x 1 cells with a `simpleGrading` 3, 5, 1. Block 2 has dimensions 20m x 0.3m x 0.1m. It is divided also into 120 x 80 x 1 cells with a `simpleGrading` 3, 5, 1. This grading will make the cells gradually smaller in the defined direction to allow for more details to be captured. The mesh is generated using standard the `blockMesh` utility[3] offered by OpenFOAM. Figure 6.1 and Figure 6.2 show the mesh as it's generated using these parameters.

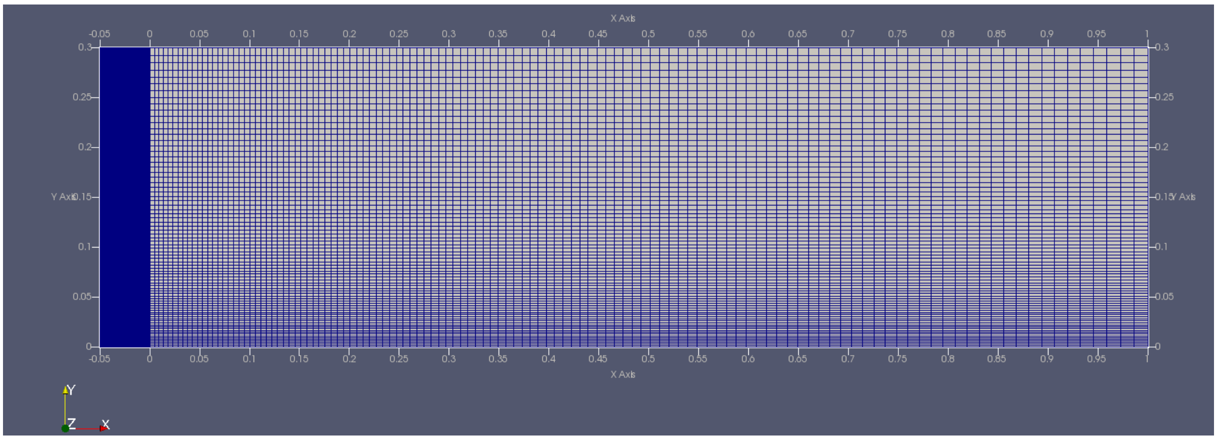

Figure 6.1: Geometry and mesh for simulation 1. Block 1 on the left hand side, block 2 on the right hand side. Note: x-axis scaled by factor 0.05

[3] https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.3-mesh-generation-with-the-blockmesh-utility

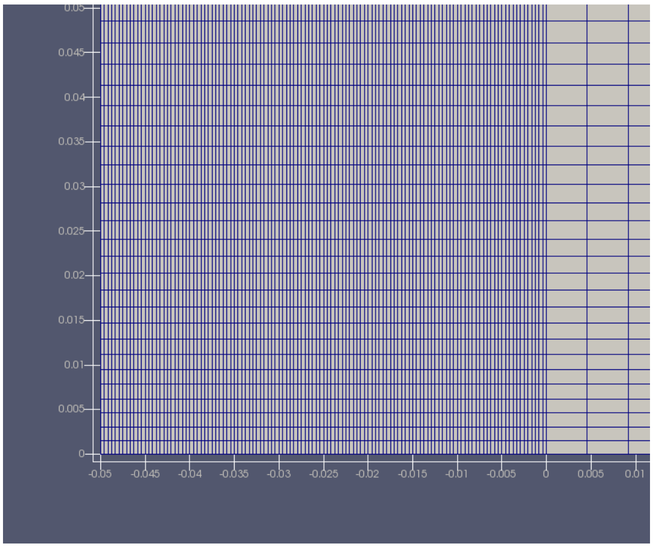

Figure 6.2: Detail of block 1, the inlet zone, for simulation 1. Note: x-axis scaled by factor 0.05

5 patches are defined on the grid. The `inlet` patch is defined in block 1, on the bottom. The `outlet` is the entire right hand side edge on block 2. The `atmosphere` is the entire top edge. On the left hand side edge of block 1 a wall is defined (`leftwall`). And the `bottom` patch is the bottom edge on block 2. Both front and back faces are empty. As such there are two non-empty solution directions: x and y. The z-direction is empty.

6.1.2. Boundary conditions

On each of the 5 patches, for each of the variables we've defined boundary condition types and values. These are shown in Table 6.1. For `csand`, `Us`, and `wsvol` the inlet boundary condition need to be set such that no sand enters the domains. Omitted from this table are the boundary conditions for `Us` and `wsvol`. They are just set such that no sand enters the domain and for completeness these files are shown in appendix D.1.

	`alpha.water`	`U`	`p_rgh`	`csand`
`inlet`	fixedValue uniform 1	flowRateInletVelocity constant 0.004	fixedFluxPressure	fixedValue uniform 0
`leftwall`	zeroGradient	noSlip	fixedFluxPressure	zeroGradient
`outlet`	zeroGradient	inletOutlet value: uniform (0 0 0)	fixedFluxPressure	zeroGradient
`bottom`	zeroGradient	noSlip	fixedFluxPressure	zeroGradient
`atmosphere`	inletOutlet inletValue 0	pressureInletOutletVelocity value: uniform (0 0 0)	totalPressure reference p0: uniform 0	zeroGradient

Table 6.1: Boundary condition types used in simulation 1

Initially, the entire domain is at rest. At the inlet, a flow of Q = 4 l/s enters the domain in the positive y-direction (upwards) at t = 0 sec. For reference, the inlet velocity as used by (van Rhee, 2017) U = 0.4m/s. The inlet area for our grid is A = 0.1 x 0.1 (width x height of the inlet). $Q = UA$, thus Q = 4 l/s.

6.1.3. Driving force

The only driving force for the flow will be a gravitational force. This is defined using a vector g (magnitude and direction). A file for this is present in the `constant` directory. Since we are interested in flow along an inclined slope, and actually angling the mesh (in blockMesh) can become somewhat indecipherable, it is easier to put the gravitational force vector under an angle and keep the mesh simpler. Components g_x and g_y are defined using vector decomposition using angle θ. The z-direction is empty, so g_z is just 0. Table 6.2 shows the angle used and the decomposition.

g (m/s^2)	**9.81**
θ (°)	2.86
g_x (m/s^2)	0.4898
g_y (m/s^2)	-9.80
g_z (m/s^2)	0

Table 6.2: Gravitational force vector decomposition for simulation 1

6.1.4. Material properties and solver parameters

The material properties in OpenFOAM are put into the `tranportProperties` dictionary. A 2 phase flow is simulated, so 2 materials are defined.

The first material is air. It's just a Newtonian model with ν = 1.48e-5 and density ρ has been set to 1 kg/m^3. Throughout this entire research these same value have been used.

For the second material, a Bingham Plastic viscosity model has been chosen. The yield stress τ_y has been set at 10 Pa and plastic viscosity μ_p at 0.2 Pa.s. These values are set to the same as was used in the 2D channel flow without sand particles in (van Rhee, 2017).

For the `interFoam` solver, those input parameters first need to be divided by density ρ before being set in the `tranportProperties` dictionary. Table 6.3 shows these.

transportModel	Talmon
rho [kg/m^3]	1249
coef m [-]	50
cmax [-]	0.6
alpha0 [-]	0.27
tau0 [m^2/s^2]	0.008006
nu0 [m^2/s]	0.00016012
numax [m^2/s]	1000e-2

Table 6.3: Material properties in the `tranportProperties` dictionary for simulation 1

It should be noted here that (van Rhee, 2017) used different transport parameters. In the `icoFoam` solver, density is not relevant or taken into account.

In the `controlDict` dictionary, we can set time step controls. In the `fvSolution` and `fvSchemes` dictionary we can configure the matrix solvers and set the discretization schemes, respectively. Again, also these files can be found in appendix D.1.

The simulation in this section used the implementation as described in section 5.6.3.2.

6.1.5. Result

The simulation is ran until t = 2000s. Then, using ParaView, we can extract multiple plots. The resulting volume fraction α can be graphically shown across the domain. Figure 6.3 shows that at t = 2000s.

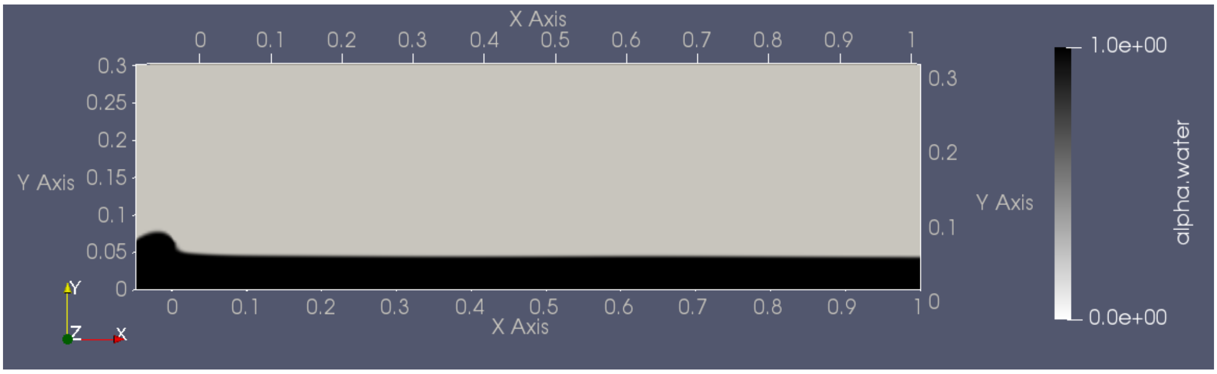

Figure 6.3: Volume fraction alpha.water at t = 2000 for simulation 1. Note: x-axis scaled by factor 0.05

At x = 19.5m, the volume fraction `alpha.water` profile and horizontal flow velocity (`Ux`) profile have been captured. This is shown in Figure 6.4.

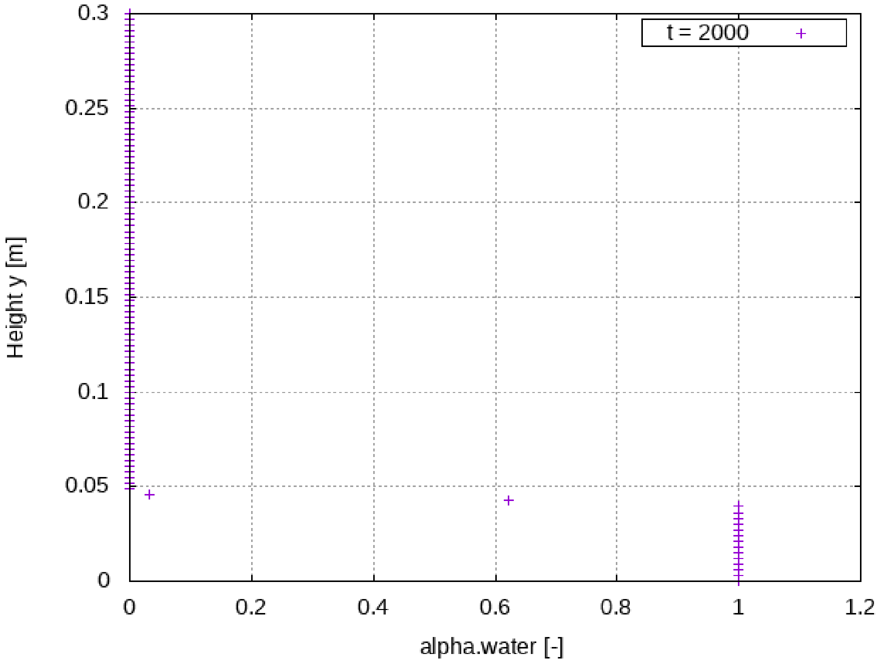

Figure 6.4: Volume fraction `alpha.water` at x = 19.5m for simulation 1

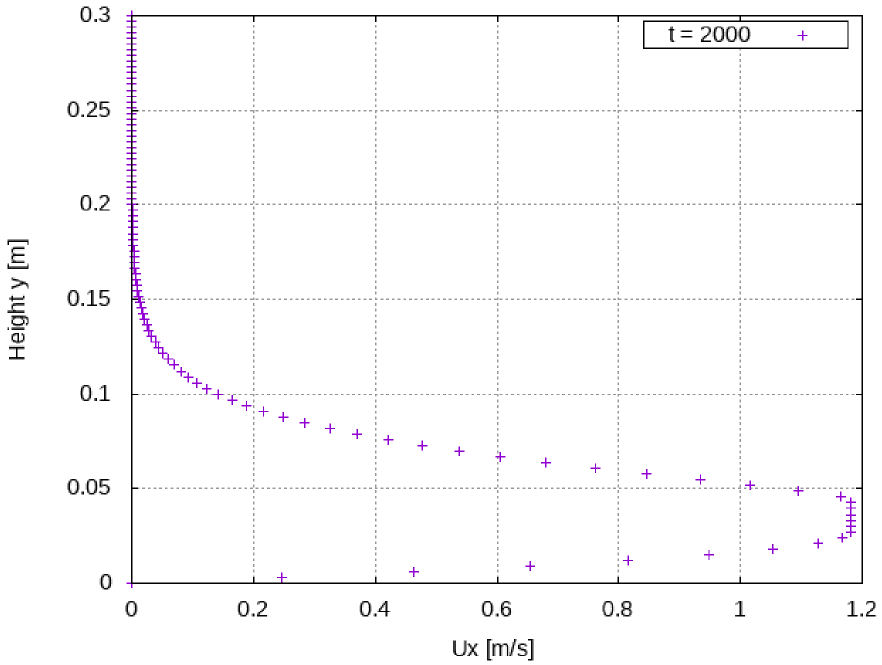

Figure 6.5: Horizontal flow velocity Ux profile at x = 19.5m for simulation 1

The flowdepth h_0 seems to stabilize at 0.044m. Both a plug zone and shearing layer can be seen. In the plug the velocity is constant. In the shearing layer, the velocity is reduced layer by layer until it reaches zero velocity at the no-slip bottom of the domain.

In Figure 6.4 we can also see that the velocity along the top of the domain tends to 0 m/s. This is not what we would have expected, because we were trying to simulate a box without any lid on the top. It should have just been an open top with slipping flow. In hindsight, this seems to have been an issue with the boundary condition on that patch. Looking into the documentation, it's not evident what went wrong here. We did not specify anything for the `tangentialVelocity` keyword, which should have result in allowing slipping tangential flow on that patch. It's apparent that it didn't. This was only noticed after all simulations have been executed. Otherwise, we would have found a solution for this sooner.

6.1.6. Validation

The results of the simulation have been compared against the previously found analytical solution for the velocity. Figure 6.6 shows the results of the numerical simulation data for horizontal velocity $U_{x,sim}$, the analytical velocity profile U_x, which has been constructed following equations (2.34) and (2.35), shear stress τ has been constructed following (2.29) and the yield stress τ_y is shown at 10 Pa.

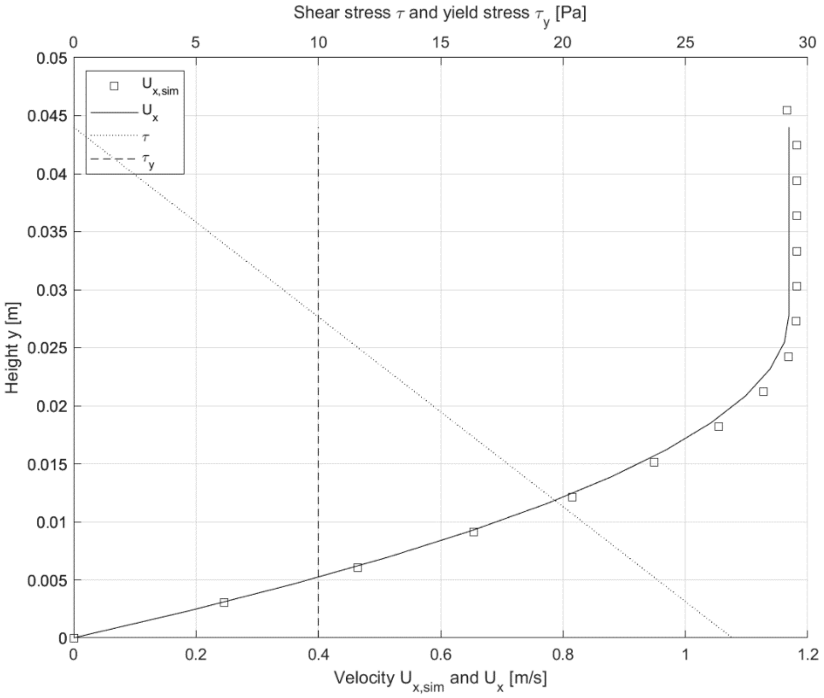

Figure 6.6: Velocity profile from analytical solution and simulation 1 results at x=15m,

We see that at the depth where the shear stress and yield stress intersect, the plug flow zone ends. The velocity that's calculated in the simulation matches quite closely with the analytical solution.

If we compare with earlier results from (van Rhee, 2017), we do see a deviation. Remember Figure 3.4 where we saw that the plug velocity is found to be roughly 0.45 m/s, much lower than the plug velocity found in the simulation: 1.18 m/s. Simultaneously, we can also see that continuity is upheld, because simultaneously the flowdepth h_0 found in the simulation is 0.044m, whereas this was fixed at 0.1m for (van Rhee, 2017). The shape of the velocity profiles shows the characteristics of the flow are the same: plug flow on top of a shearing layer and no slip at the bed.

6.1.7. Conclusions

The results found match the analytical solution for the velocity profile of a Bingham Plastic well. The velocity in the plug seems to be overestimated a little bit by the simulation. From the looks of it, this is a rather small difference and is not deemed significant.

6.2. Simulation 2

The goal of this next simulation is to see whether we can add a sand fraction to the Bingham Plastic and evaluate its flow properties. Next, we should see that sand is settling towards the bottom of the domain. Further, I expect to see a lower sand concentration in the shearing layer than in the plug zone. The plug zone is where the yield stress is preventing shear and also preventing settling of the sand particles.

Sand settling and transport has been implemented following the code described in section 5.6.1.

To get a feel for the order of magnitude of the bed sand concentration and sand concentration profile along the flowdepth, results are compared to findings by (van Rhee, 2017) and (Spelay, 2007).

6.2.1. Geometry

The geometry of the simulation domain is the same as used in simulation 1, details are noted down in section 6.1.1.

6.2.2. Boundary conditions

Table 6.4 shows the boundary condition types as used in simulation 2. The conditions for `alpha.water`, `U`, and `p_rgh` were all kept the same as in simulation 1. Hence these are excluded from the table. Appendix D.2 shows the full simulation case files.

There are two notable differences compared to simulation 1: `csand` at the inlet is now set to uniform 0.12. So we are now actually adding sand particles. This 0.12 is chosen because (van Rhee, 2017) used the same value in the pipe flow simulation.

And the second difference is the `Us` inlet flowRateInletVelocity is set to constant 0.004 so that it will get the same inlet velocity as the carrier fluid does. The inflowRate for `U` was kept the same, 4 l/s.

Further, in simulation 1, the `csand` boundary condition on the atmosphere was set to zeroGradient. In simulation 2 it was found that when the `csand` is non-zero, this results in crashes. Hence, this has been changed to an inletOutlet condition. This indeed alleviates these crashes.

	`csand`	`Us`	`wsvol`
`inlet`	fixedValue uniform 1	flowRateInletVelocity constant 0.004	fixedValue value: uniform (0 0 0)
`leftwall`	zeroGadient	noSlip	zeroGradient
`outlet`	zeroGradient	inletOutlet value: uniform (0 0 0)	zeroGradient
`bottom`	zeroGradient	noSlip	fixedValue value: uniform (0 0 0)
`atmosphere`	inletOulet inletValue 0	pressureInletOutletVelocity value: uniform (0 0 0)	fixedValue value: uniform (0 0 0)

Table 6.4: Boundary condition types used in simulation 2

6.2.3. Driving force

The only driving force for the flow will be a gravitational force. Again, the gravitational force vector is angled to allow for this. The same angle of 2.86° is used as in simulation 1.

6.2.4. Material properties and solver parameters

Again, this simulation utilizes two phases so there are two sets of material properties. The first phase, air, is configured following the same settings as for simulation 1 (see section 6.1.4).

For the second phase, a Bingham Plastic viscosity model has been chosen. Yield stress τ_y = 47.3 Pa and plastic viscosity μ_p = 0.0214 Pa.s. These material properties are set to the same as the 3D half-pipe flow in (van Rhee, 2017).

Again, for the `interFoam` solver, those material parameters first need to be divided by density ρ before being set in the `tranportProperties` dictionary. Table 6.5 shows the parameters as used in the `tranportProperties` dictionary.

transportModel	Talmon
rho [kg/m^3]	1249
coef m [-]	50
cmax [-]	0.6
alpha0 [-]	0.27
tau0 [m^2/s^2]	0.037870
nu0 [m^2/s]	1.71337e-5
numax [m^2/s]	1000e-2

Table 6.5: Material properties in the `tranportProperties` dictionary for simulation 2

In the `controlDict` dictionary, we can set time step controls. In the `fvSolution` and `fvSchemes` dictionary we can configure the matrix solvers and set the discretization schemes, respectively. Again, also these files can be found in appendix D.2.

The simulation in this section used the implementation as described in section 5.6.3.2.

6.2.5. Result

The volume fraction `alpha.water` can be graphically shown across the domain. Figure 6.7 and Figure 6.8 show this at *t* = 600s and *t* = 1200s, respectively. A red colour represents the Bingham fluid (`alpha.water`=1), and blue is air (`alpha.water`=0). On the interface between the two fluids, values between 0 and 1 for `alpha.water` are shown in a beige/orange color.

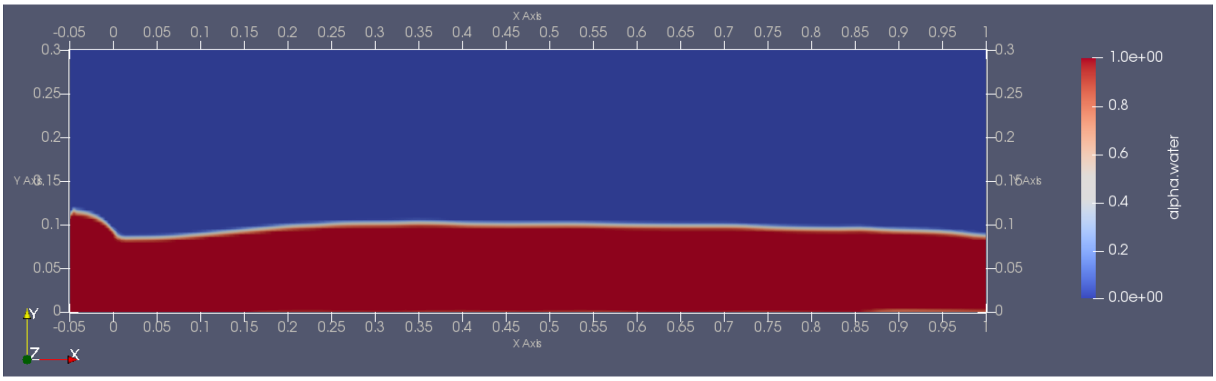

Figure 6.7: Volume fraction `alpha.water` for simulation 2 at t = 600s. Note: x-axis scaled by factor 0.05

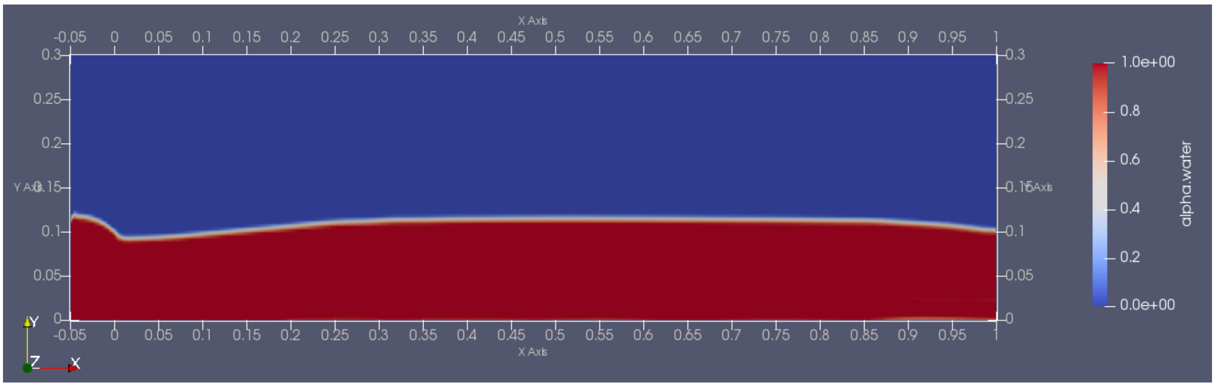

Figure 6.8: Volume fraction `alpha.water` for simulation 2 at t = 1200s. Note: x-axis scaled by factor 0.05

Similarly, the sand concentration `csand` can be visualized. Figure 6.9 and Figure 6.10 show this at *t* = 600s and *t* = 1200s, respectively.

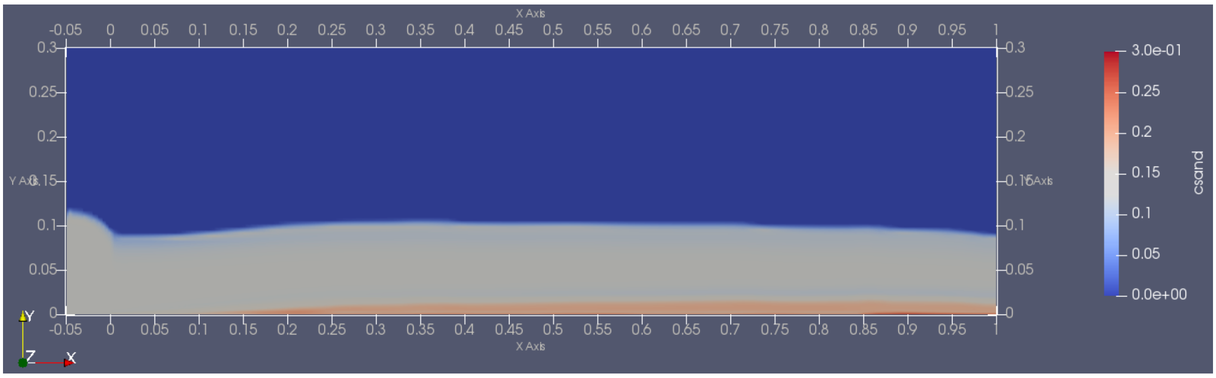

Figure 6.9: Sand fraction `csand` *for simulation 2 at t = 600s. Note: x-axis scaled by factor 0.05*

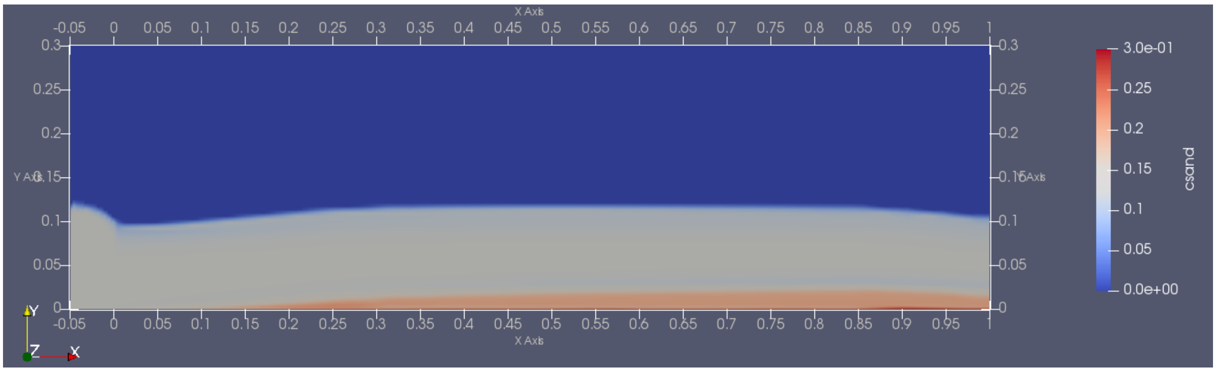

Figure 6.10: Sand fraction `csand` *for simulation 2 at t = 1200s. Note: x-axis scaled by factor 0.05*

Additionally, the profile along the flow depth for `csand` is extracted from the domain along the line x = 15m. This is shown in Figure 6.11. It can be seen that a sand bed is forming on the bottom. Further, in the shearing layer, the sand concentration is slightly lower than in the plug zone above it. Also noticeable is the lower sand concentration at the bottom vs the sand concentration directly 1 node above it (at y = 0.00303m). There's a notable drop seen, or in other words, directly at the bottom of the domain (at y = 0m) the sand concentration is noticeably higher. To this point I don't know what causes this. This dip can also be seen more prominently in Figure 6.12.

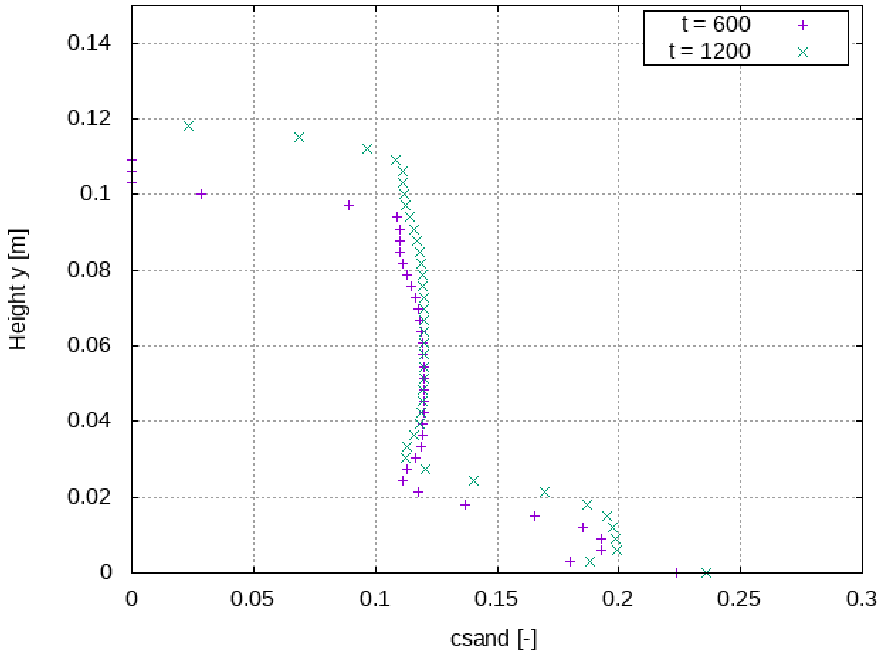

Figure 6.11: Sand fraction `csand` *for simulation 2 at t = 600s and t = 1200s at x = 15m*

As a reference, the results found in (Spelay, 2007) have also been plotted in Figure 6.12. It can be seen that the shear zone sand concentration dip is slightly smaller compared to what (Spelay, 2007) saw. (van Rhee, 2017) noted similar results.

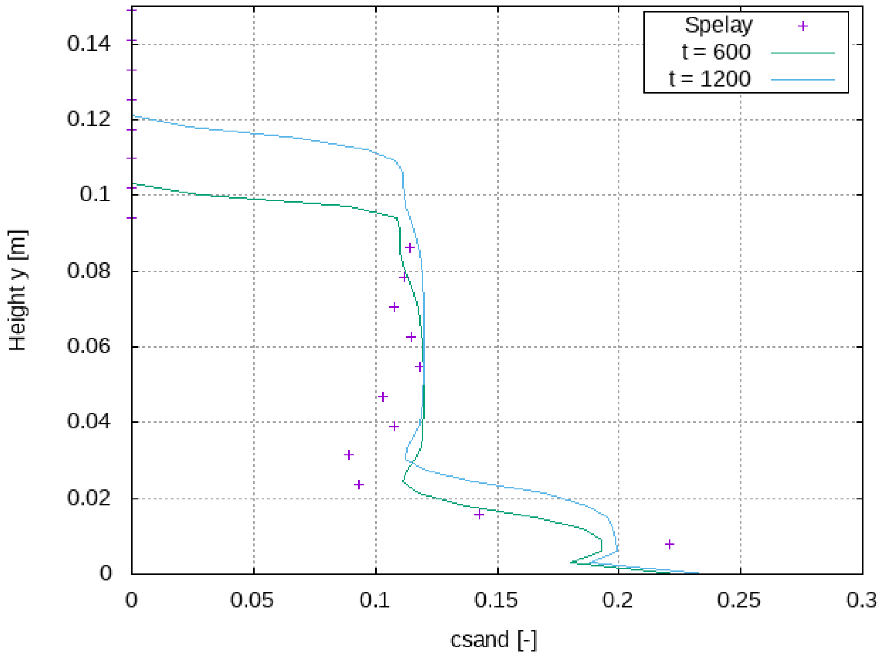

Figure 6.12: Data from (Spelay, 2007) and sand fraction `csand` *for simulation 2 at t = 600s and t = 1200s at x = 15m*

The velocity profile has also been extracted at x = 15m; this is shown in Figure 6.13. It can be seen that at the bottom of the channel stagnation occurs. This is the sand bed that has come to a halt due to increasing viscosity in the bed as well as the no-slip boundary condition on the bottom.

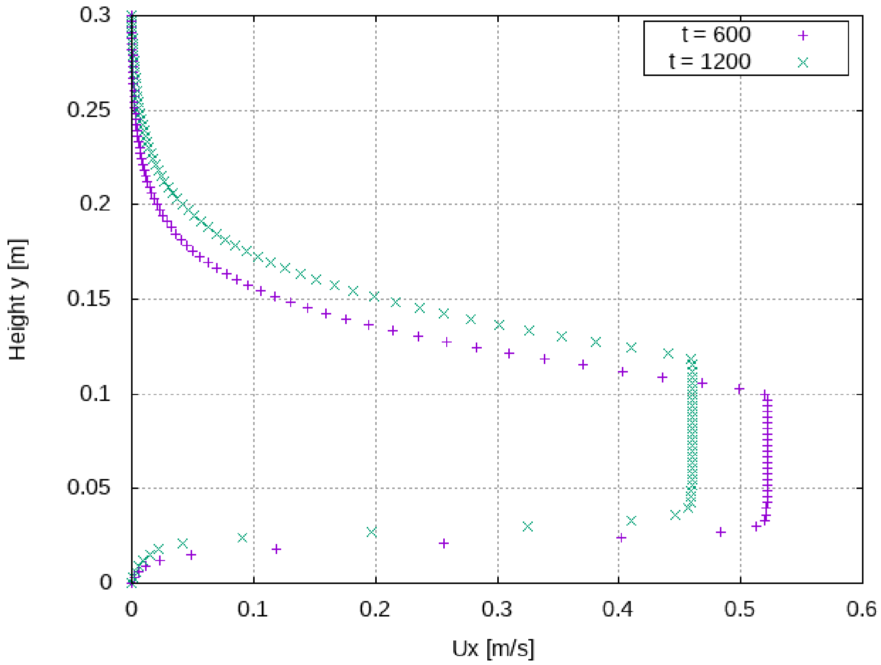

Figure 6.13: Velocity profile at x = 15m for simulation 2 at t = 600s and t = 1200s

In Figure 6.13 we can see that the velocity along the top of the domain tends to 0 m/s. Similar to our results in simulation 1 (section 6.1.5), this is not what we would have expected, because we were trying to simulate a box without any lid on the top. We again suspect this is due to the missing `tangentialVelocity` keyword on the boundary condition for velocity `U` on the atmosphere patch. This was only noticed after all simulations have been executed. Otherwise, we would have found a solution for this sooner.

6.2.6. Conclusions

Since (Spelay, 2007) performed experiments in a half open pipe, and this simulation was performed on a 2D open channel geometry, we cannot conclude whether the differences in the sand concentration profile is significant or whether it is even a problem. The same deduction can be made of comparing our results to (van Rhee, 2017). Those simulations with sand particles in the mixture have been performed in a half-pipe, not a 2D channel.

What can be concluded is that the principles of sand particles settling and a sand bed forming in a non-Newtonian fluid flow are captured using the adapted `interFoam` solver.

6.3. Simulation 3

In this simulation, I'd like to use a different hindered settling velocity model. We do this to see if this will get us even better agreement with the results found by (Spelay, 2007).

6.3.1. Case set up

The geometry of the simulation domain is the same as used in simulation 1 and 2, details are noted down in section 6.1.1. The boundary conditions of this simulation are the same as used in simulation 2 (see section 6.2.2). The only driving force for the flow will be a gravitational force. Again, the gravitational force vector is angled to allow for this. The same angle of 2.86° is used as in simulation 1 and 2. We also use the exact same Bingham Plastic viscosity model and parameters as in simulation 2 (see section 6.2.4).

The simulation in this section used the implementation as described in section 5.6.3.2.

6.3.2. Solver settings

The only difference for this specific simulation is the following. We changed the implementation for the settling velocity to now also include the return flow effect for hindered settling following equation (2.18). This equation is repeated here for ease of reading:

$$w_s = (1 - c_s)\left(1 - \frac{c_s}{c_{s,max}}\right)^2 \frac{1}{18}\frac{(\rho_s - \rho_{cf})gd^2}{\mu_{cf}} \tag{6.1}$$

The implementation of this equation is rather simple, in the CSandEqn.H file.

```
        wsvol  = factor * (((rhos-rho)*sqr(Diam)*g) / (muws_mixture));

        wsvol *= (one - csand) * sqr(one - (csand/cmax));
```

The full CSandEqn.H file can be found in appendix B.3 and the remainder of the case files (matrix solvers, discretization schemes, etc) can be found in appendix D.2.

6.3.3. Results

The volume fraction `alpha.water` can be graphically shown across the domain. Figure 6.14 and Figure 6.15 show this at *t* = 600s and *t* = 1200s, respectively. A red colour represents the Bingham fluid (`alpha.water`=1), and blue is air (`alpha.water`=0). On the interface between the two fluids, values between 0 and 1 for `alpha.water` are shown in a beige/orange color.

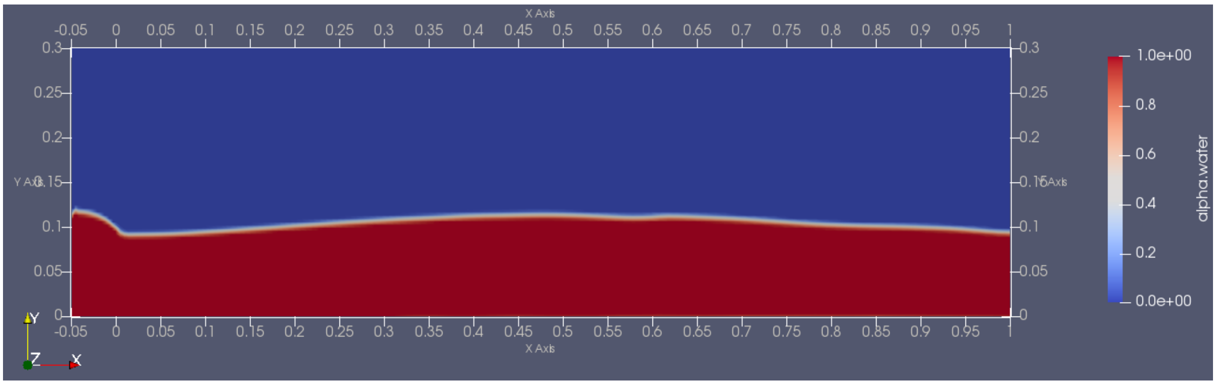

Figure 6.14: Volume fraction `alpha.water` for simulation 3 at t = 600s. Note: x-axis scaled by factor 0.05

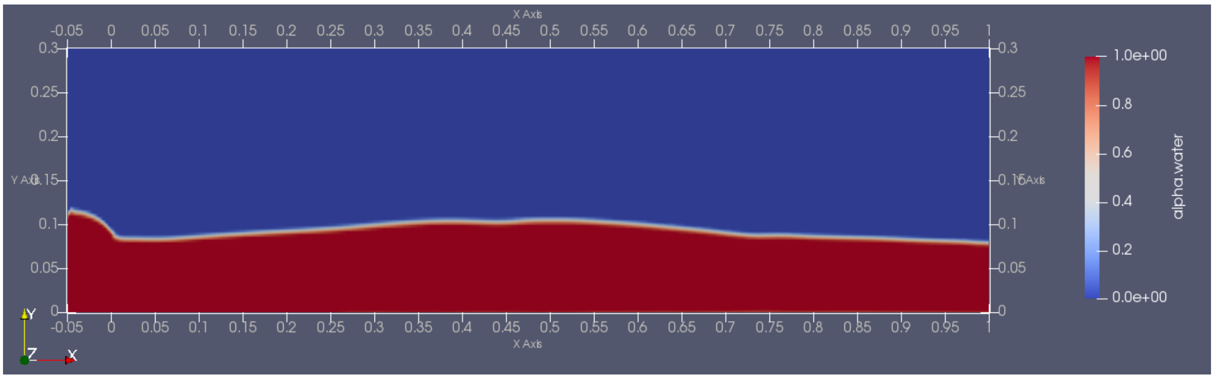

Figure 6.15: Volume fraction `alpha.water` for simulation 3 at t = 1200s. Note: x-axis scaled by factor 0.05

Similarly, the sand concentration `csand` can be visualized. Figure 6.16 and Figure 6.17 show this at t = 600s and t = 1200s, respectively.

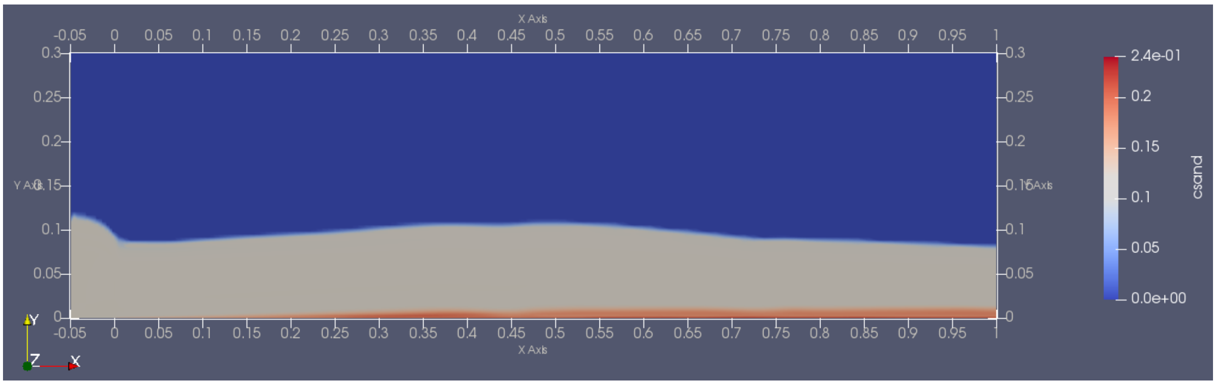

Figure 6.16: Sand fraction `csand` for simulation 3 at t = 600s. Note: x-axis scaled by factor 0.05

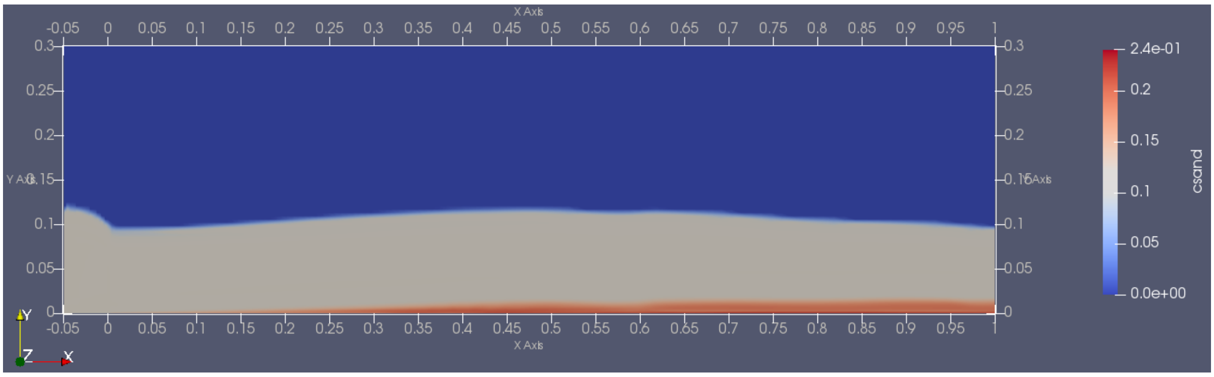

Figure 6.17: Sand fraction `csand` for simulation 3 at t = 1200s. Note: x-axis scaled by factor 0.05

Additionally, the profile along the flow depth for `csand` is extracted from the domain along the line x = 15m. This is shown in Figure 6.18. Like in simulation 2, it can be seen that a sand bed is forming on the bottom. In the plug of the flow, the sand concentration is more or less constant. In simulation 2 it was noted that the sand concentration 1 node (y = 0.00303m) above the bottom was lower than the sand concentration at the bottom (y = 0m). This can only also seen Figure 6.18 for t = 1200s, but not at t = 600s.

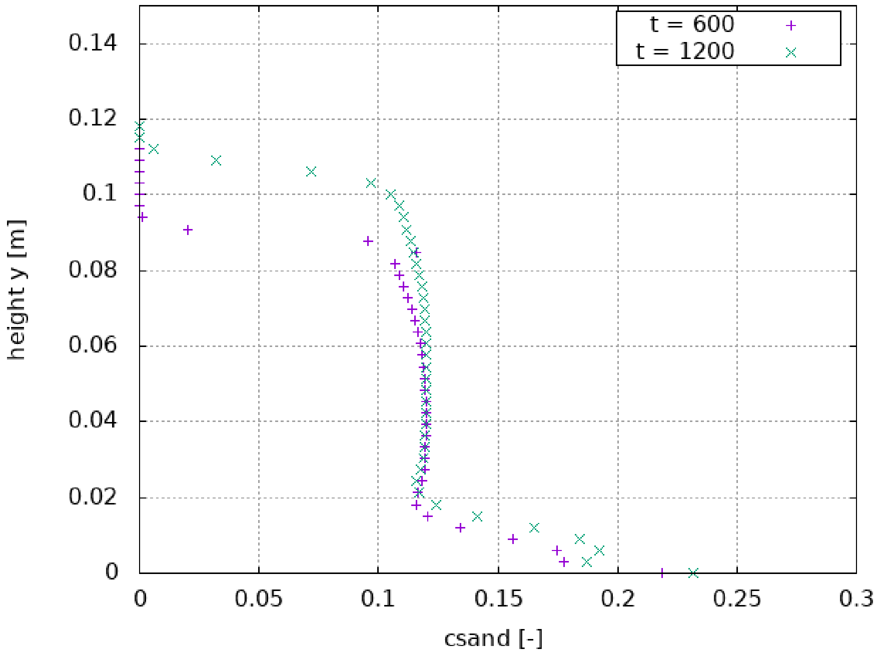

Figure 6.18: Sand fraction $csand$ *for simulation 3 at t = 600s and t = 1200s at x = 15m*

As a reference, the results found in (Spelay, 2007) and the earlier results from simulation 2 have also been plotted in Figure 6.19. It can be seen that the shear zone sand concentration dip for simulation 3 is smaller than we've seen in simulation 2. With that, it's also smaller compared to what (Spelay, 2007) saw. (van Rhee, 2017) noted similar results. What we can quite clearly see is that the sand bed that has formed in simulation 2 is a lot higher than the sand bed in simulation 3.

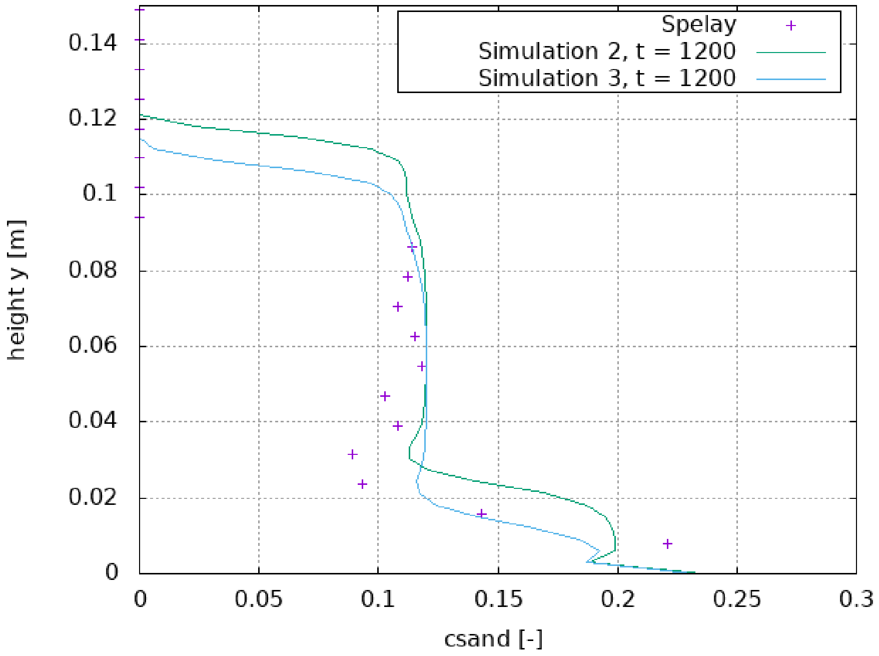

Figure 6.19: Data from (Spelay, 2007) and sand fraction $csand$ *for simulation 2 and simulation 3 at t = 1200s at x = 15m*

The velocity profile has also been extracted at x = 15m; this is shown in Figure 6.20. In Figure 6.21 we can see the velocity profile of simulation 2 and simulation 3, both at t = 600s and t = 1200s.

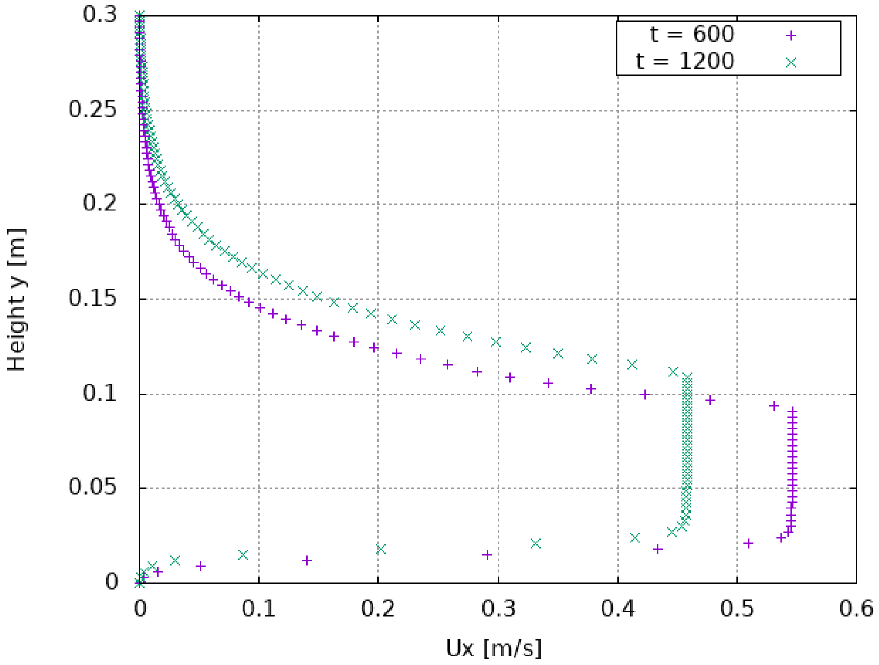

Figure 6.20: Velocity profile at x = 15m for simulation 3 at t = 600s and t = 1200s

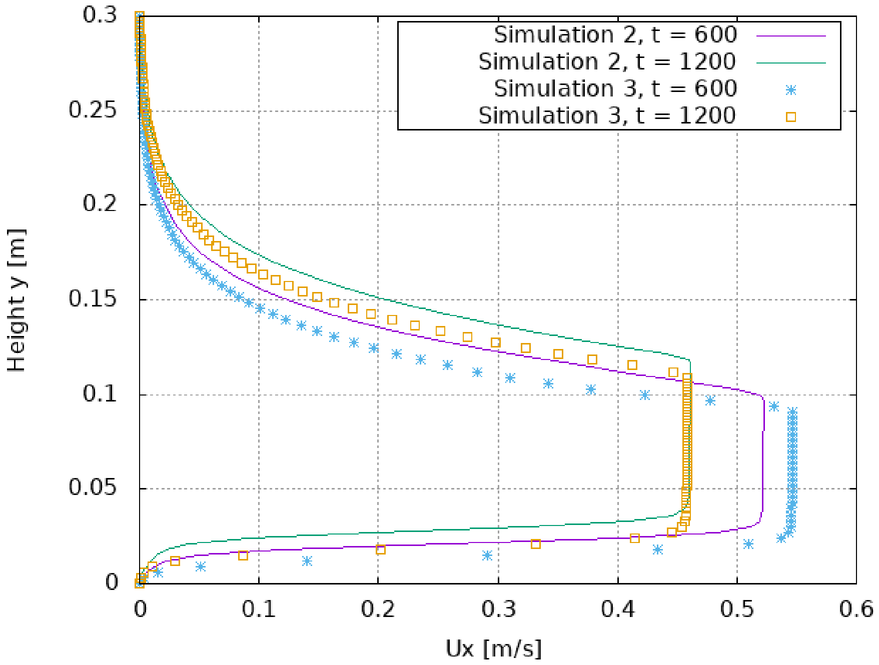

Figure 6.21: Velocity profile at x = 15m for simulation 2 and simulation 3 at t = 600s and t = 1200s

Similar to simulation 2, in simulation 3 it can also be seen that at the bottom of the channel stagnation occurs. This is the sand bed that has come to a halt due to increasing viscosity in the bed as well as the no-slip boundary condition on the bottom.

Similar to the results in simulation 1 and 2, we can see that the velocity along the top of the domain tends to 0 m/s. This is not what we would have expected, because we were trying to simulate a box without any lid on the top. We again suspect this is due to the missing `tangentialVelocity` keyword on the boundary condition for velocity `U` on the atmosphere patch. This was only noticed after all simulations have been executed. Otherwise, we would have found a solution for this sooner.

6.3.4. Conclusions

Simulation 3 has shown very similar results as simulation 2. The biggest difference that can be seen is the sand bed being less thick. This was to be expected because of our choice to implement an additional hindered settling effect on the sand. Our earlier conclusion that the principles of sand particles settling and a sand bed forming in a non-Newtonian fluid flow are captured using the adapted `interFoam` solver is still upheld.

6.4. Simulation 4

The goal of this next simulation is to recreate the experiment as it was performed by (Spelay, 2007). This means a non-Newtonian flow through a 3D pipe including sand particle transport and shear settling.

(Spelay, 2007) reported sand concentration profiles and water depths for his experiments. At least those two should be in good agreement with the results from this experiment. Additionally, the velocity profile should obviously have the typical Bingham Plastic profile.

6.4.1. Geometry

The geometry and mesh is more complex than it was in simulations 1 and 2: a 3D pipe section is used. It has length L = 17m (z-direction), and diameter D = 0.1567m. Similarly to the mesh in simulations 1 and 2, the pipe has an inlet zone and a run-off section. The inlet zone has length 2m and the run-off section is 15m in length.

The inlet zone facilitates inflow from the bottom of the pipe upwards in positive y-direction. It should be noted that due to the pipe's curvature, the actual inflow is perpendicular to each of the cells on the inlet patch. The inflow is therefore directed towards the centre of the pipe.

Figure 6.22 and Figure 6.23 show the geometry of the pipe. Please note that the z-axis has been scaled by a factor 0.05. On the left hand side we can see a red patch. This is the patch named leftWall. The blue patch on the bottom side of the pipe is the inlet patch. The orange patch is the pipeWall. And on the right hand side, the grey patch is the outlet of the pipe.

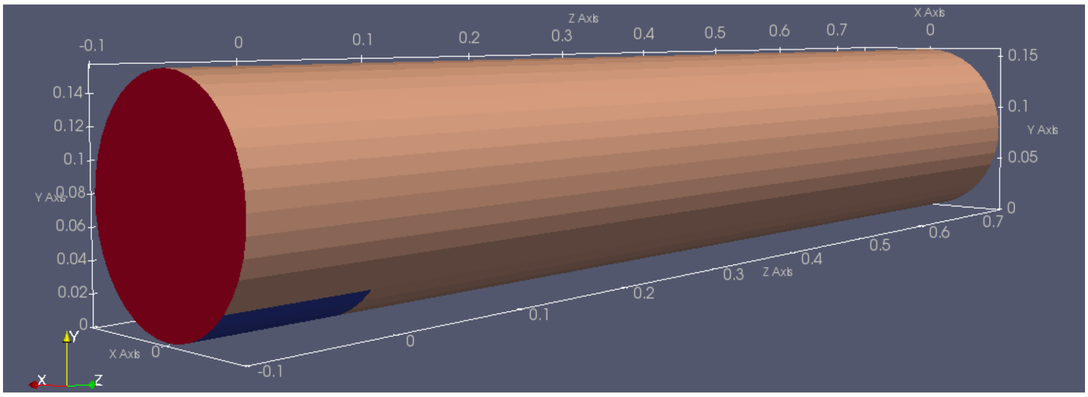

Figure 6.22: Overview of geometry for simulation 4, focus on leftWall. Note: z-axis scaled by factor 0.05

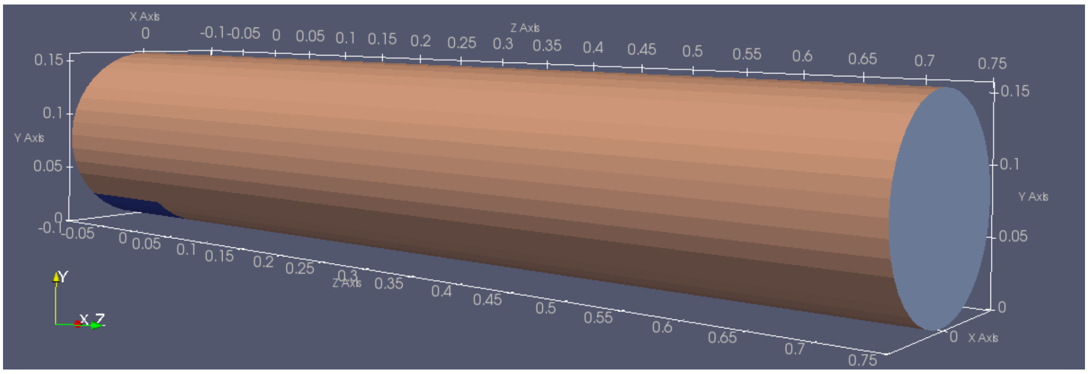

Figure 6.23: Overview of geometry for simulation 4, focus on outlet. Note: z-axis scaled by factor 0.05

Figure 6.24 shows how the mesh of the pipe has been defined. This mesh is uniform along the length of the pipe (in the z-direction). From the pipe centre outward, `simpleGrading` 3,1,1 has been applied.

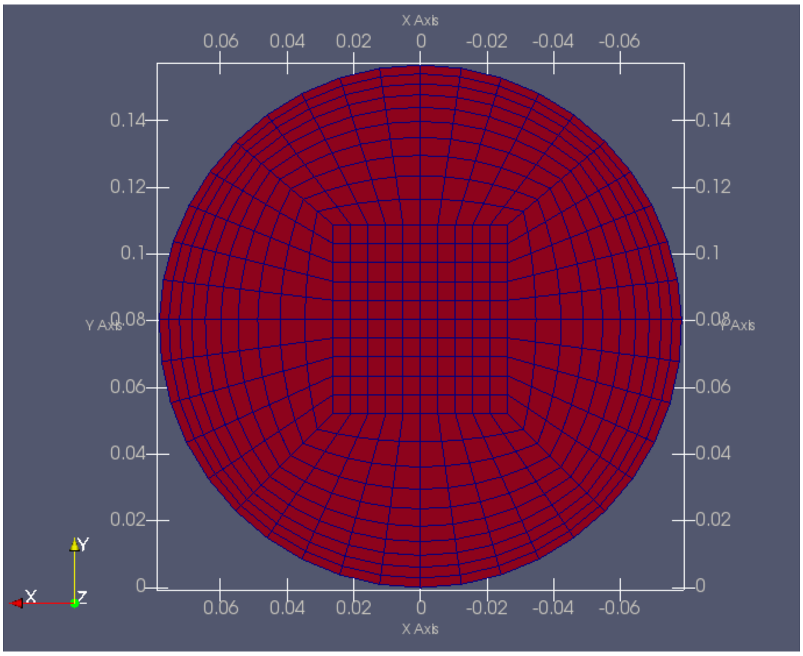

Figure 6.24: Geometry and mesh for simulation 4

It should be noted that the mesh is not entirely symmetrical. We have defined the bottom-half up to the line y = 0.0805m. Figure 6.25 has been generated with the command `paraFoam -block` command and visually shows how the blocks have been defined.

The actual midline, if it were a symmetrical mesh would have been at y = 0.07835m. This has been done to also facilitate a case where the inflow would be from the left wall of the inlet zone, and not the bottom patch. In preparation for this case, an inlet flowdepth of $h0$ = 0.0805m was created. The effects this asymmetry has on the (results of the) simulation is not further examined.

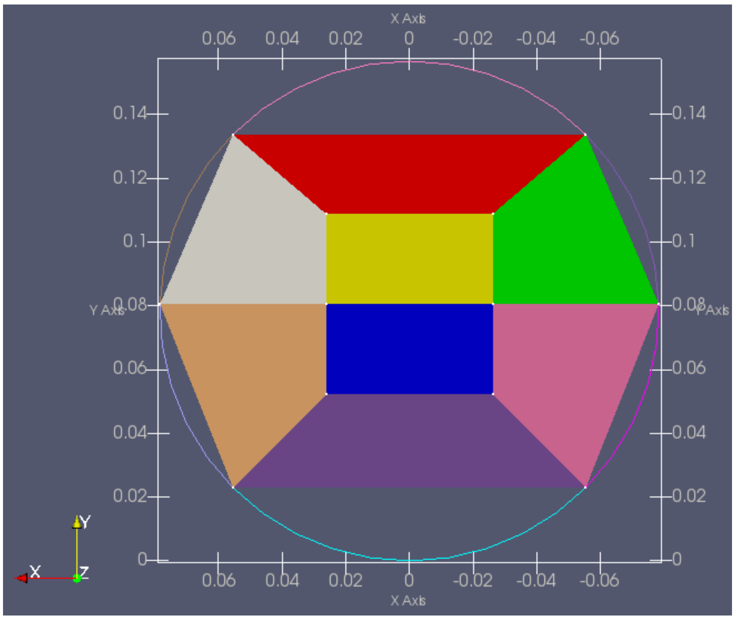

Figure 6.25: Blocks defined in mesh for simulation 4. Each colour represents a block

6.4.2. Boundary conditions and driving force

	`alpha.water`	`csand`	`p_rgh`	`U and Us`	`wsvol`
`inlet`	fixedValue uniform 1	fixedValue (see Table 6.7)	fixedFluxPressure	flowRateInletVelocity constant 0.005	fixedValue uniform (0 0 0)
`leftWall`	zeroGradient		fixedFluxPressure	noSlip	
`outlet`	inletOutlet inletValue uniform 0		prghTotalPressure reference p0: uniform 0	pressureInletOutletVelocity value: uniform (0 0 0)	zeroGradient
`pipeWall`	zeroGradient		noSlip		

Table 6.6: Boundary condition types used in simulation 4

Initially, the entire domain is at rest. At the inlet, a flow of Q = 5 l/s enters the domain through the inlet patch at t = 0s. Further, the only driving force for the flow will be a gravitational force. Since we are to recreate the experiments performed by (Spelay, 2007), the same degree inclination will be used. The same vector decomposition is applied as in for the previous cases (see section 6.1.2). The only difference is the angle θ and the fact that the vector is now decomposed in the yz-plane (Table 6.8). The value for `csand` at the inlet was set to 0.12 for simulation 4b and 4c, and 0 for simulation 4a (Table 6.7). The inlet flow rate for `Us` has been set to the same value as for `U` so that the sand will get the same inlet velocity as the fluid itself does.

	Simulation 4a	**Simulation 4b**	**Simulation 4c**
`csand` at inlet	0	0.12	0.12

Table 6.7: Sand fraction `csand` at the inlet patch

g (m/s^2)	**9.81**
θ (°)	5.4
g_x (m/s^2)	0
g_y (m/s^2)	-9.76646
g_z (m/s^2)	0.92320

Table 6.8: Gravitational vector decomposition for simulation 4

6.4.3. Material properties and solver parameters

Again, this simulation utilizes two phases so there are two sets of material properties. The first phase, air, is configured following the same settings as for simulation 1 (see section 6.1.4).

For the second phase, a Bingham Plastic viscosity model has been chosen. Yield stress τ_y = 47.3 Pa and plastic viscosity μ_p = 0.0214 Pa.s. In the implementation, those two parameters are first divided by density ρ and then put into the `tranportProperties` dictionary. Table 6.9 shows these.

After simulation 4b had been completed, a third simulation was run (4c). Taking a head-start on the results of simulation 4b, this was done in an attempt to see if the pipe will not fill up completely if we use a higher density for our fluid. In simulations 4b and 4c the only difference is the chosen density for ρ_{cf}. Simulation 4b used the density of just the carrier fluid (1303 kg/m^3) and simulation 4c used the density of the mixture (carrier fluid + sand particles: 1510 kg/m^3).

In our simulation, this new density is still assumed to be constant and not influenced by the sand concentration field. In reality, the sand particles will influence this density locally.

It should be noted that the changes on the settling velocity calculations as described in section 5.6.2 have not been applied at this point. This will probably result in an underestimation of the settling velocity as it now just uses an artificially higher fluid density.

	Simulation 4a	**Simulation 4b**	**Simulation 4c**
transportModel	Talmon	Talmon	Talmon
rho [kg/m^3]	1303	1303	1510
coef m [-]	50	50	50
cmax [-]	0.6	0.6	0.6
alpha0 [-]	0.27	0.27	0.27
tau0 [m^2/s^2]	0.0363008	0.0363008	0.0313245
nu0 [m^2/s]	1.64236e-5	1.64236e-5	1.4172185e-5
numax [m^2/s]	1000e-2	1000e-2	1000e-2

Table 6.9: Material properties in the `tranportProperties` dictionary for simulation 4

In the `controlDict` dictionary, we can set time step controls. In the `fvSolution` and `fvSchemes` dictionary we can configure the matrix solvers and set the discretization schemes, respectively. Again, these case input files can be found in the appendix: D.3.

The simulation in this section used the implementation as described in section 5.6.3.2.

6.4.4. Results

This section will go into presenting the results of simulation 4.

6.4.4.1. Figure creation

For a good understanding, it's important to explain how the figures in the next section have been created. These figures are created using a slice vertically at the midplane of the pipe. In ParaView, this is done using a Clip with Clip Type "Plane". The origin of the plane is at (0,0,0) and the normal is directed following (1,0,0). This means we are now looking inside the mixture at the vertical midplane of the pipe.

A second clip is also applied (on top of the first clip) to make the mixture interface visible. In ParaView, this can be done by applying a Clip with Clip Type "Scalar". We configure it to use `alpha.water` as the scalar, and we set the threshold value to 0.5. This will clip it right on the mixture interface and show us where our Bingham fluid is.

Further, the result is coloured by the velocity in z-direction, Uz. In semi-transparent grey, we can see the pipe wall, which is also cordoned off by the axes.

Figure 6.26 though Figure 6.41 show this for a different time for simulations 4a, 4b and 4c.

6.4.4.2. Results simulation 4a

For simulation 4a, we stopped seeing significant shifts in the water depth after about 100s. At 141s, we stopped the simulation and moved on to the next simulation, with sand: simulation 4b.

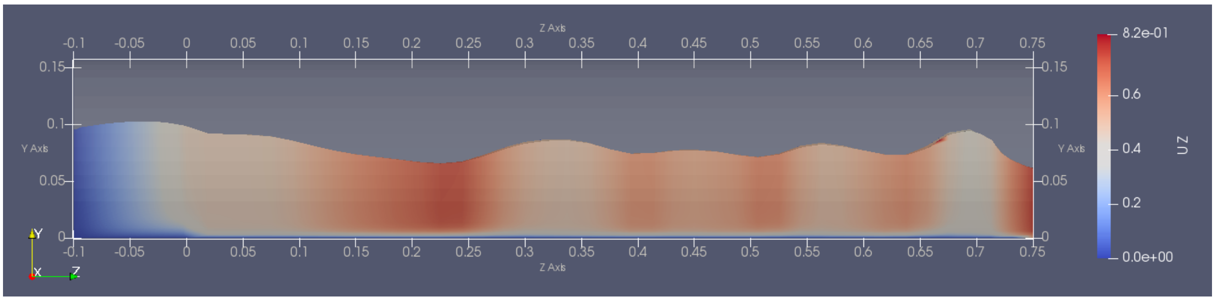

Figure 6.26: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4a at t = 50s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

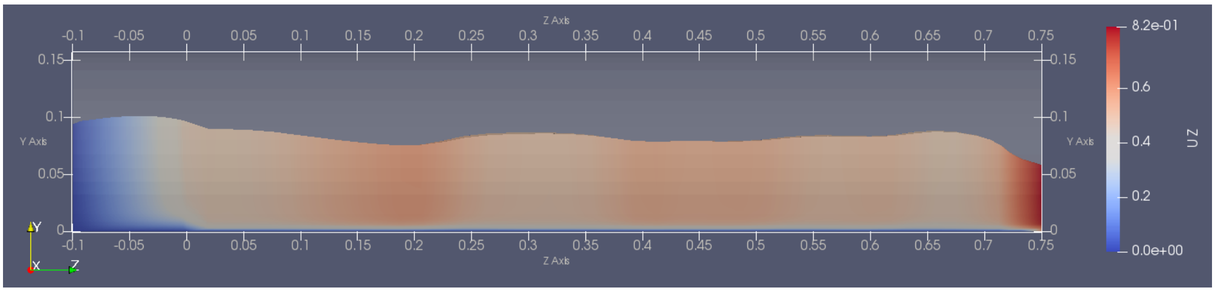

Figure 6.27: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4a at t = 100s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

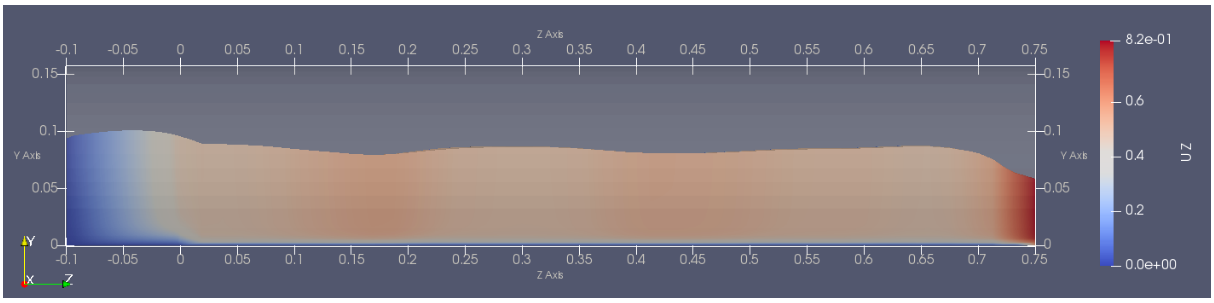

Figure 6.28: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4a at t = 141s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

6.4.4.3. Results simulation 4b

For simulation 4b, we stopped seeing any shifts in the flow between 250s and 500s. We stopped the simulation at 825s as we saw the pipe was fully filled up and this didn't seem to restore.

Initially, what we can see is that the fluid flows towards the outlet of the pipe. But as time progresses, we can see that the pipe starts to fill more and more. All the while, the velocity at outlet patch seems to influence the flow field in the pipe. This is an unexpected result as the outlet of the pipe should have been configured such that it allows free outflow (zeroGradient).

Due to the sand being included, the mixture is now a lot more viscous compared to simulation 4a. It seems to be simply so viscous that the resistance against flowing is too high. Especially if we compare it to the results found in simulation 4a, where the pipe did not seem to fill up.

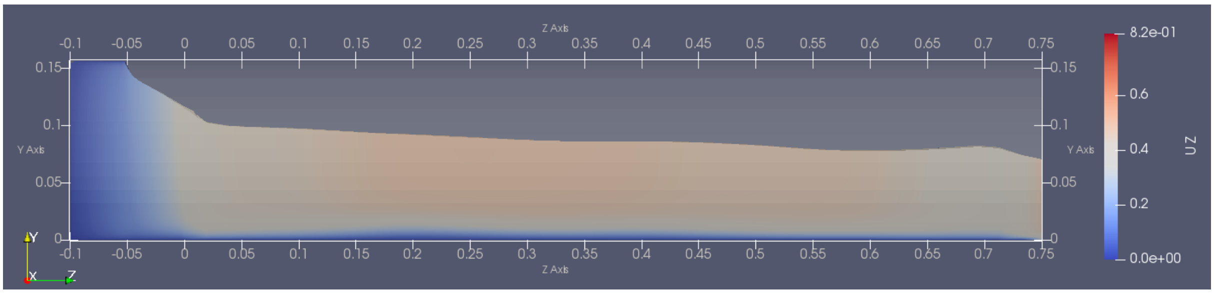

Figure 6.29: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4b at t = 50s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

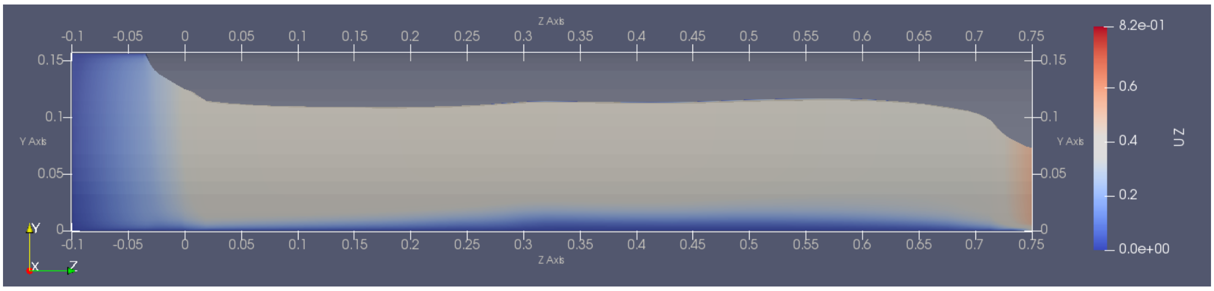

Figure 6.30: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4b at t = 100s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

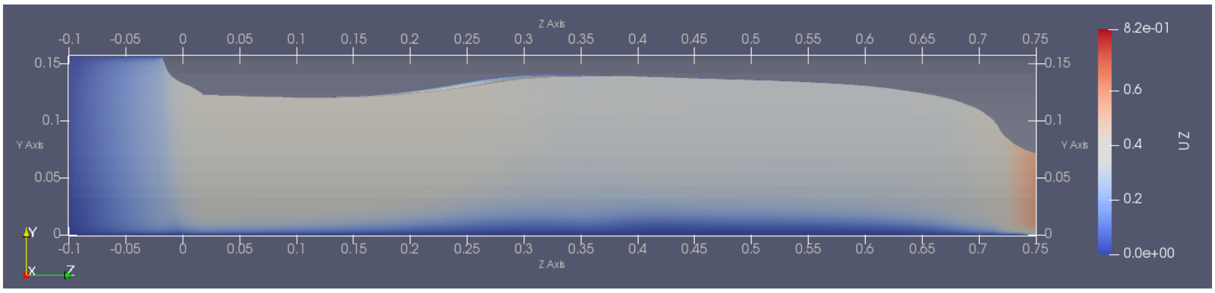

Figure 6.31: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4b at t = 150s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

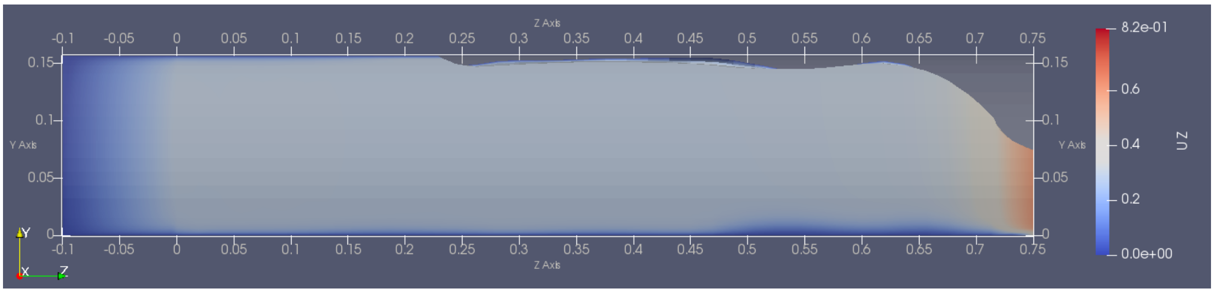

Figure 6.32: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4b at t = 200s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

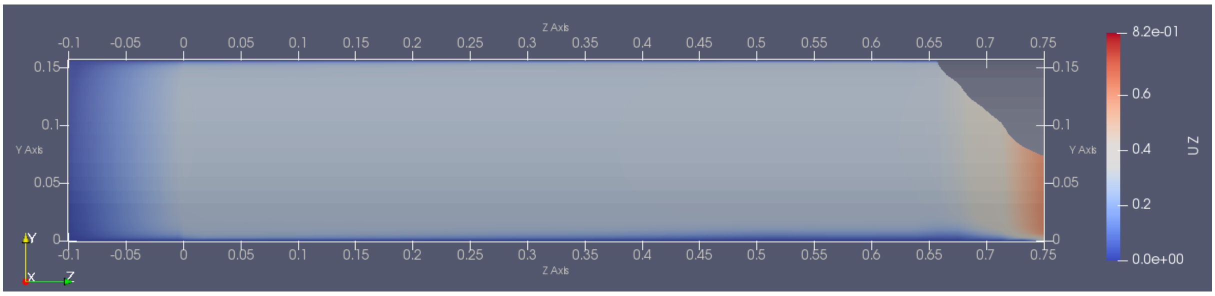

Figure 6.33: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4b at t = 250s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

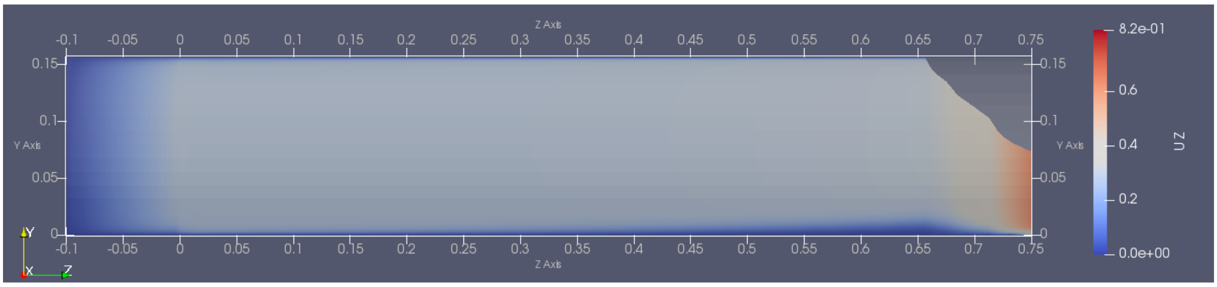

Figure 6.34: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4b at t = 500s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

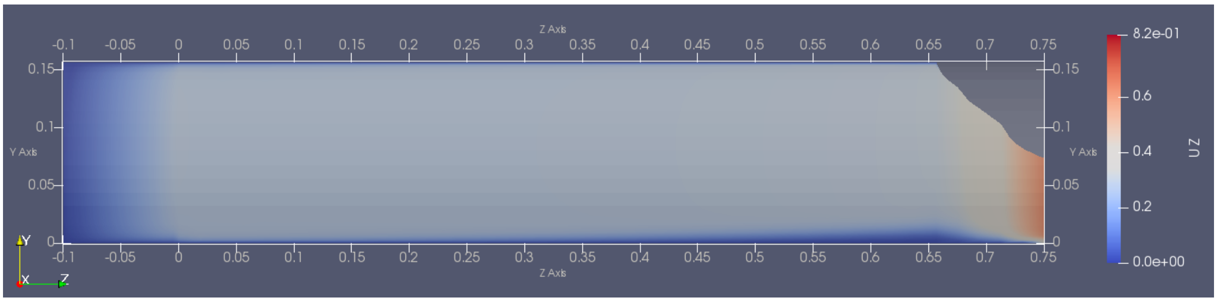

Figure 6.35: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4b at t = 825s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

6.4.4.4. Results simulation 4c

Simulation 4c has been stopped at 427s, when we noticed that the pipe fully filled up in this simulation too.

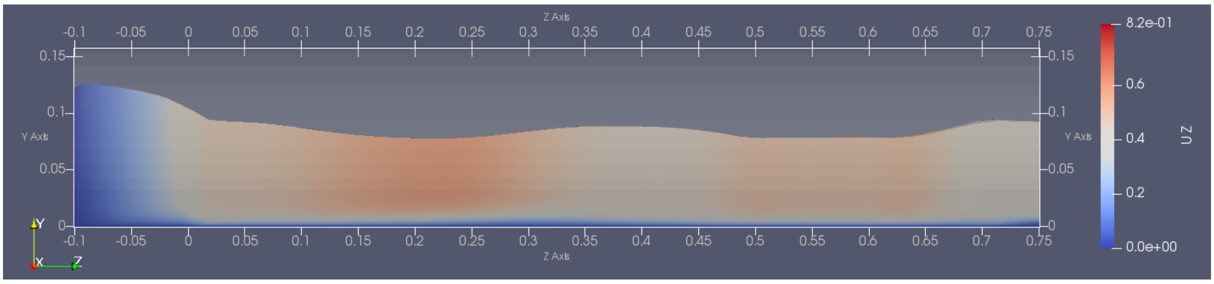

Figure 6.36: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4c at t = 50s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

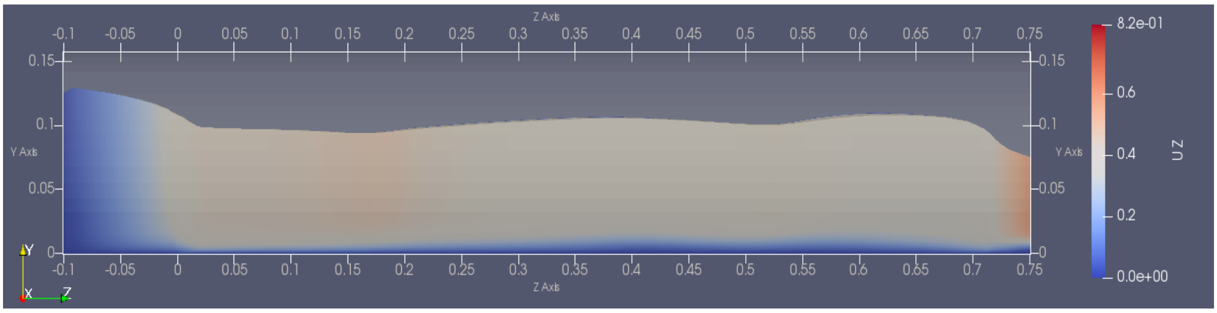

Figure 6.37: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4c at t = 100s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

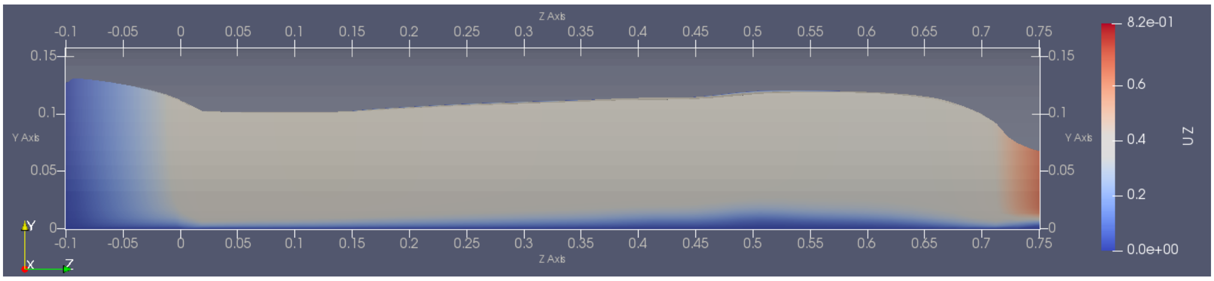

Figure 6.38: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4c at t = 150s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

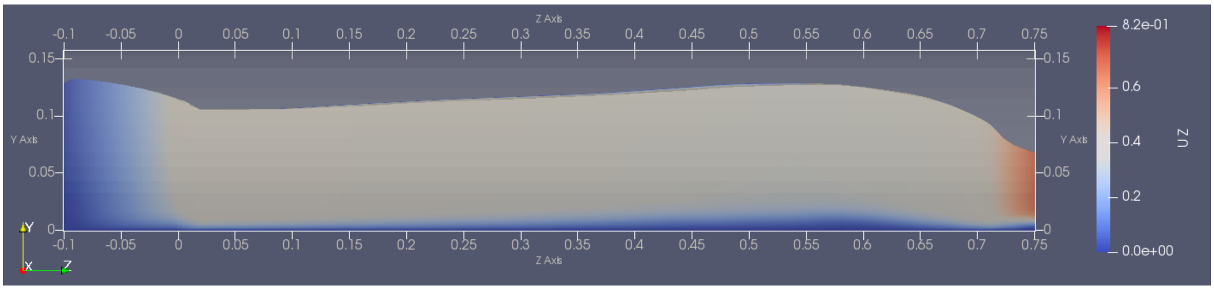

Figure 6.39: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4c at t = 200s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

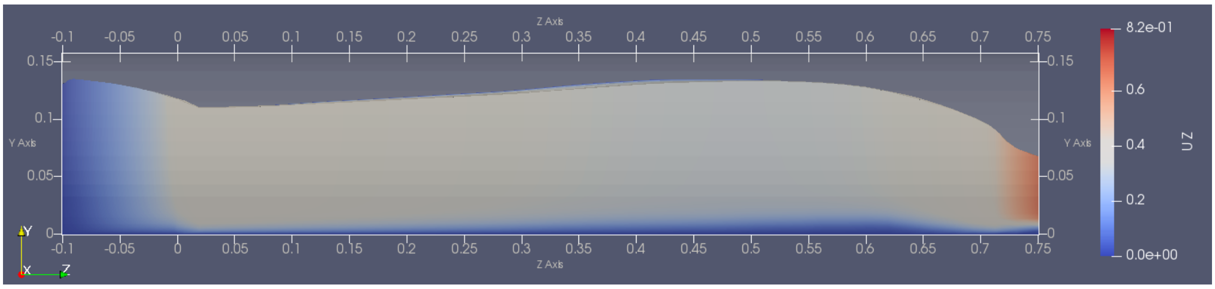

Figure 6.40: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4c at t = 250s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

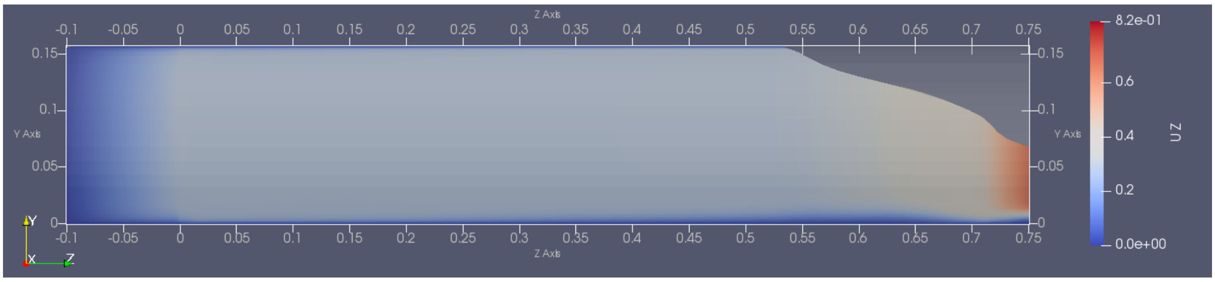

Figure 6.41: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 4c at t = 427s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

At t = 250sec, the pipe hasn't been filled to the top just yet. But the flowdepth is still quite deep. We also see the velocity at outlet patch influences the flow field in the pipe. This is an unexpected result as the outlet of the pipe should have been configured such that it allows free outflow (zeroGradient).

When comparing to the results of simulation 4b, we do see that it now takes longer for the entire pipe to fill up. This was to be expected as the density of our fluid was set at a higher value.

The sand concentration field `csand` is shown in Figure 6.42.

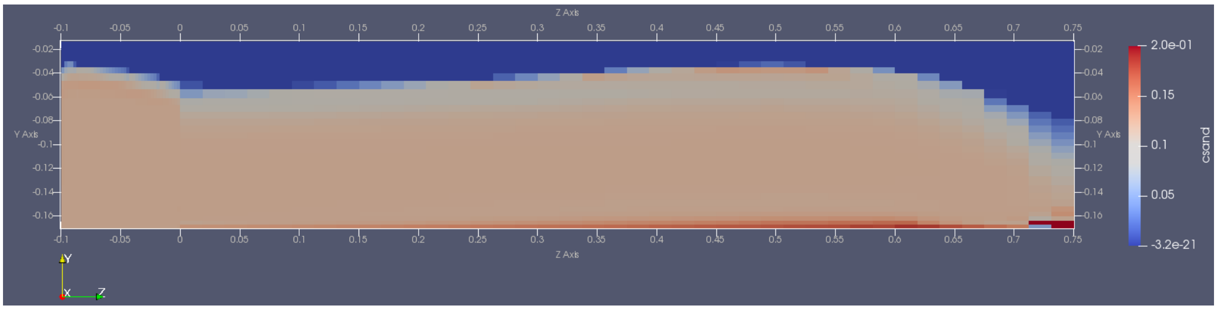

Figure 6.42 `csand` for simulation 4c at t = 250s. Note: z-axis scaled by factor 0.05

We can see that the sand particles are all throughout the mixture. At the bottom of the pipe, a bed begins to form.

6.4.5. Conclusions

In section 6.4 we have simulated with a non-Newtonian flow through a 3D pipe including sand particle transport and settling. The results have shown that in those simulations the pipe tends to completely fill up with the mixture. The force driving the flow in these simulations has only been the gravity exerted on the fluid due to the angle on the pipe. (Spelay, 2007) did not note the pipe would fill up to the top in his experiments. In that sense our results differ.

The sand particles in the simulations up to thus far have not had any influence on the density of the mixture. Thus the density, and by proxy the driving force, can be assumed to have been underestimated when comparing to the experiments. It's not unthinkable that this lack of driving force on the flow is the cause of the pipe fully filling up. After all, in reality, the presence of sand particles is influencing the viscosity of the mixture, and therefore limiting the flow, but the effect the sand particles have on the driving force is not taken into account in the simulation.

6.5. Simulation 5

As was concluded in simulation 4, it's not unthinkable the lack of driving force is causing our pipe to be completely filled with the injected mixture. It's been concluded that due to the sand particles are not being taken into account in the mixture density, this leads to an underestimation of the density and driving force, which results in the pipe fully filling up.

The goal of simulation 5 is to see if the added weight of the sand particles can be taken into account to prevent the behaviour we saw in simulation 4. In other words: will increasing the driving force prevent that tendency of the pipe to fully fill up?

Many attempts have been made at this. Overall, a lot of the simulations have crashed, many of the causes remain unknown. The lack of debugging tooling in OpenFOAM contribute to this very strongly. Many other simulations still showed the pipe completely filling up with the mixture as was also found in Simulation 4. Besides simulation 5a, 5b, and 5c, many more attempts have been made. Not all have been (fully) logged in favour of brevity.

Nonetheless, this section gives an overview of the setup of some simulations that have been performed.

6.5.1. Geometry

The geometry used is quite similar to the geometry of the pipe in simulation 4: a 3D pipe section is used. For simulation 4, it has a length L = 15m (z-direction), and diameter D = 0.1567m. So, as far as the length and diameter are concerned, this is the same as in simulation 4.

At z = 0 an inlet section has been created. This inlet is parallel to the xy-plane to facilitate inflow in z-direction. Only a section of the diameter of the pipe has been set as the inlet patch. The top edge of this inlet patch is situated at y = 0.0805.

Figure 6.43 and Figure 6.44 show the geometry of the pipe. On the left hand side we can see the inlet patch coloured in dark-blue. The red patch is the pipeWall. The light-blue patch is also a wall boundary, named leftWall. And on the right hand side, the orange patch is the outlet of the pipe.

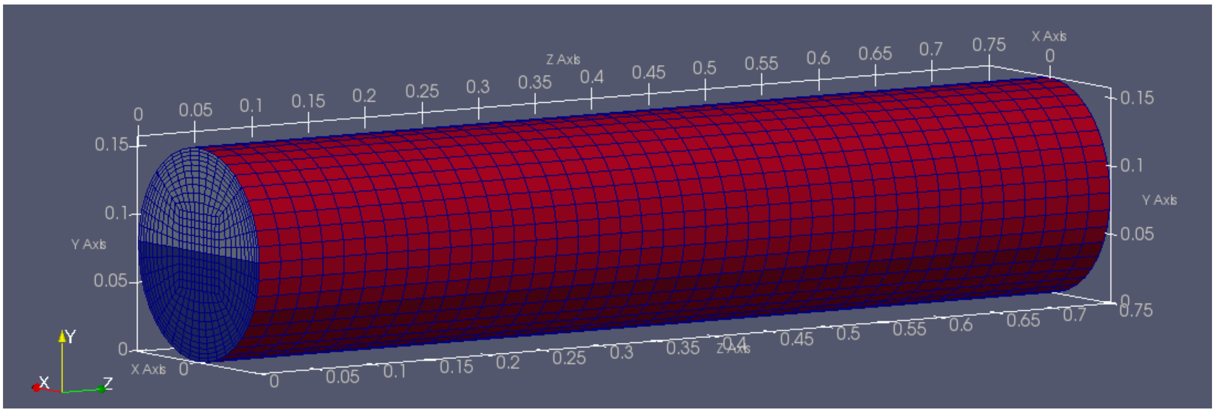

Figure 6.43: Overview of geometry for simulation 5, focus on leftWall and inlet. Note: z-axis scaled by factor 0.05

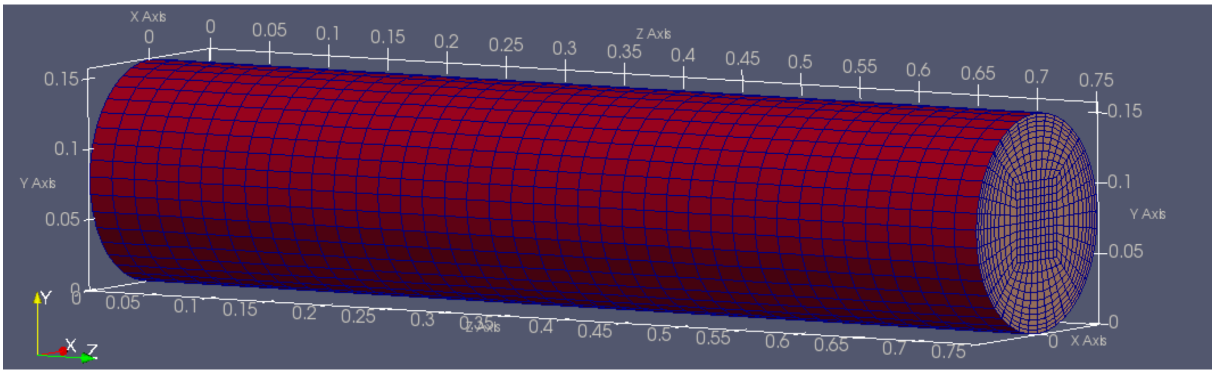

Figure 6.44: Overview of geometry for simulation 5, focus on outlet. Note: z-axis scaled by factor 0.05

Figure 6.45 shows how the mesh of the pipe has been defined. This mesh is uniform along the length of the pipe (in the z-direction) and is divided into 40 cells. From the pipe centre outward, `simpleGrading` 3,1,1 has been applied. This is the same as the grid used in simulation 4. Same as in simulation 4, the mesh is not entirely symmetrical along the midline. More information on that can be found in section 6.4.1.

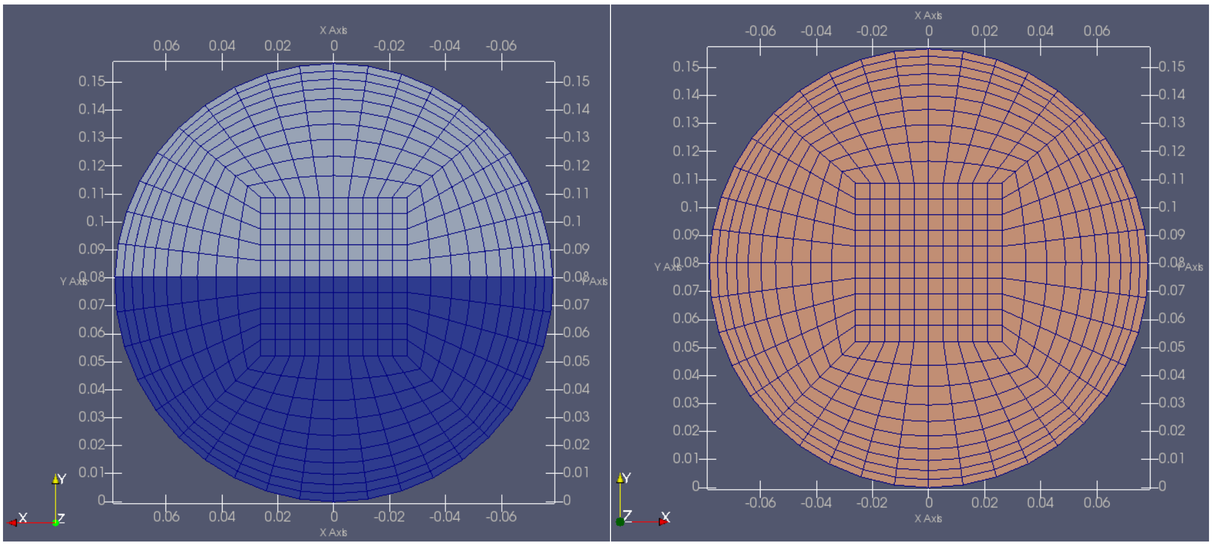

Figure 6.45: Mesh profile in xy-plane for simulation 5. Inlet shown in darkblue and leftWall in lightblue shown on the left and outlet shown in orange on the right

6.5.2. Boundary conditions and driving force

	alpha.water	csand	p_rgh	U	Us	wsvol
inlet	fixedValue uniform 1	codedFixedValue (ramp)	fixedFluxPressure	flowRateInletVelocity constant 0.005		fixedValue uniform (0 0 0)
leftWall	zeroGradient			noSlip		
outlet	inletOutlet (inletValue uniform 0)		prghPressure reference p: uniform 0	pressureInletOutletVelocity		zeroGradient
pipeWall	zeroGradient			noSlip		

Table 6.10: Boundary condition types used in simulation 5

Initially, the entire domain is at rest. At the inlet, a flow of Q = 5l/s enters the domain at t = 0s. Again, the only driving force for the flow is the gravitational force. The inclination angle of the pipe has been varied throughout the simulations. The same vector decomposition method is applied as was used in the previous simulations. The value for `csand` at the inlet is ramped up over time, instead of being stepped at t = 0. From t = 0 to t = 100s it linearly ramps up from 0 to 0.154. Table 6.11 shows these input parameters.

	Simulation 5a	**Simulation 5b**	**Simulation 5c**
Q [l/s]	5	5	5
csand at inlet [-]	0 - 0.154 (linear ramp from t = 0 to t = 100s)	0 - 0.154 (linear ramp from t = 0 to t = 100s)	0 - 0.154 (linear ramp from t = 0 to t = 100s)
θ [°]	5.4	6.4	7.4
g [m/s^2]	9.81	9.81	9.81
g_x [m/s^2]	0	0	0
g_y [m/s^2]	-9.76646	-9.74886	-9.72829
g_z [m/s^2]	0.92320	1.09351	1.26348

Table 6.11: Inlet flow rate, sand fraction `csand` at the inlet and gravitational vector decomposition for simulation 5

6.5.3. Material properties and solver parameters

Again, this simulation utilizes two phases so there are two sets of material properties. The first phase, air, is configured following the same settings as for simulation 1 (see section 6.1.4).

Similar to simulation 4, for the second phase, a Bingham Plastic model has been chosen. Again, yield stress τ_y = 47.3 Pa and plastic viscosity μ_p = 0.0214 Pa.s. In the implementation, those two parameters are first divided by density ρ_{cf}. Table 6.12 shows these.

	Simulation 5a, 5b and 5c
transportModel	Talmon
ρ_{cf} [kg/m^3]	1303
coef m [-]	50
cmax [-]	0.6
alpha0 [-]	0.27
tau0 [m^2/s^2]	0.036301
nu0 [m^2/s]	1.64236e-5
numax [m^2/s]	1000e-2

Table 6.12: Material properties in the `tranportProperties` dictionary for simulation 5

It should be noted at this point that the effective density of the mixture (carried fluid + density of sand particles) is calculated and used in the momentum equation for the mixture. This is density ρ_{mix} instead of the density ρ_{cf}. This is done following equation (6.2).

$$\rho = csand * \rho_s + (1 - csand) * \rho_{cf} \tag{6.2}$$

At the inlet, when a fraction of 0.154 for sand particles `csand` is injected, the density of the combined mixture is 1510 kg/m^3.

The simulation in this section used the implementation as described in section 5.6.3.2.

The full simulation case input files can be found in appendix D.4.

6.5.4. Results

Figure 6.46 through Figure 6.59 show the results for simulation 5. These figures have been created in the same manner as for simulation 4. This is explained in section 6.4.4.1 and not repeated here.

6.5.4.1. Results simulation 5a

For simulation 5a, we let the simulation run all the way until t = 600s. We then see the pipe fully filled up with the mixture. We also see the velocity at outlet patch influences the flow field in the pipe. This was also noted in simulation 4.

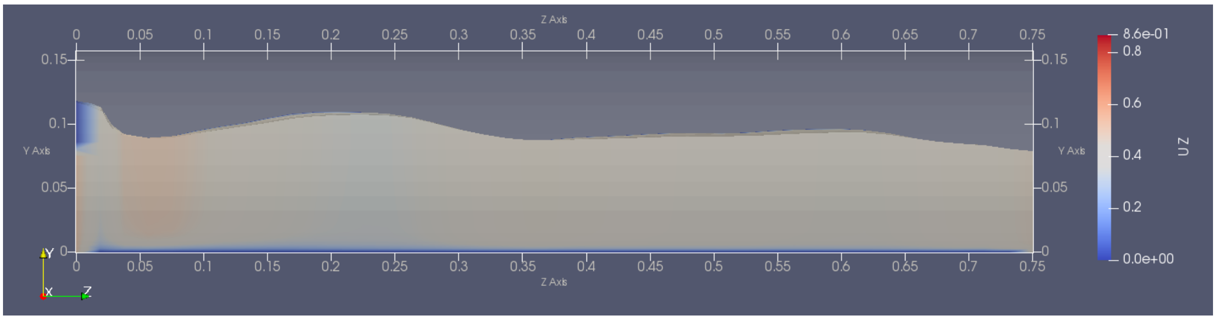

Figure 6.46: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5a at t = 50s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

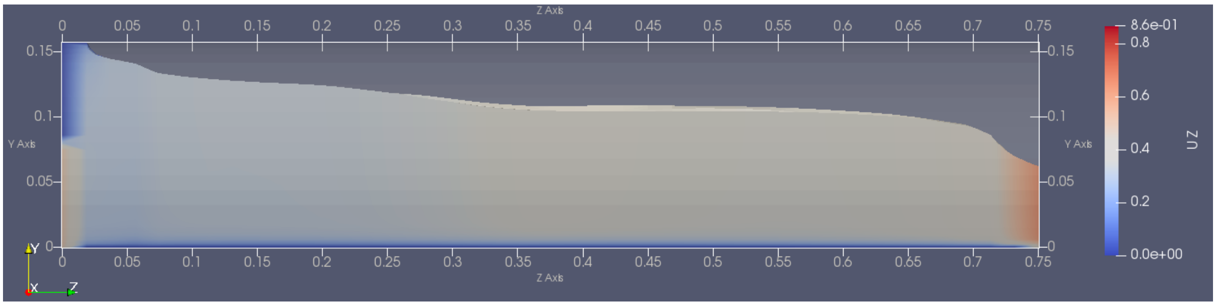

Figure 6.47: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5a at t = 100s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

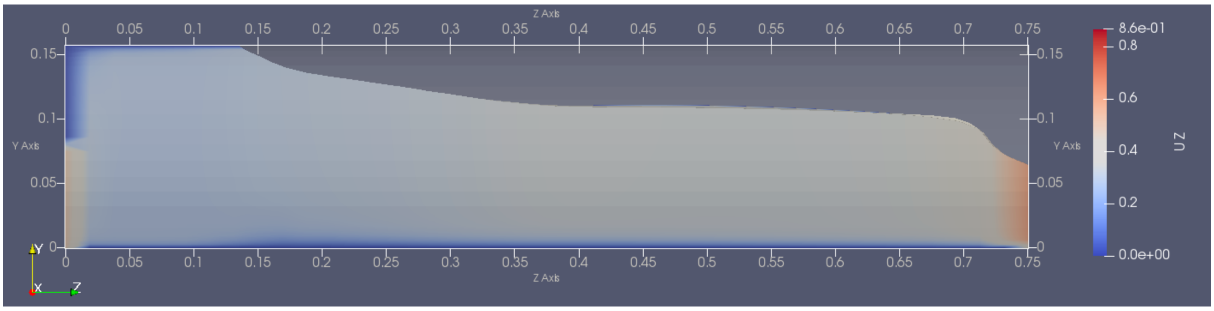

Figure 6.48: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5a at t = 123s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

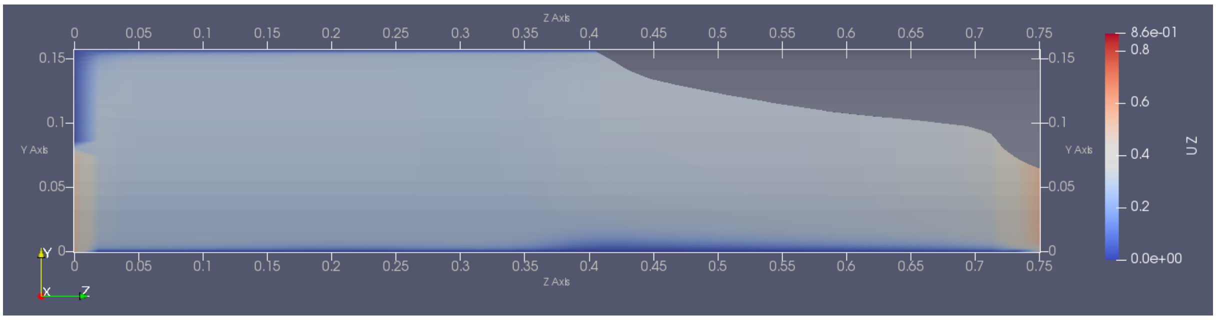

Figure 6.49: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5a at t = 150s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

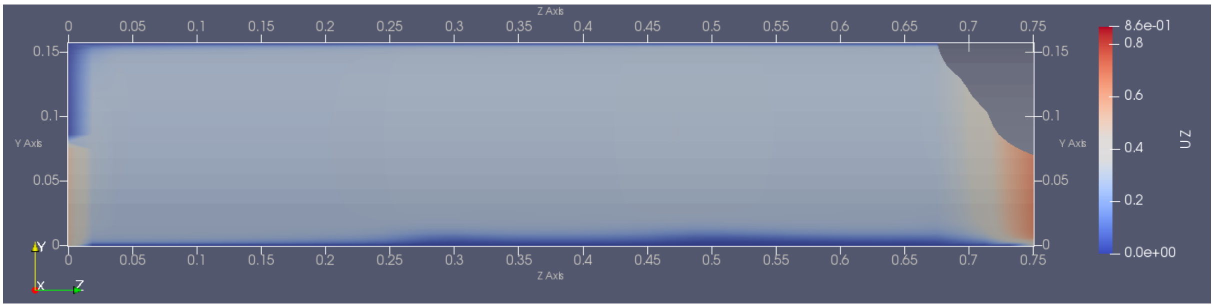

Figure 6.50: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5a at t = 220s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

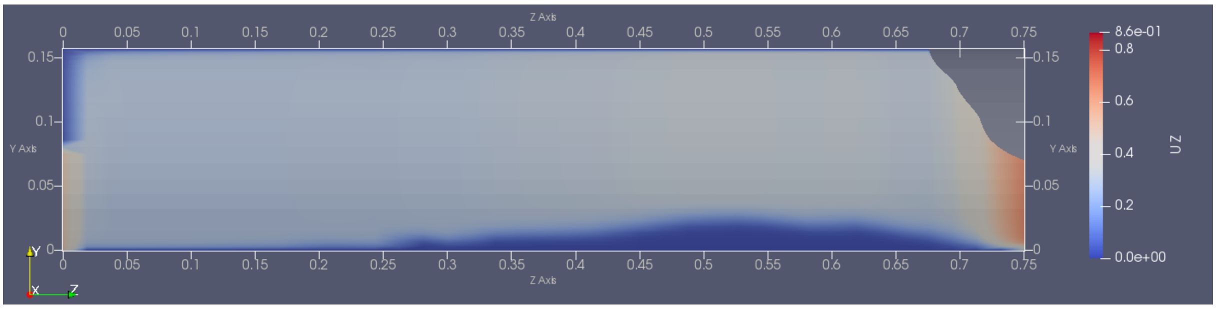

Figure 6.51: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5a at t = 600s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

6.5.4.2. Results simulation 5b

To see if we can actually simulate a pipe that's not filling up, we tilt the gravitational force to a greater angle and try it again. This did not seem to help. In simulation 5b, the pipe still fills up all the way to the top. We let it run to 221s and that's when the solver crashed.

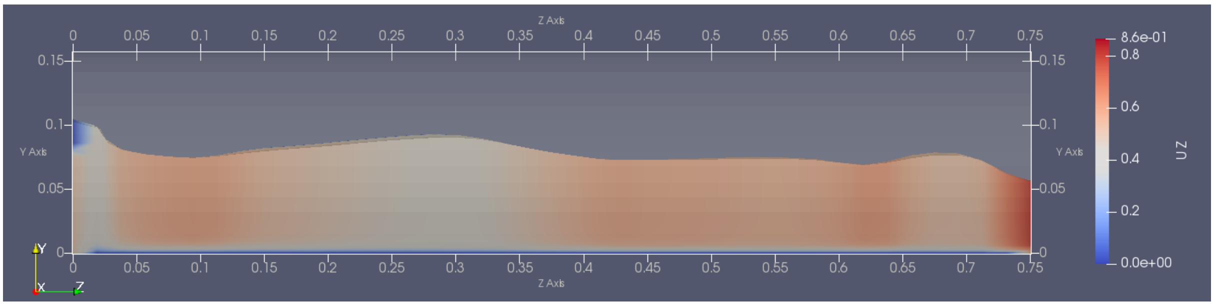

Figure 6.52: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5b at t = 50s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

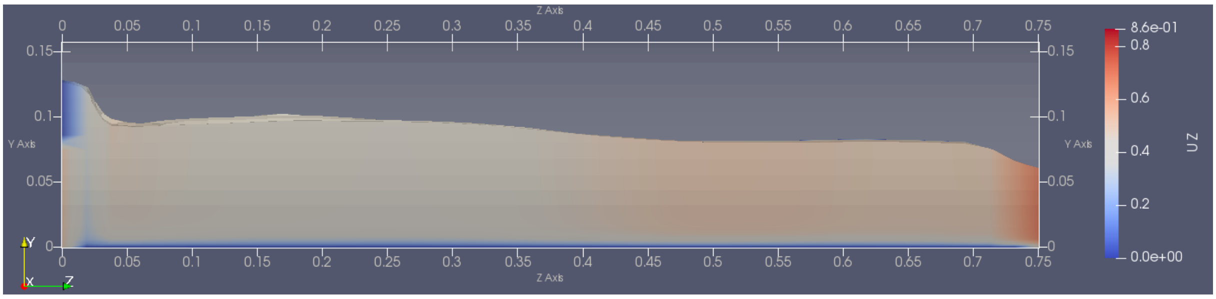

Figure 6.53: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5b at t = 100s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

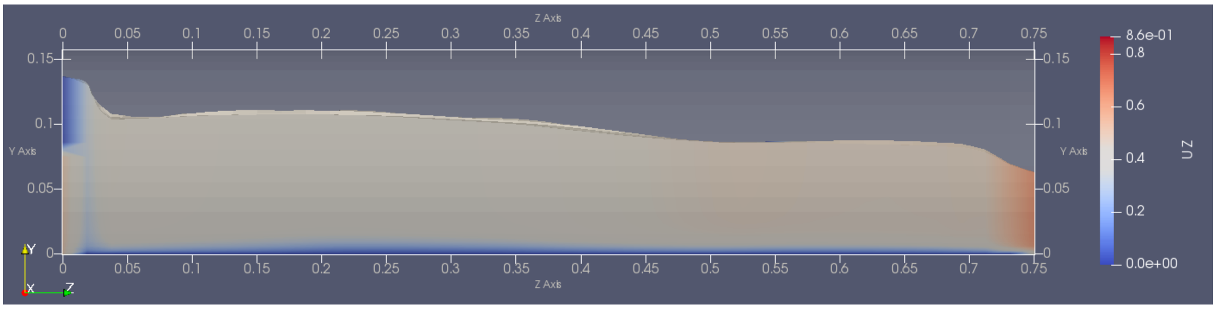

Figure 6.54: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5b at t = 123s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

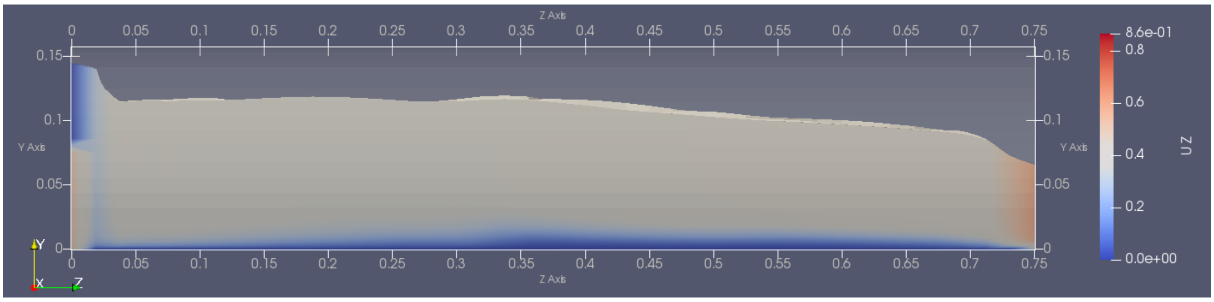

Figure 6.55: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5b at t = 150s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

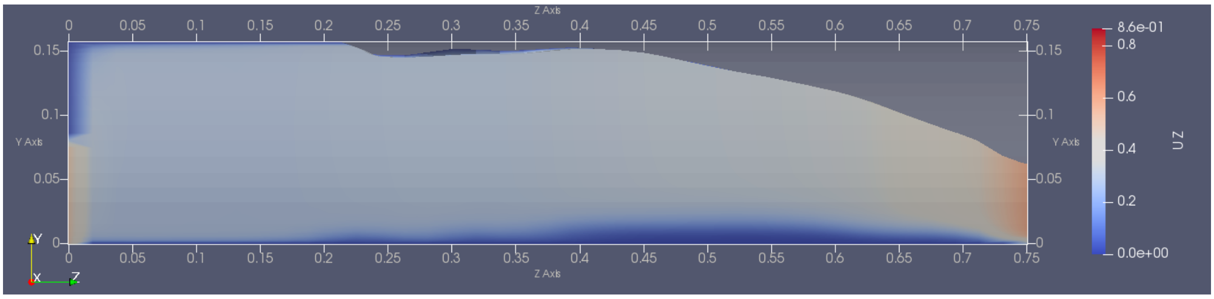

Figure 6.56: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5b at t = 220s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

6.5.4.3. Results simulation 5c

To see if the pipe would not fill up if we increase the angle even further. Now, the solver crashes after 123s.

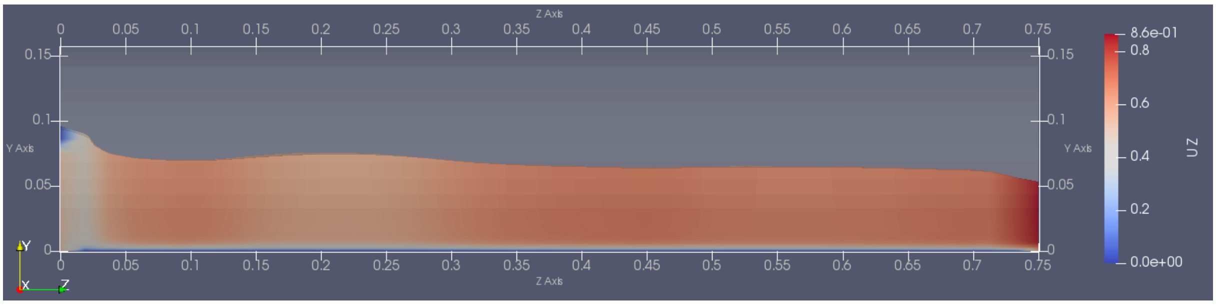

Figure 6.57: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5c at t = 50s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

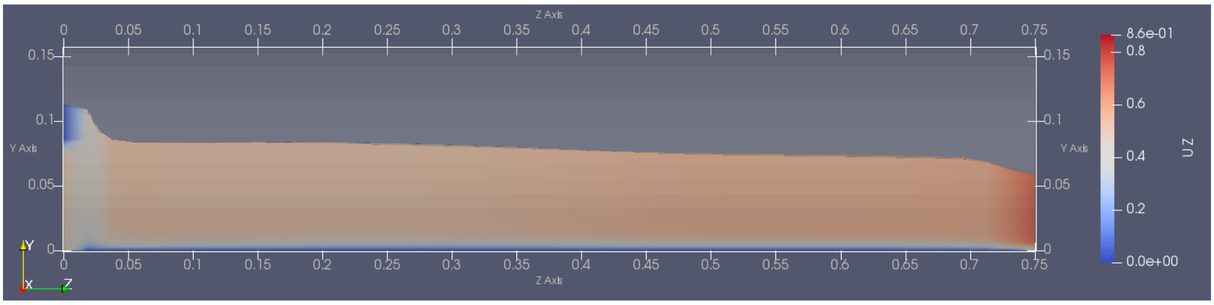

Figure 6.58: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5c at t = 100s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

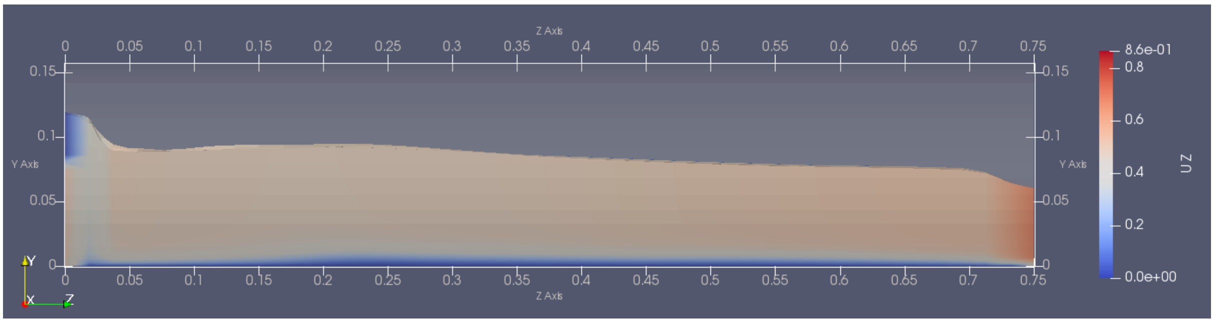

Figure 6.59: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 5c at t = 123s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

Simulation 5c crashed at t = 123s. We were not able to find the reason why at this point in time

6.5.5. Conclusions

In simulation 5, we have simulated a non-Newtonian flow through a 3D pipe including sand particle transport and settling. The presence of sand particles in these simulations were taken into account in the density of the mixture. The force driving the flow is now also including the sand particles. Still, our pipe fully fills up (simulation 5a and 5b). Further, simulation 5b and 5c both crashed.

6.5.6. Next steps

At this point in time, we started experimenting more. All resulting simulations have either crashed or the solver's timestep became very small. To provide insight into what was tried, we tried:

- Turning on the momentumPredictor in PIMPLE
- Ramping the inlet velocity of the mixture (besides only ramping the sand particle concentration)
- Using a dynamically adjustable timestep
- Using prghTotalPressure instead of prghPressure on the outlet
- Using PIMPLE with 250 outerCorrectors (nOuterCorrectors)
- Relaxing the pressure and velocity fields
- Rearranging the calculation of csand with the pimple pressure corrector loop
- Removing the capped exponents calculation (section 5.6.3.2)
- Reducing tolerances on solvers
- Using different matrix solvers

At this point, there are multiple things not going as expected. Either the pipe fills up with the mixture, or the simulation crashes, or the solver's timestep becomes very small. We don't understand what's causing the velocity to increase right before the outlet of the pipe. And we also don't understand what is causing these crashes. The lack of debugging tools in OpenFOAM is also preventing us from diving into the actual issue and pinpointing what's wrong.

Cees van Rhee brought forward two ideas:

1. Use the leftWall patch of simulation 5 and use it to pump in air. The idea here is to see if this alleviates the trouble we've been seeing with all the pipe simulations. The simulations we ran with a 2D open channel (simulation 1 and 2) did not crash. One of the differences is that that mesh allowed for an atmosphere to be applied, besides having an outlet section. The pipe we've been simulating with was a pipe section with only an inlet and an outlet. As a result, either the pipe inlet or outlet were chosen as atmosphere and reference pressure. Simulation 6 goes into this idea.
2. We can also change our mesh such that it resembles simulation 1 and 2 a little closer. If we do this, we will need to make sure we can apply an atmosphere and reference pressure in the vertical direction on top of our pipe. A half pipe would allow for this. To prevent spill-over if the fluid reaches too high flowdepths, we can extrude the walls of the pipe upwards. So, we'd basically be using a half-pipe mesh where the pipe is extruded vertically upwards creating a rectangular block on top. This would then be very similar simulation 1 and 2, except it's a 3D instead of 2D simulation and the bottom is round instead of flat. Simulation 7 goes into this idea.

6.6. Simulation 6

Cees van Rhee brought forward an idea to use the leftWall patch of simulation 5 and use it to pump in air. The idea here is to see if this alleviates the trouble we've been seeing with all the pipe simulations. The simulations we ran with a 2D open channel (simulation 1 and 2) did not crash. One of the differences is that that mesh allowed for an atmosphere to be applied, besides having an outlet section. The pipe we've been simulating with was a pipe section with only an inlet and an outlet. As a result, either the pipe inlet or outlet were chosen as atmosphere and reference pressure.

In simulation 6, we try to pump in additional air just above the fluid inlet. We now basically get 2 inlets, 1 inlet for the Bingham Plastic and sand mixture, and 1 inlet for the air.

6.6.1. Geometry

The geometry and mesh of the simulation domain is the identical to that of simulation 5, details are noted down in section 6.4.1.

6.6.2. Boundary conditions and driving force

	`alpha.water`	`csand`	`p_rgh`	`U`	`Us`	`wsvol`
`inlet`	fixedValue uniform 1	fixedValue uniform 0	fixedFluxPressure	flowRateInletVelocity constant 0.005		fixedValue uniform (0 0 0)
`leftWall`	fixedValue uniform 0		prghTotalPressure reference p0: uniform	flowRateInletVelocity constant 0.005		fixedValue uniform (0 0 0)
`outlet`	zeroGradient		fixedFluxPressure	zeroGradient		
`pipeWall`	zeroGradient		fixedFluxPressure	noSlip		

Table 6.13: Boundary condition types used in simulation 6

Initially, the entire domain is at rest. At the inlet, a flow of Q = 5 l/s enters the domain through the inlet patch at t = 0s. The only driving force for the flow will be a gravitational force. The value for `csand` at the inlet was set to 0. This is done to make this simulation a little easier. This allows us to see if the simulation runs without crashing before we add sand into the equation.

The only driving force for the flow will be a gravitational force. Table 6.14 shows the angle of inclination and the vector decomposition.

$\boldsymbol{g}$ **(m/s^2)**	**9.81**
$\boldsymbol{\theta}$ **(°)**	5.4
$\boldsymbol{g_x}$ **(m/s^2)**	0
$\boldsymbol{g_y}$ **(m/s^2)**	-9.76646
$\boldsymbol{g_z}$ **(m/s^2)**	0.92320

Table 6.14: Gravitational vector decomposition for simulation 6

6.6.3. Material properties and solver parameters

Again, this simulation utilizes two phases so there are two sets of material properties. The first phase, air, is configured following the same settings as for simulation 1 (see section 6.1.4).

The second phase is the Bingham Plastic viscosity model that has been used before as well. Again, yield stress τ_y = 47.3 Pa and plastic viscosity μ_p = 0.0214 Pa.s. In the implementation, those two parameters are first divided by density ρ_{cf}. Table 6.15 shows all material parameters used.

It should be noted at this point that the effective density of the mixture (carried fluid + density of sand particles) is calculated and used in the momentum equation for the mixture. This is density ρ_{mix} instead of the density ρ_{cf}. At the inlet, when a fraction of 0.154 for sand particles `csand` is injected, the density of the combined mixture is 1510 kg/m^3.

	Simulation 6
transportModel	Talmon
ρ_{cf} [kg/m^3]	1303
coef m [-]	50
cmax [-]	0.6
alpha0 [-]	0.27
tau0 [m^2/s^2]	0.036301
nu0 [m^2/s]	1.64236e-5
numax [m^2/s]	1000e-2

Table 6.15: Material properties in the `tranportProperties` dictionary for simulation 6

The simulation in this section used the implementation as described in section 5.6.3.2.

The full simulation case input files can be found in appendix D.5.

6.6.4. Results

Figure 6.60 through Figure 6.65 show the results for simulation 6. These figures have been created in the same manner as for simulation 4 and 5. This is explained in section 6.4.4.1 and not repeated here.

We can see that the flow starts out as expected. However, between 16 and 17 seconds, the solver starts running into very high values for the Courant number. We can see this in figure Figure 6.66. We can also see in Figure 6.64 (t = 17s) that the velocities have suddenly increased in some places when comparing to the velocities at 16s (Figure 6.63). After t = 17.475s, the simulation crashes completely.

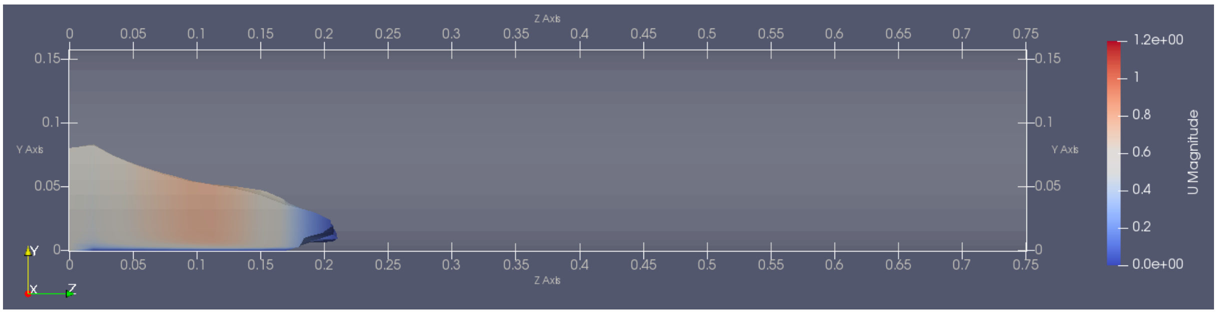

Figure 6.60: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 6 at t = 5s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

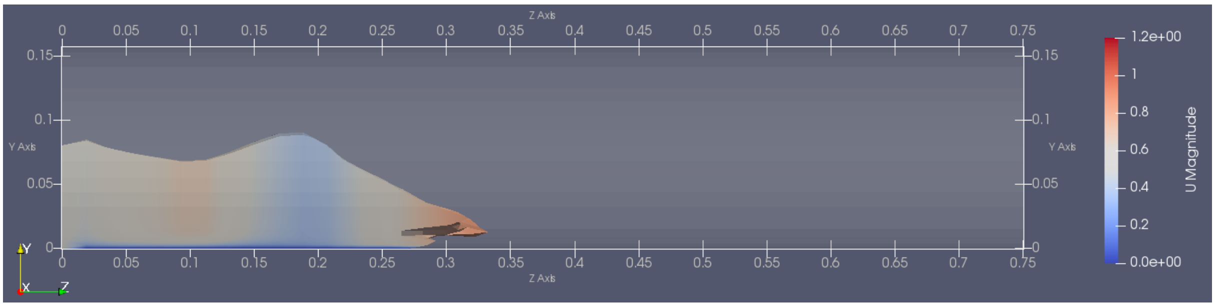

Figure 6.61: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 6 at t = 10s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

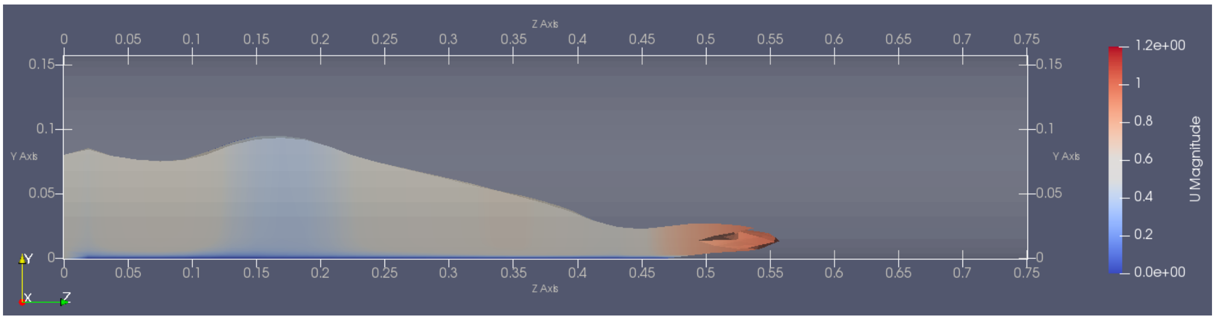

Figure 6.62: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 6 at t = 15s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

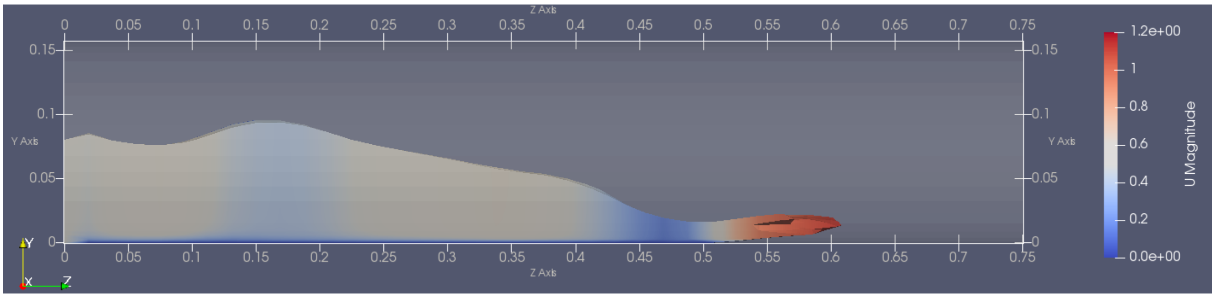

Figure 6.63: Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 6 at t = 16s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

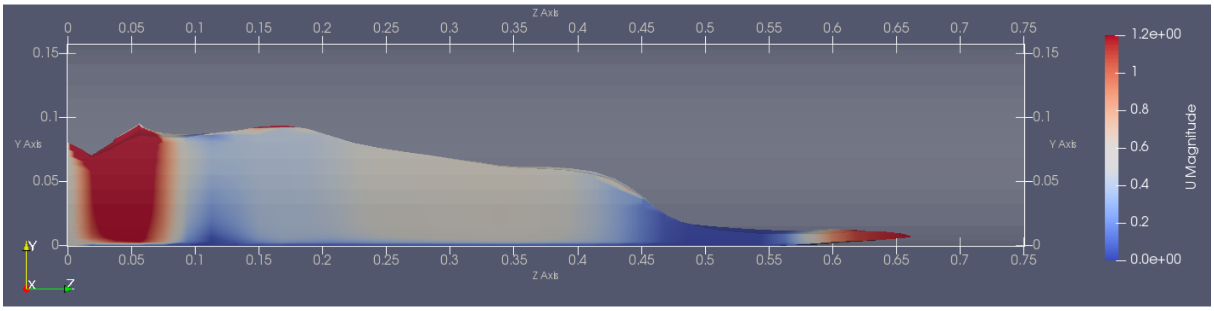

Figure 6.64 Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 6 at t = 17s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

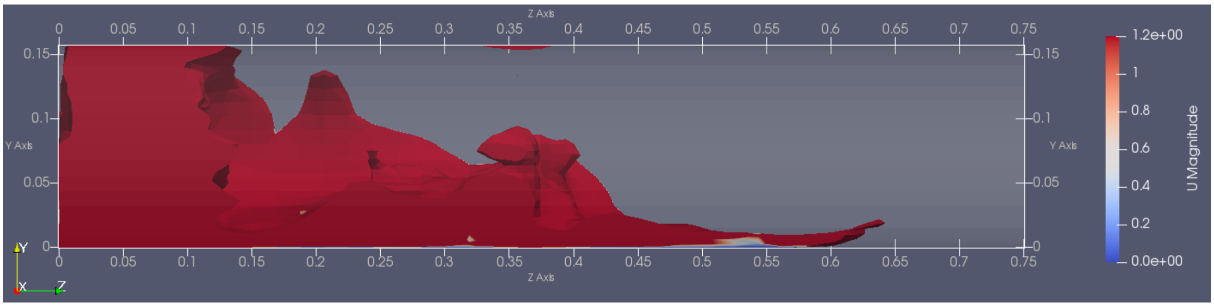

Figure 6.65 Velocity in z-direction, Uz, for the Bingham Plastic fluid in simulation 6 at t = 17.3s. In semi-transparent grey we can see the pipe wall. Note: z-axis scaled by factor 0.05

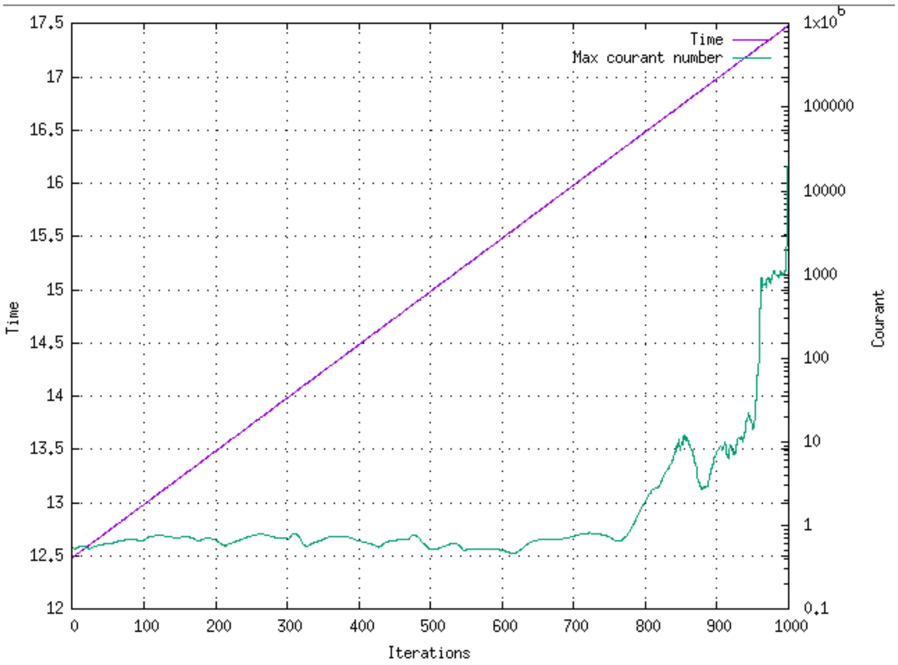

Figure 6.66: graph showing the solver runs into high courant numbers in simulation 6. Only the last 1000 iterations are shown. At around iteration 780 we can see the courant number increase. Iteration index is shown on the x-axis. The left y-axis shows the simulation time. On the x-axis we see the amount of iterations, and the right y-axis shows the maximum courant number

6.6.5. Conclusions

We've seen the simulation run into huge courant numbers after a while and the results are no longer physical.

Applying a dynamic timestep would not have solved this, as in that case the solver would just keep reducing the timestep it takes until it meets the CFL criterion. I suspect it would have never (within reason) recovered from this.

It's still not understood what actually causes this to happen. At this point, I'd rather move over to give the second idea (section 6.5.6) a shot.

6.7. Simulation 7

The previously simulations failing leads us to believe that the problem might be an incompatibility of boundary conditions. The pipe geometry used in simulations 4 and 5 was a fully enclosed pipe section. It could be that the conditions for atmospheric pressure and inflow/outflow are just not equipped to be placed on either the outlet or inlet of the pipe section.

Therefore, we devise a new grid for a new simulation. Much like the 2D open channel case, the grid used in this simulation will just have a flat lid. The grid of the pipe used in simulations 4 and 5 is extruded upwards to essentially create a half-pipe with a rectangular block on top of it. This will allow us to set an atmospheric boundary conditions from the top of the grid, like has been done in simulation 1 and 2

6.7.1. Geometry

The geometry is a little more complex than for previous simulations. This time, we have 2 blocks. Block 1 is a half-pipe with the height equal set to the radius of the pipe. Block 2 is a rectangular block and is placed on top of the half-pipe.

The total domain has a length L = 15m (z-direction) and the half-pipe diameter D = 0.1567m. The rectangular block has a height equal to the radius of the pipe. Thus, the total height is of the domain is equal to the pipe diameter. The inlet is placed at z = 0 in the xy-plane (normal in z-direction). Figure 6.67 shows an overview of the geometry focussed on the inlet end of the pipe. In dark-blue we see the inlet patch, in light-blue a wall patch called leftWall, orange resembles the pipeWall and the red patch shows the atmosphere patch. In Figure 6.68 we can see the outlet side of the pipe, and in beige colour we see the outlet patch.

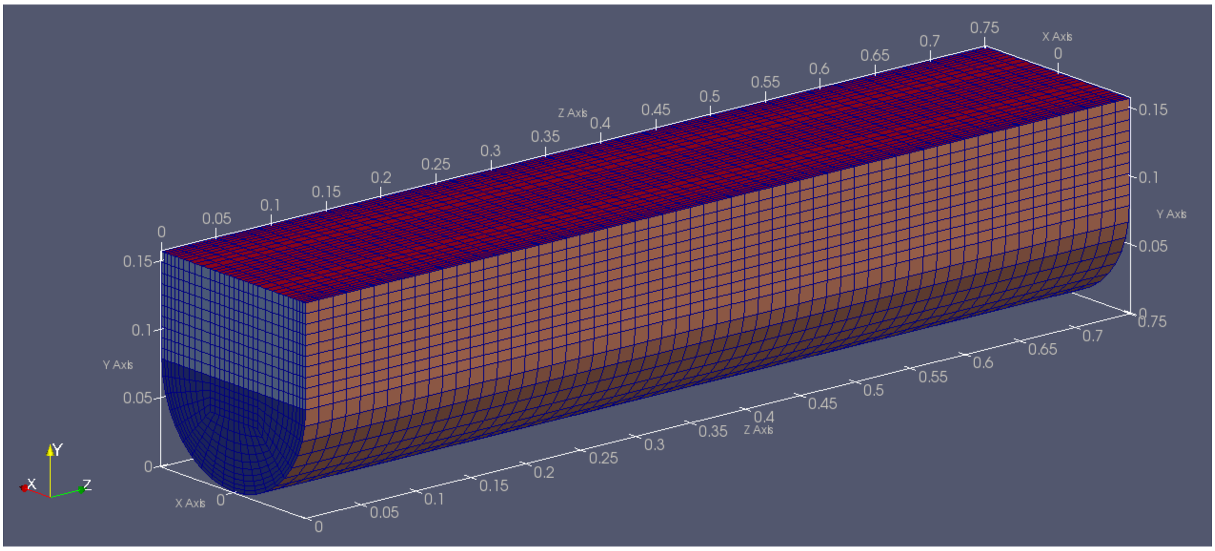

Figure 6.67: Overview of geometry for simulation 7, focus on inlet side of pipe. Note: z-axis scaled by factor 0.05

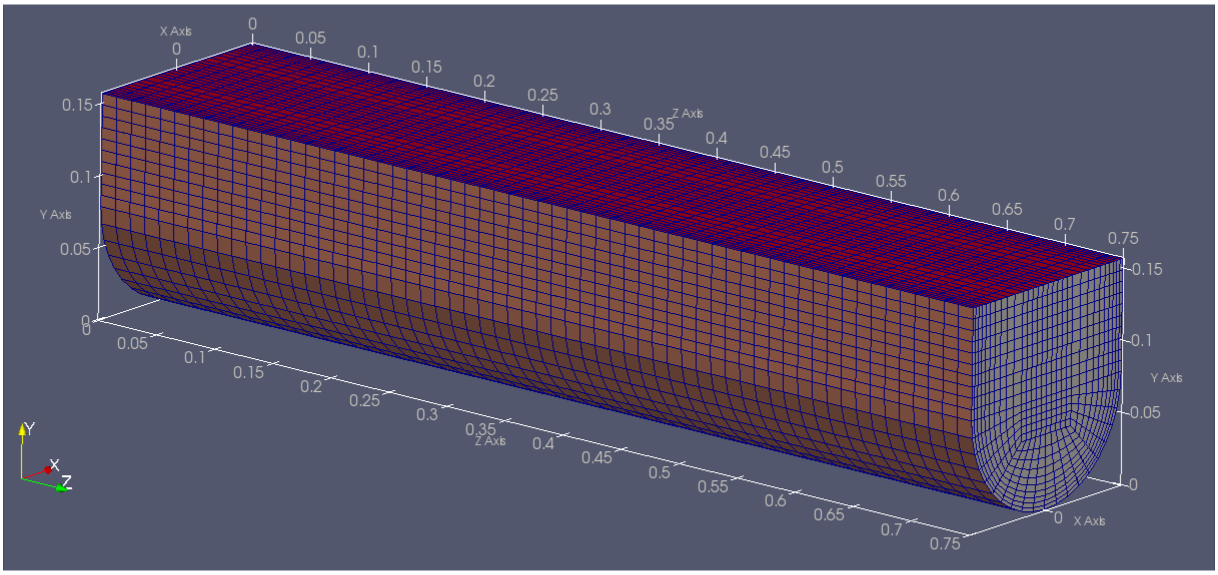

Figure 6.68: Overview of geometry for simulation 7, focus on outlet side of pipe. Note: z-axis scaled by factor 0.05

The mesh profile is uniform along the length of the pipe (z-direction) and in the z-direction it's been divided in 40 cells. This is shown in Figure 6.70. Grading has been applied in the xy-plane from the centre outward to the pipe wall at `simpleGrading` 3, 1, 1. This can be seen in Figure 6.69.

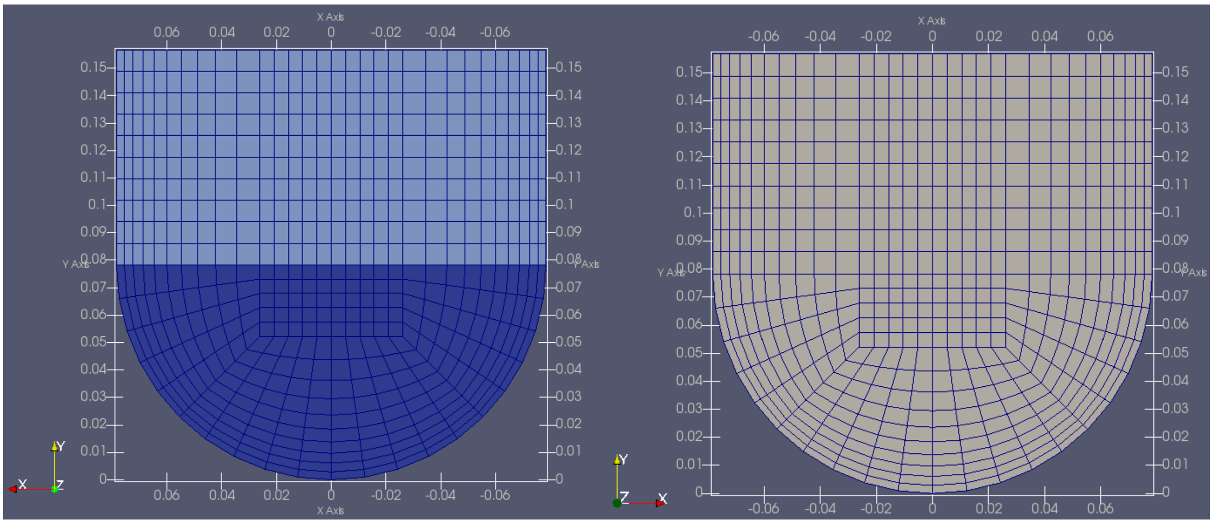

Figure 6.69: Mesh profile in xy-plane for simulation 7. Focus on inlet side of pipe on the left, and on the right hand side focus on the outlet side

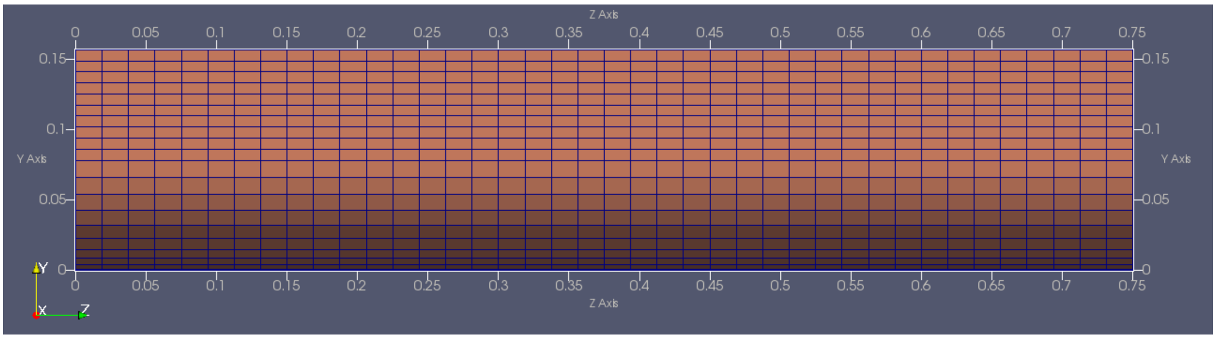

Figure 6.70: Overview of grid along z-axis for simulation 7 discretized into 40 cells along z-axis. Note: z-axis scaled by factor 0.05

6.7.2. Boundary conditions

The following boundary condition types have been defined.

	alpha.water	csand	p_rgh	U	Us	wsvol
inlet	fixedValue uniform 1	fixedValue uniform 0.154	fixedFluxPressure	flowRateInletVelocity constant 0.005		fixedValue uniform (0 0 0)
leftWall	zeroGradient		fixedFluxPressure	noSlip	noSlip	zeroGradient
outlet	zeroGradient		fixedFluxPressure	inletOutlet inletValue: (0 0 0)		zeroGradient
pipeWall	zeroGradient		fixedFluxPressure	noSlip	noSlip	noSlip
atmosphere	inletOutlet (inletValue uniform 0)		prghPressure reference p: uniform 0	pressureInletOutletVelocity value: uniform (0 0 0)		fixedValue uniform (0 0 0)

Table 6.16: Boundary condition types in simulation 7

Initially, the entire domain is at rest. At the inlet, a flow of Q = 5 l/s enters the domain through the inlet patch at t = 0s. The inlet flow rate for `Us` has been set to the same so that it will get the same inlet velocity as the fluid itself does. The value for `csand` at the inlet was set to 0.154.

6.7.3. Driving force

The only driving force for the flow will be a gravitational force. As with previous simulations, actually angling the mesh through blockMesh can result in an indecipherable mesh, it is easier to put the gravitational force vector under an angle and keep the mesh simpler. Components g_x, g_y and g_z are defined using vector decomposition using angle θ. Table 6.17 shows the angle used and the decomposition.

g (m/s^2)	**9.81**
θ (°)	5.4
g_x (m/s^2)	0
g_y (m/s^2)	-9.76646
g_z (m/s^2)	0.92320

Table 6.17: Gravitational force vector decomposition for simulation 7

6.7.4. Material properties and solver parameters

Again, this simulation utilizes two phases so there are two sets of material properties. The first phase, air, is configured following the same settings as for simulation 1 (see section 6.1.4).

For the second phase, a Bingham Plastic viscosity model has been chosen. The yield stress of the carrier fluid $\tau_{y,cf}$ has been set to 47.3 Pa and plastic viscosity of the carrier fluid $\mu_{p,cf}$ at 0.0214 Pa.s. In the implementation in `interFoam`, those input parameters need to first divided by density ρ before being set. Table 6.18 shows the parameters as used in the `tranportProperties` dictionary.

	Simulation 7
transportModel	Talmon
rho [kg/m^3]	1303
coef m [-]	50
cmax [-]	0.6
alpha0 [-]	0.27
tau0 [m^2/s^2]	0.036301
nu0 [m^2/s]	1.64236e-5
numax [m^2/s]	1000e-2

Table 6.18: Material properties in the `tranportProperties` dictionary for simulation 7

It should be noted at this point that the effective density of the mixture is calculated and used in the momentum equation for the mixture. This is density ρ_{mix} instead of the density ρ_{cf}. This is done following equation (6.2). At the inlet, a fraction of 0.154 for sand particles `csand` is injected. This means the density of the combined mixture is 1510 kg/m^3 following equation (3.1).

The full simulation case files are to be found in appendix D.6.

6.7.5. Results

The result of simulation 7 is not satisfactory. It was seen that the timesteps become very small. This is shown in Figure 6.71. We will dive into why we think this happens in section 6.7.6.

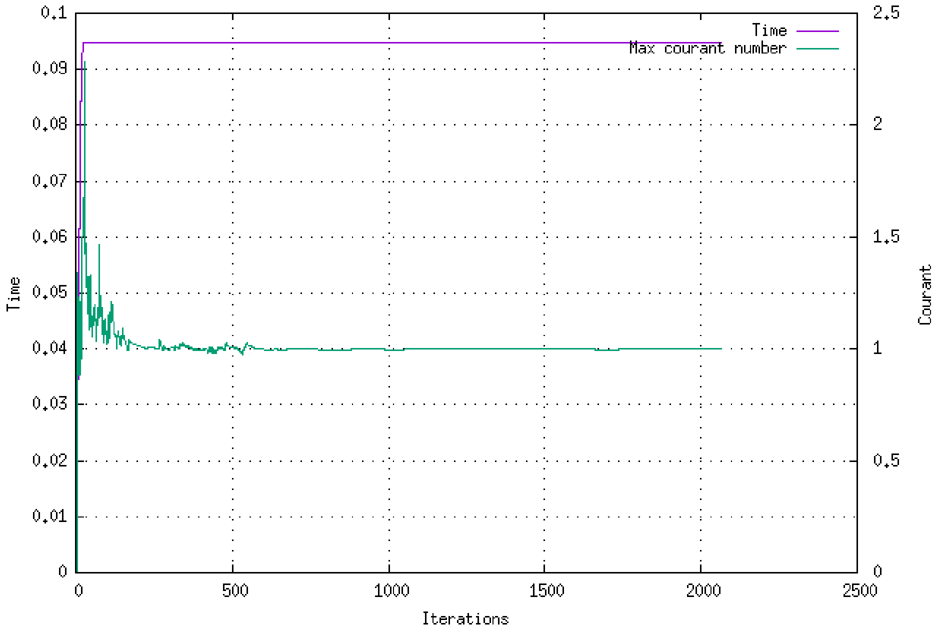

Figure 6.71: Graph showing small timesteps for simulation 7a.

6.7.6. Removing underdetermined cells

When we run a `checkMesh` command on our blockMesh, we actually see that the mesh is evaluated OK. However, it's been discovered that we can also run a more elaborate check following the `checkMesh -allTopology -allGeometry` command. When we run this, we actually see 1 of the checks fail. The below section shows a part of the result of the more elaborate check. We can see that it's found 28 cells that are underdetermined.

```
...
    Cell determinant (wellposedness) : minimum: 0.00050217519 average: 0.019025868
 ***Cells with small determinant (< 0.001) found, number of cells: 28
```

```
  <<Writing 28 under-determined cells to set underdeterminedCells
...
```

We can try to make these cells more determined, or we can remove them from our grid. As a first try, I've removed the cells from my grid. OpenFOAM has some built-in tools that allows us to remove these cells.

The snippet of code below removes all `underdeterminedCells` from a grid. When we run the simulation, it runs well, and doesn't suffer from running into very small timesteps.

```
foamJob -s checkMesh -allTopology -allGeometry
foamJob -s setSet -constant
cellSet temp new cellToCell underdeterminedCells any
cellSet temp invert
cellSet temp subset
foamJob -s subsetMesh temp
```

After only a few timesteps, it still crashes, but the logs show us it crashes right when it tries to calculate the viscosity in our material model. I know why this is and it's already been solved before. This was solved in section 5.6.3.2 before.

6.7.7. Fixed material model

We've enabled the piece of code that's been described in section 5.6.3.2 and re-ran the simulation. What we can now see is that the fluid mixture just seems to leak out of our domain. This happens near our inlet and it's quite likely this is happening because we blatantly removed some cells from our domain. Figure 6.72 shows the velocity in y-direction at $t \cong$ 14.6s. We can see that some cells have been removed from the domain, and at that point in the grid we also see higher velocities.

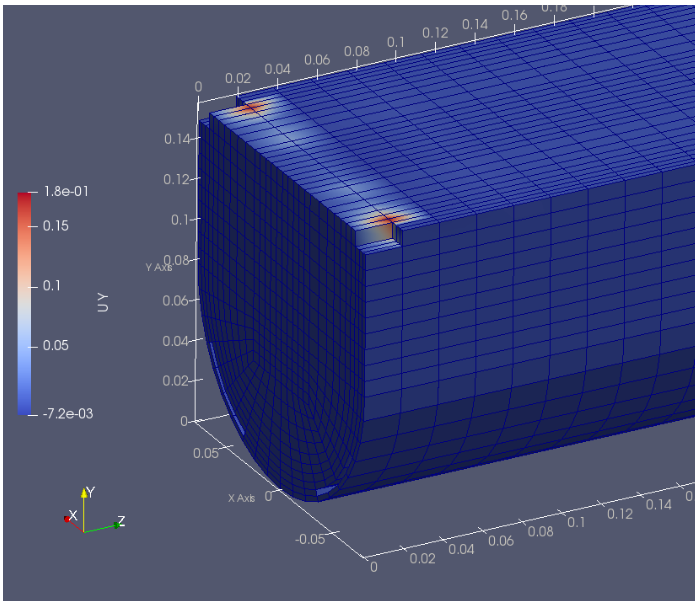

Figure 6.72: Vertical velocity Uy in simulation 7 after roughly 14.6s, focus on inlet zone. Note: z-axis is scaled by factor 0.05.

6.7.8. Fix underdetermined cells

It's obvious that our attempt at removing the underdetermined cells did not help. To overcome the trouble, we can also change our grid so that no cells are underdetermined cell to begin with. First, let's plot these cells so we can see where they are located in the grid. We need to know this in order to figure out how to change our grid.

The regular checkMesh command offered by OpenFOAM doesn't report cell underdetermined-ness. We need to call checkMesh with additional parameters. We can do so as follows, and this will save any problematic cells in Sets. These Sets can be plotted in ParaView.

```
checkMesh -allGeometry -allTopology
```

This command resulted in the following result:

```
    ...
       Cell determinant (wellposedness) : minimum: 0.00050217519 average: 0.019025868
  ***Cells with small determinant (< 0.001) found, number of cells: 28
   <<Writing 28 under-determined cells to set underdeterminedCells
    ...
```

In Figure 6.73, we can see the 28 underdetermined cells being highlighted in grey colour with a blue wireframe outline. These cells are located at the top corners of the pipe profile and close to the bottom edge of the pipe.

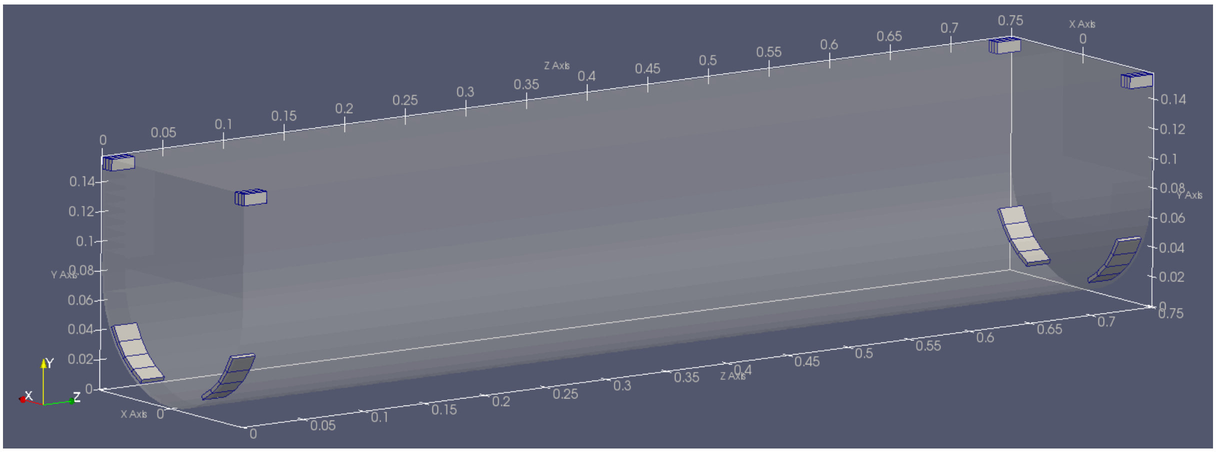

Figure 6.73: grid overview showing underdetermined cells in grey with blue wireframe. General pipe shape outline is shown in semi-transparency. Note: z-axis is scaled by factor 0.05.

The fact that these underdetermined cells are located on the outer edges of the pipe tells me the grading of the cell mesh is resulting in this underdetermined-ness. The cell grading has made cells closer to the pipe wall thinner compared to the centre of the pipe. We could reduce this grading effect, or we can increase the amount of cells we use to discretize our grid along the z-axis, both will result in a lower aspect ratio on the cells. Having a lower aspect ratio will increase the determinant on each cell.

To make sure no detail near the wall of the pipe is lost, it's better to decrease the general cell-size in the z-direction. In previous instances, our grid was discretized into 40 cells along the z-axis. In this instance, we have discretized it in 60 cells. The pipe grid profile in the xy-plane has remained unchanged. Figure 6.74 shows the old grid, and Figure 6.75 shows our new grid. With this new grid, the elaborate `checkMesh -allGeometry -allTopology` shows all checks are OK.

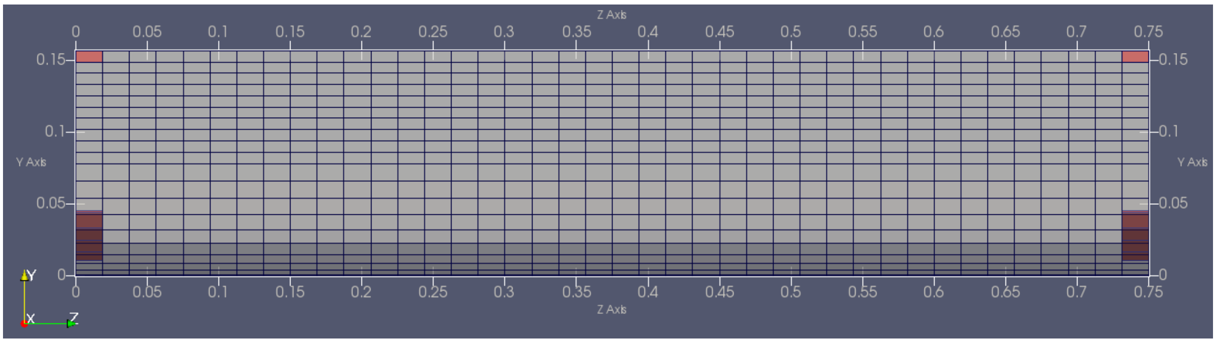

Figure 6.74: overview of grid along z-axis, discretized into 40 cells along z-axis. Highlighted in red are the underdetermined. cells. Note: z-axis is scaled by factor 0.05.

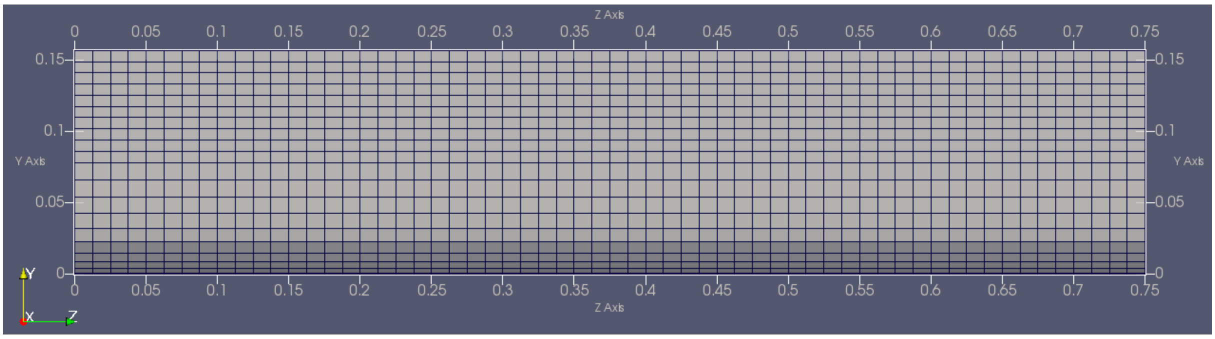

Figure 6.75: overview of grid along z-axis, discretized into 60 cells along z-axis. Note: z-axis is scaled by factor 0.05.

When running the simulation with this new grid, it unfortunately still doesn't perform as expected. After a few iterations, the timestep again becomes very small and the simulation doesn't progress anymore. Figure 6.76 shows this. A reason why this happens has not been found thus far.

Interestingly enough, the maximum courant number exceeds the maximum courant number after just a few iterations. An explanation for this has not been found.

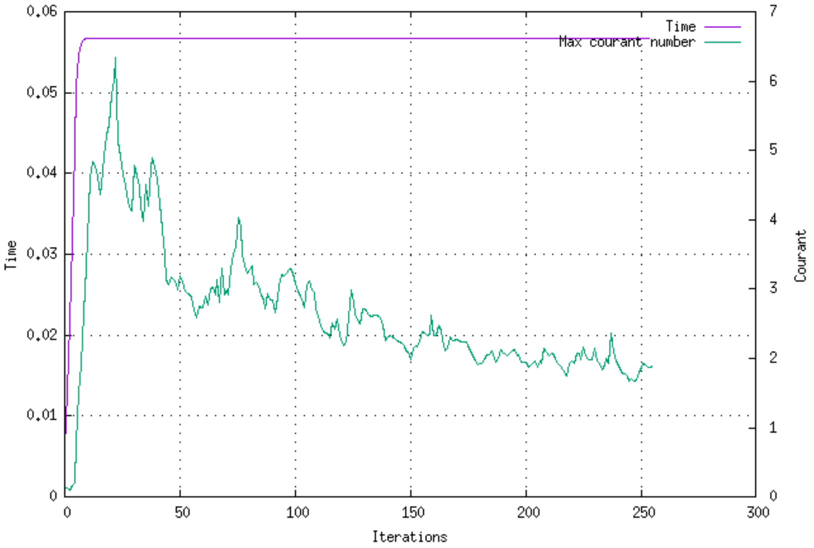

Figure 6.76: graph showing the solver continues to take smaller time steps as iterations progress (x-axis). The left y-axis shows the simulation time. On the x-axis we see the amount of iterations, and the right y-axis shows the maximum courant number

6.7.9. Conclusions

After looking at the results of the initial simulation 7, we thought the underdetermined cells in our grid were causing small timesteps. As it turns out, that didn’t matter much for the outcome. After changing our grid to get rid of the underdetermined cells, the result was the same. In both cases, the simulation started taking too small timesteps and stopped progressing.

7. Conclusions & recommendations

This chapter will present the conclusions based on the findings of this study. Further, it intends to pose recommendations towards further research based on educated guesses of what might have gone wrong in our research, and topics that might be an interesting exploration in the field in general.

7.1. Conclusions

Thus far, this work has not proved it possible to compare the simulation results to actual experimental work as most simulations crashed or showed unphysical results. The simulations ran with the pipe geometry have all failed in that sense.

However, not all is lost, for the earlier simulations (1, 2, and 3) we could actually compare the qualitative effects of sand particles settling and a general sand density profile over the flowdepth. These profiles seemed to match pretty well with experimental work performed by (Spelay, 2007). So it seems fair to say the adaptation of `interFoam` has been successful and the solver is capable of simulating settling processes of solid particles in a non-Newtonian free mixture surface flow.

7.2. Recommendations based on this research

The recommendations in this section are mostly aimed at findings a solution to overcome the trouble found in simulation 5, 6 and 7. In those specific simulations, the density of the sand particle field is taken into account for the density of the mixture and thus as a driving force for the fluid flow. As we have seen, each of the attempts has failed and not yielded a usable result. Unfortunately, the scope and time constraint of this research was not sufficient enough to find a solution.

7.2.1. Allow for slip

In our 2D open channel simulations (simulation 1, 2, and 3) we found that the velocity tends to 0 along the top edge of the domain. Along that edge, we wanted to have a slipping boundary. On this boundary we had used a `pressureInletOutletVelocity` we believed allows for slipping in the tangential direction.

The documentation[4] and the source code[5] are not definitive or explicit, but the `pressureInletOutletVelocity` boundary condition type seems to set the `tangentialVelocity` to be (0 0 0) in the constructor, if the keyword `tangentialVelocity` is omitted. In the configuration for both simulations, this keyword was indeed not added. It should be noted here that the documentation also makes mention of a value keyword, whereas this is not associated with this boundary condition as found in the source code.

While researching the source code, a boundary condition called `pressureInletOutletParSlipVelocity` was also encountered. According to the description, this type of condition always applies a slip condition tangentially. This seems to be the better option.

[4] https://www.openfoam.com/documentation/guides/latest/doc/guide-bcs-outlet-pressure-inlet-outlet.html

[5] https://github.com/OpenFOAM/OpenFOAM-5.x/blob/master/src/finiteVolume/fields/fvPatchFields/derived/pressureInletOutletVelocity/pressureInletOutletVelocityFvPatchVectorField.H

7.2.2. Simulation stability

It would be wise to investigate simulation stability. As we have seen, many of the simulations have been concluded with an unsatisfactory result. The simulations have crashed many times. A reason for this has not been found. The recommendation is to investigate further why this is happening. A first attempt could be to run the simulation without sand added to the mixture, and in a second step add sand to the mixture again. If this is the only factor that's changed, then this should tell us something about the influence sand has on the stability of the algorithm.

When we were doing simulations without the sand particle density field in the mixture density, stability was not as much of an issue as it was in later simulations. In each of the timesteps, the density of the mixture is altered in preparation for the remaining calculations. It could be that the placement of the density alteration could influence the stability. If this is the case, then it could be that the density alteration has an effect on things like (cell-face) fluxes, surface tension calculations, and in a more general sense the momentum balance equations.

Further, it is imaginable the boundary conditions applied to the sand field could also be having an influence on the stability of the simulation.

It's also recommended to run the same (or similar) simulations simply using a (now available) newer version of OpenFOAM. This work was performed using OpenFOAM 5.0 (foundation), and since the start of this work, versions 6 through 10 have been released. It's possible that newer versions of OpenFOAM contain updates that solve (some of the noted) simulation stability issues.

Between different simulation sets in this work, we switch from a 2D rectangular open channel to a 3D pipe. The simulations using a 2D rectangular open channel in this work did not show stability issues, while the 3D pipe did. Looking at it from a complexity perspective, our last recommendation regarding the simulation stability issues is to simulate a 3D rectangular open channel. A simulation with such a geometry can be used to learn if the stability issues find their origin in the switch from a 2D geometry to a 3D geometry, or if that's related to the round 3D pipe geometry.

7.2.3. Underdetermined cells and grid coarseness

In our last simulation, simulation 7, it's been discovered that the grid we've initially configured contained underdetermined cells. An alternative, slightly finer mesh has been proposed and used for a re-try. For completeness, it would be recommended to check the grids used in simulation 1 through 6 to see if those grids had any underdetermined cells.

In hindsight, the grid used for the simulations with the pipe (4-7) was a lot coarser than the grid used in simulation 1 and 2. Looking back on the simulation performed by (van Rhee, 2017), the half-pipe grid was a lot finer meshed. The grids used in this research could be revised to be finer meshed in an attempt to see if that will help with simulation progression and to see if that would have an influence on the pipe filling up as seen in simulation 4 and 5.

7.2.4. Pipe inclination

Taking a step back, it was never conclusively discovered why in simulation 4 the entire pipe filled up with the mixture. We've tried setting the pipe inclination angle to a higher value, but this resulted in crashing simulation. The experimental work performed by (Spelay, 2007) has shown the pipe doesn't fill up at all, so somewhere the simulation must be showing unphysical results.

Based on the yield stress and for a given geometry, it is possible to calculate a theoretical critical angle for yield stress flow. The force balance should result in a minimum inclination.

7.3. Recommendations for future work in non-Newtonian CFD

This next section intends to describe recommendations for future work in non-Newtonian CFD. The recommendations are based on findings in this research and relate to different types of material models, different geometries, and a different way to model the sand particles in the mixture.

7.3.1. Time dependent fluid

Section 2.1.1 showed there are different classes of non-Newtonian rheological fluid models. One class that has not been investigated is fluids that show time-dependent behaviour. These can either be thixotropic (thickening over time) or rheopectic (thinning over time). It's not unthinkable that in basins and reclamation areas, fluids come to a standstill at some point in time or position in the basin. At this point, the fluid stops shearing and this could influence the viscosity (unremoulded) and it's tendency to start flowing again under a certain force.

As seen in the simulations performed in this work, a plug zone forms. Besides a potential standstill of the fluid in a basin, in this plug zone the fluid also stops shearing.

In future work it could be interesting to implement a time dependent viscosity model to simulate this behaviour. Obviously, it's crucial to find a worthy verification case in search of a proper implementation.

7.3.2. Different geometries

The simulation that have been performed in this research pertained to two shapes: a 2D sloped channel and a 3D straight pipe section. Rather simple geometries have been chosen to better facilitate comparing results to empirical results.

However, in the field of dredging engineering, pipe geometries are not necessarily just straight forward. The pipes could have many twists, turns and bends. It could be interesting to see if a simulation with a twisty pipe shows realistic results in terms of, for example, sand bed build up in those bends.

Similarly, it can also be interesting to simulate river sand bed sedimentation with comparable material models as used in this research. In that case a choice could be made to just simulate a water-sand mixture, so the whole non-Newtonian aspect of the carrier fluid would be omitted. Although for this type of problem, a 3-phase simulation would potentially allow for more realistic simulation. The 3 phases could be air, water, and sand+mud. The only reason air would be incorporated in such a simulation would be to allow for a free water surface. It could be debated whether that's really an interesting to look at in a first simulation for sand sedimentation research in rivers.

7.3.3. Beach slope prediction

As the problem domain is focused on waste materials (tailings) in mining engineering, it's interesting to run a full-fledged, full-scale simulation on a basin-like domain. These basins are bounded by dams and are considered quite large. Parallelly, it's just as interesting for the dredging engineering field to run a full-scale simulation.

In this research, the behaviour of the fluids was simulated in either a rectangular (2D) domain or a circular pipe-section (3D). It's interesting to see how the implementation would hold up in a problem domain that's a little wider than that; a little closer to the scope of the problems out in the field that is. This would entail building a simulation domain that's equal (or close to) some real life scenarios.

Doing research in this direction could help find answers on questions like:

- how does the beach slope develop over time? Knowing this helps in predicting requirements for additional or raising embankment.
- how much sand is deposited anywhere throughout the domain? (inhomogeneous concentration and bed height)
- how strong or stiff is the deposited bed?

On top of the above, the simulation could also be used to investigate mud lobe forming and channelization at the mixture's surface.

7.3.4. Uniform particle size vs size distribution

To be able to use these simulations as a prediction method for real life scenarios, the simulation should preferably be executed using parameters that are the same, or as close as can be to the real thing.

In this research, a uniform sand particle size was used to model the sand particles that are part of the fluid mixture. In reality, it's much more likely that the sand particle size in the considered fluid is non-uniform. Each of the particles has its own (hindered) settling velocity and the differently sized particles have different influences on the (local) viscosity of the fluid. This can influence the rheological conditions of the fluid and in term can influence most (if not all) other parameters of the flow. Further, in reality this size distribution can also vary across the domain considered, leading to an additional layer of complexity in the analysis.

In potential future research, this implementation could be done using a multiphase approach. The implementation could allow for 2 (or n for that matter) phases of sand. Each phase would represent a different subset of particle sizes in the sand particle distribution. Each of these phases would have to get their own settling velocity.

Combining the phases to compute a total sand fraction allows for calculating influence on the local viscosity. Section 2.2 shows there to be no relation between particle size and the influence on the viscosity, though at this point I'm unaware of the conditions for which that holds true.

In turn, the effect on the density of the mixture could be calculated by combining the sand phases using a fraction method. After all, the current implementation already combines the (total) sand density into the mixture density. Researching an adapted model for the hindered settling would be advisable.

List of Figures

List of Tables

Bibliography

Afshar, M. A. (2010). *Numerical Wave Generation In OpenFOAM®.* Chalmers tekniska högskola.

Boger, D. V., Scales, P. J., & Sofra, F. (2006). *Rheological concepts.* Perth, Australia: Australian Centre for Geomechanics.

Chhabra, R. P. (2007). *Bubbles, Drops, and Particles in non-Newtonian Fluids* (2nd ed.). CRC Press.

Courant, R., Friedrichs, K., & Lewy, H. (1928). Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 32-74. doi:https://doi.org/10.1007/BF01448839

Damián, S. M. (2013). *An Extended Mixture Model for the Simultaneous Treatment of Short and Long Scale Interfaces.* PhD thesis, UNIVERSIDAD NACIONAL DEL LITORAL. doi:http://dx.doi.org/10.13140/RG.2.1.3182.8320

Dankers, P. J., & Winterwerp, J. C. (2007). Hindered settling of mud flocs: Theory and validation. *Continental Shelf Research, 27*(14), 1893-1907. doi:https://doi.org/10.1016/j.csr.2007.03.005

De Kee, D., Chhabra, R. P., Powley, M. B., & Roy, S. (1990). Flow of viscoplastic fluids on an inclined plane: evaluation of yield stress. *Chemical Engineering Communications, 96, 1*, 229-239. doi:https://doi.org/10.1080/00986449008911493

Haldenwang, R., Kotzé, R., & Chhabra, R. (2012). Determining the Viscous Behavior of Non-Newtonian Fluids in a Flume Using a Laminar Sheet Flow Model and Ultrasonic Velocity Profiling (UVP) System. *Journal of the Brazilian Society of Mechanical Sciences and Engineering Vol 34, 3*. doi:http://dx.doi.org/10.1590/S1678-58782012000300008

Haldenwang, R., Slatter, P. T., & Chhabra, R. P. (2010). An experimental study of non-Newtonian fluid flow in rectangular flumes in laminar, transition and turbulent flow regimes. *Jounal of the south african institution of civil engineering, 52*, 11-19. Retrieved from http://www.scielo.org.za/scielo.php?script=sci_arttext&pid=S1021-20192010000100002&lng=en&tlng=en

Hanssen, J. L. (2016). *Towards improving predictions of non-Newtonian settling slurries with Delft3D.* Delft: Delft University of Technology.

Issa, R. I. (1986, January). Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, 40-65. doi:https://doi.org/10.1016/0021-9991(86)90099-9

Jacobs, W., Le hir, P., Van Kesteren, W., & Cann, P. (2011, July 15). Erosion threshold of sand–mud mixtures. *Continental Shelf Research, 31*(10), S14-S25. doi:https://doi.org/10.1016/j.csr.2010.05.012

Jasak, H. (1996). *Error Analysis and Estimation for the Finite Volume Method with Applications to Fluid Flows.* Department of Mechanical Engineering Imperial College of Science, Technology and Medicine. Retrieved from https://www.researchgate.net/publication/230605842_Error_Analysis_and_Estimation_for_the_Finite_Volume_Method_With_Applications_to_Fluid_Flows

Papanastasiou, T. C. (1987). Flows of Materials with Yield. *Journal of Rheology 31, 385*. doi:https://doi.org/10.1122/1.549926

Patankar, S. V. (1980). *Numerical Heat Transfer and Fluid Flow.* CRC Press. doi:https://doi.org/10.1201/9781482234213

Peeters, P. T. (2016). *CFD of multiphase pipe flow: a comparison of solvers.* Delft: Delft University of Technology.

Slatter, P. (2011). The Engineering Hydrodynamics of Viscoplastic Suspensions. *Particulate Science and Technology`*, 139-150. doi:https://doi.org/10.1080/02726351.2010.527429

Spelay, R. B. (2007). *Solids transport in laminar, open channel flow of non-Newtonian slurries.* PhD Thesis, University of Saskatchewan, Saskatoon, Saskatchewan, Canada.

Talmon, A. M. (2016). *OE4625 Lecture notes - Segregating non-Newtonian slurries.* Delft: Delft University of Technology.

Talmon, A. M., & Huisman, M. (2005). Fall Velocity of particles in shear flow of drilling fluids. *Tunnelling and Underground Space Technology, 20*, 193-201. doi:https://doi.org/10.1016/j.tust.2004.07.001

Talmon, A. M., Hanssen, J. L., Winterwerp, J. C., Sitoni, L., & van Rhee, C. (2016). Implementation of Tailings Rheology in a Predictive Open-Channel Beaching Model. *PASTE 2016, 19th International Seminar on Paste and Thickened Tailings.*

Talmon, A. M., van Kesteren, W. G., Sittoni, L., & Hedblom, E. P. (2013). Shear Cell Tests For Quantification Of Tailings Storage Facilities. *The Canadian Journal Of Chemical Engineering*, 362-373. doi:https://dx.doi.org/10.1002/cjce.21856

Thomas, A. D. (1999). The influence of coarse particles on the rheology of fine particle slurries. *Proceedings of Rheology in the Mineral Industry II*, 113–123.

Van De Ree, T. (2015). *Deposition of high density tailings on beaches.* Delft: Delft University of Technology.

van Es, H. E. (2017). *Development of a numerical model for dynamic depositioning of non-Newtonian slurries.* Delft: Delft University of Technology.

van Rhee, C. (2017). Simulation of the settling of solids in a non-Newtonian fluid. *18th International Conference on Transport and sedimentation of solid particles*, (pp. 265-270). Prague, Czech Republic.

Winterwerp, J. C., & van Kesteren, W. G. (2004). *Introduction to the physics of cohesive sediment in the marine environment.* Elsevier.

Appendices

Appendix A. Source code – van Rhee (2017)

A.1. CVRinterFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    CVRinterFoam

Description
    Solver for 2 incompressible, isothermal immiscible fluids using a VOF
    (volume of fluid) phase-fraction based interface capturing approach.

    The momentum and other fluid properties are of the "mixture" and a single
    momentum equation is solved.

    Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.

    For a two-fluid approach see twoPhaseEulerFoam.

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "CMULES.H"
#include "EulerDdtScheme.H"
#include "localEulerDdtScheme.H"
#include "CrankNicolsonDdtScheme.H"
#include "subCycle.H"
#include "immiscibleIncompressibleTwoPhaseMixture.H"
#include "turbulentTransportModel.H"
#include "pimpleControl.H"
#include "fvOptions.H"
#include "CorrectPhi.H"
#include "fvcSmooth.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "postProcess.H"
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createTimeControls.H"
    #include "createRDeltaT.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "createFvOptions.H"
    #include "correctPhi2.H"

    turbulence->validate();

    if (!LTS)
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setInitialDeltaT.H"
    }

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        #include "readTimeControls.H"

        if (LTS)
        {
            #include "setRDeltaT.H"
        }
        else
        {
            #include "CourantNo.H"
            #include "alphaCourantNo.H"
            #include "setDeltaT.H"
        }

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "alphaControls.H"
            #include "alphaEqnSubCycle.H"

            mixture.correct();

            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
            #include "csandEqn.H"
        }

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return 0;
}

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

A.2. CSandEqn.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    CVRnonNewtonianIcoFoam
    AlphaEqn.C     -file met transport equation for alpha

Description
    Transient solver for incompressible, laminar flow of non-Newtonian fluids.

\*---------------------------------------------------------------------------*/

              Info << " Calculation Sand transport " << endl;

              dimensionedScalar zero("zero", dimless, 0);
              dimensionedScalar one("one", dimless, 1.0);
              dimensionedScalar factor("factor", dimless, 1.0/18.0);

       //       Info<< "rhoc = " << rhoc << endl;

       //       volVectorField wsvol(U*zero);

       //       volScalarField alpham(alpha);

        //      alpham+=cfine;

        //      Info<< "max alpham = " << max(alpham) << endl;

              wsvol = factor * (rhos-rho)*sqr(Diam) / (mixture.muws())*gz;

              wsvol *= (one -csand);

           //   ws1 = Wsettle.ws();

        //      if (Method=="TalmonHuisman")
        //      {
       //          wsvol*=(one -alpham)* sqr(one - alpha/cmax);   // hindered settling
            //     Info<< "Method settling velocity = " << Method << endl;
        //      }
       //       else  // volgens Spilay
          //    {
            //     wsvol*=(one -alpha);
           //   }

             // Driftflux ten opzichte van bulk velocity

             Us = U + wsvol;

         //   volScalarField visco(fluid.nu());

         //   surfaceScalarField viscface = fvc::interpolate(visco);

         //   surfaceVectorField ws = factor * (rhos-rhow)*g*sqr(Diam) / (viscface*rhow)*eenheidsvector;

           // Us = fvc::interpolate(U);

           // Us+=ws;

          //     Info<< "min abs(viscface) = " << min(viscface) << endl;
          //     Info<< "min abs(ws) = " << min(ws.component(1)) << endl;
           //    Info<< "max abs(wsvol) = " << max(mag(wsvol)) << endl;
            //   Info<< "max max Us .y = " << max(Us.component(1)) << endl;

            //= fvc::interpolate(U)+ws;

            surfaceScalarField phised = fvc::interpolate(Us) & mesh.Sf();

        //    surfaceScalarField phised = fvc::flux(Us1);

            fvScalarMatrix csandEqn
        (
            fvm::ddt(csand)
          + fvm::div(phised, csand)

        );
         csandEqn.solve();

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

A.3. createFields.H

```
IOdictionary transportProperties
    (
        IOobject
        (
            "transportProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );

   dictionary& subDict = transportProperties.subDict("TalmonCoeffs");

   dimensionedScalar cmax
   (
     subDict.lookup("cmax")

   );

  // subDict = transportProperties.subDict("SettlingVelocityMethod");
  // word Method
  // (

  //  subDict.lookup("Method")
  // );

  // Info<< "Selecting " << Method << " as settling velocity method\n" << endl;

dimensionedScalar Diam
    (
        transportProperties.lookup("Diam")
    );
dimensionedScalar rhow
    (
        transportProperties.lookup("rhow")
    );
dimensionedScalar rhos
    (
        transportProperties.lookup("rhos")
    );

Info<< "Reading field p_rgh\n" << endl;
volScalarField p_rgh
(
    IOobject
    (
        "p_rgh",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field U\n" << endl;
volVectorField U
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);
Info<< "Reading field Us\n" << endl;
volVectorField Us
(
IOobject
        (
        "Us",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
        ),
        mesh

);

volVectorField wsvol
(
IOobject
        (
        "wsvol",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
        ),
        mesh

);

```

```
volScalarField csand
(
    IOobject
    (
        "csand",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "createPhi.H"

Info<< "Reading transportProperties\n" << endl;
immiscibleIncompressibleTwoPhaseMixture mixture(U, phi);

volScalarField& alpha1(mixture.alpha1());
volScalarField& alpha2(mixture.alpha2());

const dimensionedScalar& rho1 = mixture.rho1();
const dimensionedScalar& rho2 = mixture.rho2();


// Need to store rho for ddt(rho, U)
volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT
    ),
    alpha1*rho1 + alpha2*rho2
);
rho.oldTime();


// Mass flux
surfaceScalarField rhoPhi
(
    IOobject
    (
        "rhoPhi",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    fvc::interpolate(rho)*phi
);

// Construct incompressible turbulence model
autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, mixture)
);

#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"

volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rho*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
(
    p,
    p_rgh,
    pimple.dict(),
    pRefCell,
    pRefValue
);

```

```
if (p_rgh.needReference())
{
    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
        pRefValue - getRefCellValue(p, pRefCell)
    );
    p_rgh = p - rho*gh;
}

mesh.setFluxRequired(p_rgh.name());
mesh.setFluxRequired(alpha1.name());

dimensionedVector gz("gz", dimLength/sqr(dimTime), vector(0, -9.81,0));

// MULES flux from previous time-step
surfaceScalarField alphaPhi
(
    IOobject
    (
        "alphaPhi",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT,
        IOobject::AUTO_WRITE
    ),
    phi*fvc::interpolate(alpha1)
);

// MULES Correction
tmp<surfaceScalarField> talphaPhiCorr0;
#include "createMRF.H"
```

Appendix B. Source code - solver

B.1. interFoamPeter.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2017 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Application
    interFoam

Description
    Solver for 2 incompressible, isothermal immiscible fluids using a VOF
    (volume of fluid) phase-fraction based interface capturing approach.

    The momentum and other fluid properties are of the "mixture" and a single
    momentum equation is solved.

    Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.

    For a two-fluid approach see twoPhaseEulerFoam.

    Author: Cees van Rhee & Peter Dobbe

\*---------------------------------------------------------------------------*/

#include "fvCFD.H"
#include "CMULES.H"
#include "EulerDdtScheme.H"
#include "localEulerDdtScheme.H"
#include "CrankNicolsonDdtScheme.H"
#include "subCycle.H"
#include "immiscibleIncompressibleTwoPhaseMixture.H"
#include "turbulentTransportModel.H"
#include "pimpleControl.H"
#include "fvOptions.H"
#include "CorrectPhi.H"
#include "fvcSmooth.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

int main(int argc, char *argv[])
{
    #include "postProcess.H"

    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "createControl.H"
    #include "createTimeControls.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "createAlphaFluxes.H"
    #include "createFvOptions.H"
    #include "correctPhi2.H" // to prevent capital sensitive issues

    turbulence->validate();

    if (!LTS)
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setInitialDeltaT.H"
    }

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        Info << "running readTimeControls.H" <<endl;
        #include "readTimeControls.H"

        if (LTS)
        {
            #include "setRDeltaT.H"
        }
        else
        {
            #include "CourantNo.H"
            #include "alphaCourantNo.H"
            #include "setDeltaT.H"
        }

        runTime++;
        Info<< "Time = " << runTime.timeName() << nl << endl;

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "alphaControls.H"

            #include "alphaEqnSubCycle.H"

            mixture.correct();
            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                Info << "Running pimple.correct()" << endl;
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                Info << "Running turbulence->correct()" << endl;
                turbulence->correct();
            }
            #include "CSandEqn.H"
        }
        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
    Info<< "End\n" << endl;

    return 0;
}
// ************************************************************************* //
```

B.2. CSandEqn.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------



Application
    interFoamPeter

Description
    Transport equation for sand.

    Author: Cees van Rhee & Peter Dobbe

\*---------------------------------------------------------------------------*/

    Info << "Running CSandEqn.H" << endl;

    dimensionedScalar zero("zero", dimless, 0);

    dimensionedScalar one("one", dimless, 1.0);

    dimensionedScalar factor("factor", dimless, 1.0/18.0);

    volScalarField muws_mixture = mixture.muws();
    Info << "muws in CSandEqn.  Min(muws) = " << min(muws_mixture).value() << " Max(muws) = " << max(muws_mixture).value() << endl;

    wsvol = factor * (((rhos-rho)*sqr(Diam)*g) / (muws_mixture));

    wsvol *= (one - csand);

    Info << "wsvol in CSandEqn.  Min(wsvol) = " << min(wsvol).value() << " Max(wsvol) = " << max(wsvol).value() << endl;

    Us = U + wsvol;

    surfaceScalarField phised = fvc::interpolate(Us) & mesh.Sf();

    fvScalarMatrix csandEqn
    (
        fvm::ddt(csand)
        + fvm::div(phised, csand)

    );

    csandEqn.solve();

    Info << "Alpha in CSandEqn.  Min(csand) = " << min(csand).value() << " Max(csand) = " << max(csand).value() << endl;

    Info << "csand-cmax in CSandEqn.  Min(csand-cmax) = " << min(csand-cmax).value() << " Max(csand-cmax) = " << max(csand-cmax).value() << endl;

    Info << "mag(csand-cmax) in CSandEqn.  Min(mag(csand-cmax)) = " << min(mag(csand-cmax)).value() << " Max(mag(csand-cmax)) = " << max(mag(csand-cmax)).value() << endl;



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

B.3. CSandEqn.H – alternative for simulation 3

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2016 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------



Application
    interFoamPeter

Description
    Transport equation for sand.

    Author: Cees van Rhee & Peter Dobbe

\*---------------------------------------------------------------------------*/

    Info << "Running CSandEqn.H" << endl;

    dimensionedScalar zero("zero", dimless, 0);

    dimensionedScalar one("one", dimless, 1.0);

    dimensionedScalar factor("factor", dimless, 1.0/18.0);

    volScalarField muws_mixture = mixture.muws();
    Info << "muws in CSandEqn.  Min(muws) = " << min(muws_mixture).value() << " Max(muws) = " << max(muws_mixture).value() << endl;

    wsvol = factor * (((rhos-rho)*sqr(Diam)*g) / (muws_mixture));

    wsvol *= (one - csand) * sqr(one - (csand/cmax));

    Info << "wsvol in CSandEqn.  Min(wsvol) = " << min(wsvol).value() << " Max(wsvol) = " << max(wsvol).value() << endl;

    Us = U + wsvol;

    surfaceScalarField phised = fvc::interpolate(Us) & mesh.Sf();

    fvScalarMatrix csandEqn
    (
        fvm::ddt(csand)
        + fvm::div(phised, csand)

    );

    csandEqn.solve();

    Info << "Alpha in CSandEqn.  Min(csand) = " << min(csand).value() << " Max(csand) = " << max(csand).value() << endl;

    Info << "csand-cmax in CSandEqn.  Min(csand-cmax) = " << min(csand-cmax).value() << " Max(csand-cmax) = " << max(csand-cmax).value() << endl;

    Info << "mag(csand-cmax) in CSandEqn.  Min(mag(csand-cmax)) = " << min(mag(csand-cmax)).value() << " Max(mag(csand-cmax)) = " << max(mag(csand-cmax)).value() << endl;



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

B.4. createFields.H

```
#include "createRDeltaT.H"

    Info<< "Reading field csand\n" << endl;
    volScalarField csand
    (
        IOobject
        (
            "csand",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

    // Read wsvol field initialized in 0 folder
    Info<< "Reading field wsvol\n" << endl;
    volVectorField wsvol
    (
        IOobject
        (
            "wsvol",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

    // Read Us field initialized in 0 folder
    Info<< "Reading field Us\n" << endl;
    volVectorField Us
    (
        IOobject
        (
            "Us",
            runTime.timeName(),
            mesh,
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        mesh
    );

Info<< "Reading field p_rgh\n" << endl;
volScalarField p_rgh
(
    IOobject
    (
        "p_rgh",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

Info<< "Reading field U\n" << endl;
volVectorField U
```

```
(
    IOobject
    (
        "U",
        runTime.timeName(),
        mesh,
        IOobject::MUST_READ,
        IOobject::AUTO_WRITE
    ),
    mesh
);

#include "createPhi.H"

Info<< "Reading transportProperties\n" << endl;
immiscibleIncompressibleTwoPhaseMixture mixture(U, phi);

      Info << mixture << endl;
   IOdictionary transportProperties
    (
        IOobject
        (
            "transportProperties",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );

    dimensionedScalar Diam
    (
        transportProperties.lookup("Diam")
    );
    dimensionedScalar rhow
    (
        transportProperties.lookup("rhow")
    );
    dimensionedScalar rhos
    (
        transportProperties.lookup("rhos")
    );
    dimensionedScalar cfine
    (
        transportProperties.lookup("cfine")
    );
    dimensionedScalar cmax
    (
        transportProperties.lookup("cmax")
    );
    //dimensionedVector gz("gz", dimLength/sqr(dimTime), vector(0, -9.81,0));

dimensionedScalar rhoc("rhoc",cfine*rhos + (1-cfine)*rhow);

volScalarField& alpha1(mixture.alpha1());
volScalarField& alpha2(mixture.alpha2());

const dimensionedScalar& rho1 = mixture.rho1();
const dimensionedScalar& rho2 = mixture.rho2();


// Need to store rho for ddt(rho, U)
volScalarField rho
(
    IOobject
    (
        "rho",
        runTime.timeName(),
        mesh,
        IOobject::READ_IF_PRESENT
    ),
    alpha1*rho1 + alpha2*rho2
);
rho.oldTime();

// Mass flux
surfaceScalarField rhoPhi
(
    IOobject
    (
        "rhoPhi",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::NO_WRITE
    ),
    fvc::interpolate(rho)*phi
);
```

```

// Construct incompressible turbulence model
autoPtr<incompressible::turbulenceModel> turbulence
(
    incompressible::turbulenceModel::New(U, phi, mixture)
);

#include "readGravitationalAcceleration.H"
#include "readhRef.H"
#include "gh.H"

volScalarField p
(
    IOobject
    (
        "p",
        runTime.timeName(),
        mesh,
        IOobject::NO_READ,
        IOobject::AUTO_WRITE
    ),
    p_rgh + rho*gh
);

label pRefCell = 0;
scalar pRefValue = 0.0;
setRefCell
(
    p,
    p_rgh,
    pimple.dict(),
    pRefCell,
    pRefValue
);

if (p_rgh.needReference())
{
    p += dimensionedScalar
    (
        "p",
        p.dimensions(),
        pRefValue - getRefCellValue(p, pRefCell)
    );
    p_rgh = p - rho*gh;
}

mesh.setFluxRequired(p_rgh.name());
mesh.setFluxRequired(alpha1.name());

#include "createMRF.H"
```

Appendix C. Source code - viscosity models

C.1. Talmon.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::viscosityModels::Talmon

Description
     Talmon non-Newtonian viscosity model.

SourceFiles
    Talmon.H

\*---------------------------------------------------------------------------*/

#ifndef Talmon_H
#define Talmon_H

#include "viscosityModel.H"
#include "dimensionedScalar.H"
#include "volFields.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace viscosityModels
{

/*---------------------------------------------------------------------------*\
                        Class Talmon Declaration
\*---------------------------------------------------------------------------*/

class Talmon
:
    public viscosityModel
{
    // Private data

        dictionary TalmonCoeffs_;

        dimensionedScalar coef_;
        dimensionedScalar cmax_;
        dimensionedScalar alpha0_;
        dimensionedScalar tau0_;
        dimensionedScalar nu0_;
        dimensionedScalar numax_;
        word phasename_;
        dimensionedScalar maxlabda;
        dimensionedScalar minlabda;

        const volScalarField& alpha_;
        volScalarField nu_;
        volScalarField nuws_;
        volScalarField labda_;


    // Private Member Functions

        //- Calculate and return the laminar viscosity
        tmp<volScalarField> calcNu() const;
        tmp<volScalarField> calcNuws() const;
        tmp<volScalarField> calcLabda() const;

protected:

public:

    //- Runtime type information
    TypeName("Talmon");

    // Constructors

        //- Construct from components
        Talmon
        (
            const word& name,
            const dictionary& viscosityProperties,
            const volVectorField& U,
            const surfaceScalarField& phi
        );

    //- Destructor
    ~Talmon()
    {}

    // Member Functions

        //- Return the laminar viscosity
        tmp<volScalarField> nu() const
        {
            return nu_;
        }

         //- Return the laminar viscosity
        tmp<volScalarField> nuws() const
        {
            return nuws_;
        }

        //- Return the linear concentration labda
        tmp<volScalarField> labda() const
        {
            return labda_;
        }

        //- Return the laminar viscosity for patch
        tmp<scalarField> nu(const label patchi) const
        {
            return nu_.boundaryField()[patchi];
        }

        //- Correct the laminar viscosity
        void correct()
        {
            nu_ = calcNu();
            nuws_ = calcNuws();
        }

        //- Read transportProperties dictionary
        bool read(const dictionary& viscosityProperties);
};

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace viscosityModels
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

// ************************************************************************* //

```

C.2. Talmon.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2015 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "Talmon.H"
#include "addToRunTimeSelectionTable.H"
#include "surfaceFields.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
    namespace viscosityModels
    {
        defineTypeNameAndDebug(Talmon, 1);

        addToRunTimeSelectionTable
        (
            viscosityModel,
            Talmon,
            dictionary
        );
    }
}

// * * * * * * * * * * * * Private Member Functions  * * * * * * * * * * * * //

Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Talmon::calcNu() const
{
    dimensionedScalar one("one", dimless, 1.0);
    //dimensionedScalar one3("onethird", dimless, 1.0/3.0);
    dimensionedScalar klein("klein", dimless, 1e-5);

    tmp<volScalarField> sr(strainRate());

    Info << phasename_ << " in calcNu. ("<<phasename_<<") Min("<<phasename_<<") = " << min(alpha_).value() << " Max("<<phasename_<<") = " << max(alpha_).value() << endl;

    Info << phasename_<<"-cmax_ in calcNu.  Min("<<phasename_<<"-cmax_) = " << min(alpha_-cmax_).value() << " Max("<<phasename_<<"-cmax_) = " << max(alpha_-cmax_).value() << endl;

    Info << "mag("<<phasename_<<"-cmax_) in calcNu.  Min(mag("<<phasename_<<"-cmax_)) = " << min(mag(alpha_-cmax_)).value() << " Max(mag("<<phasename_<<"-cmax_)) = " << max(mag(alpha_-cmax_)).value() << endl;

    Info << ""<<phasename_<<"+klein in calcNu.  Min("<<phasename_<<"+klein) = " << min(alpha_+klein).value() << " Max("<<phasename_<<"+klein) = " << max(alpha_+klein).value() << endl;

    Info << "cmax_/("<<phasename_<<"+klein) in calcNu.  Min(cmax_/("<<phasename_<<"+klein)) = " << min(cmax_/(alpha_+klein)).value() << " Max(cmax_/("<<phasename_<<"+klein)) = " << max(cmax_/(alpha_+klein)).value() << endl;

    volScalarField labda_= calcLabda();

    Info << "Labda in calcNu. Min(labda) = " << min(labda_).value() << " Max(labda) = " << max(labda_).value() << endl;

    Info << "Minimum strainrate(): " << min(sr()) << endl;

    volScalarField capped_exponent = min(exp(alpha0_*labda_), dimensionedScalar ("ROOTVGREAT", dimless, ROOTVGREAT));

    if (max(exp(alpha0_*labda_)).value() >= dimensionedScalar ("ROOTVGREAT", dimless, ROOTVGREAT).value())
    {
        Info << "Warning, maximum of exponent: >= " << dimensionedScalar ("ROOTVGREAT", dimless, ROOTVGREAT) << " : " << max(exp(alpha0_*labda_)).value() << endl;
    }
    return
    (
        min(
            numax_,
            nu0_*capped_exponent + (tau0_*capped_exponent*(one-exp(-coef_*sr())) )
                /(max(sr(), dimensionedScalar ("VSMALL", dimless/dimTime, VSMALL)))
        )
    );
}

```

```
Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Talmon::calcNuws() const
{
    dimensionedScalar one("one", dimless, 1.0);
  //  dimensionedScalar one3("onethird", dimless, 1.0/3.0);
  //  dimensionedScalar klein("klein", dimless, 1e-5);

    tmp<volScalarField> sr(strainRate());

    Info<< " Berekening van eta voor valsnelheid " << endl;

    return
    (
        min(
            numax_,
            nu0_ +  ( tau0_*(one-exp(-coef_*sr())) )

            /(max(sr(), dimensionedScalar ("VSMALL", dimless/dimTime, VSMALL)))
        )
    );
}

Foam::tmp<Foam::volScalarField>
Foam::viscosityModels::Talmon::calcLabda() const
{
    dimensionedScalar one("one", dimless, 1.0);
    dimensionedScalar one3("onethird", dimless, 1.0/3.0);
    dimensionedScalar klein("klein", dimless, 1e-5);

    if (min(alpha_+klein).value() <= 0)
    {
        Info << "Warning: min(alpha_+klein) <= 0: " << min(alpha_+klein).value() << endl;
    }

    if (max(alpha_+klein).value() >= cmax_.value())
    {
        Info << "Warning: max(alpha_+klein). >= "<< cmax_.value() << " : "  << max(alpha_+klein).value() << endl;
    }

    volScalarField labda_return = (one / (pow(cmax_/(alpha_+klein),one3)-one));

    if (max(labda_return).value() > maxlabda.value())
    {
        Info << "Warning: max(labda_return)= " << max(labda_return).value() << endl;
    }

    if (min(labda_return).value() < minlabda.value())
    {
        Info << "Warning: min(labda_return)= " << min(labda_return).value() << endl;
    }
    return min(labda_return,maxlabda);
}


// * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::viscosityModels::Talmon::Talmon
(
    const word& name,
    const dictionary& viscosityProperties,
    const volVectorField& U,
    const surfaceScalarField& phi
)
:
    viscosityModel(name, viscosityProperties, U, phi),

    TalmonCoeffs_(viscosityProperties.subDict(typeName + "Coeffs")),
   // k_("k", dimViscosity, TalmonCoeffs_),
    coef_("coef", dimTime, TalmonCoeffs_),
    cmax_("cmax", dimless, TalmonCoeffs_),
    alpha0_("alpha0", dimless, TalmonCoeffs_),
    tau0_("tau0", dimViscosity/dimTime, TalmonCoeffs_),
    nu0_("nu0", dimViscosity, TalmonCoeffs_),
    numax_("numax", dimViscosity, TalmonCoeffs_),
    phasename_(TalmonCoeffs_.lookup("Phasename")),
    maxlabda("maxlabda", dimless, log(std::numeric_limits<double>::max())/alpha0_.value()),
    minlabda("minlabda", dimless, log(std::numeric_limits<double>::min())/alpha0_.value()),

    alpha_(

       U_.mesh().lookupObject<volScalarField>(phasename_)

    ),
    nu_
    (
        IOobject
        (
            name,
            U_.time().timeName(),
            U_.db(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        calcNu()
    ),
    nuws_
    (
        IOobject
        (
            name,
            U_.time().timeName(),
            U_.db(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        calcNuws()
    ),
    labda_
    (
        IOobject
        (
            name,
            U_.time().timeName(),
            U_.db(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        calcLabda()
    )

{
    Info<< " Defining CVR Talmon model " << endl;
}

// * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * //

bool Foam::viscosityModels::Talmon::read
(
    const dictionary& viscosityProperties
)
{

    viscosityModel::read(viscosityProperties);
    Info<< " Defining CVR Talmon model " << endl;
    TalmonCoeffs_ = viscosityProperties.subDict(typeName + "Coeffs");

    TalmonCoeffs_.lookup("coef") >> coef_;
    TalmonCoeffs_.lookup("cmax") >> cmax_;
    TalmonCoeffs_.lookup("alpha0") >> alpha0_;
    //TalmonCoeffs_.lookup("n") >> n_;
    TalmonCoeffs_.lookup("tau0") >> tau0_;
    TalmonCoeffs_.lookup("nu0") >> nu0_;
    TalmonCoeffs_.lookup("numax") >> numax_;
    TalmonCoeffs_.lookup("Phasename") >> phasename_;

    return true;
}

// ************************************************************************* //

```

C.3. CVRNewtonian.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2015 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::viscosityModels::CVRNewtonian

Description
    An incompressible CVRNewtonian viscosity model.

SourceFiles
    CVRNewtonian.H

\*---------------------------------------------------------------------------*/

#ifndef CVRNewtonian_H
#define CVRNewtonian_H

#include "viscosityModel.H"
#include "dimensionedScalar.H"
#include "volFields.H"

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{
namespace viscosityModels
{

/*---------------------------------------------------------------------------*\
                           Class CVRNewtonian Declaration
\*---------------------------------------------------------------------------*/

class CVRNewtonian
:
    public viscosityModel
{
    // Private data

        dimensionedScalar nu0_;

        volScalarField nu_;


public:

    //- Runtime type information
    TypeName("CVRNewtonian");


    // Constructors

        //- Construct from components
        CVRNewtonian
        (
            const word& name,
            const dictionary& viscosityProperties,
            const volVectorField& U,
            const surfaceScalarField& phi
        );


    //- Destructor
    ~CVRNewtonian()
    {}


    // Member Functions

        //- Return the laminar viscosity
        tmp<volScalarField> nu() const
        {
            return nu_;
        }
        tmp<volScalarField> nuws() const
        {
            return nu_;
        }

        //- Return the laminar viscosity for patch
        tmp<scalarField> nu(const label patchi) const
        {
            return nu_.boundaryField()[patchi];
        }

        //- Correct the laminar viscosity (not appropriate, viscosity constant)
        void correct()
        {}

        //- Read transportProperties dictionary
        bool read(const dictionary& viscosityProperties);
};


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace viscosityModels
} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

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

C.4. CVRNewtonian.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2015 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "CVRNewtonian.H"
#include "addToRunTimeSelectionTable.H"
#include "surfaceFields.H"

// * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * //

namespace Foam
{
namespace viscosityModels
{
    defineTypeNameAndDebug(CVRNewtonian, 0);
    addToRunTimeSelectionTable(viscosityModel, CVRNewtonian, dictionary);
}
}

// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * //

Foam::viscosityModels::CVRNewtonian::CVRNewtonian
(
    const word& name,
    const dictionary& viscosityProperties,
    const volVectorField& U,
    const surfaceScalarField& phi
)
:
    viscosityModel(name, viscosityProperties, U, phi),
    nu0_("nu", dimViscosity, viscosityProperties_),
    nu_
    (
        IOobject
        (
            name,
            U_.time().timeName(),
            U_.db(),
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        U_.mesh(),
        nu0_
    )
{}


// * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * //
```

```

bool Foam::viscosityModels::CVRNewtonian::read
(
    const dictionary& viscosityProperties
)
{
    viscosityModel::read(viscosityProperties);

    viscosityProperties_.lookup("nu") >> nu0_;
    nu_ = nu0_;

    return true;
}


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

Appendix D. Simulation case files

D.1. Simulation 1

D.1.1. File 0/alpha.water

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1;
    }
    bottom
    {
        type            zeroGradient;
    }
    leftwall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

D.1.2. File 0/csand

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      csand;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    leftwall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    bottom
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

D.1.3. File 0/p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

boundaryField
{
    atmosphere
    {
        type            totalPressure;
        p0              uniform 0;
    }
    ".*"
    {
        type            fixedFluxPressure;
        value           uniform 0;
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

D.1.4. File 0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.004;
    }
    bottom
    {
        type            noSlip;
    }
    leftwall
    {
        type            noSlip;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

D.1.5. File 0/Us

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      Us;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.004;
    }
    leftwall
    {
        type            noSlip;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
        value           uniform (0 0 0);
    }
    bottom
    {
        type            noSlip;// fixedValue;
       // value           uniform (0 0 0);
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}
// ************************************************************************* //
```

D.1.6. File 0/wsvol

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      wsvol;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    leftwall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    bottom
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            fixedValue; //pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.1.7. File constant/g

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0.4898 -9.80 0);

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

D.1.8. File constant/transportProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Diam              Diam [ 0 1 0 0 0 0 0 ] 188e-06;

rhos              rhos [ 1 -3 0 0 0 0 0 ] 2650;

rhow              rhow [ 1 -3 0 0 0 0 0 ] 1000;

nu              [0 2 -1 0 0 0 0]  1.48e-05;

cfine             cfine [ 0 0 0 0 0 0 0 ] 0.151;

cmax              cmax [ 0 0 0 0 0 0 0 ] 0.6;

TalmonCoeffs
{
        Phasename csand;

        coef [0 0 1 0 0 0 0]     50;
        cmax       cmax [0 0 0 0 0 0 0]     0.6;
        alpha0       [0 0 0 0 0 0 0]     0.27;
        tau0 [0 2 -2 0 0 0 0]   0.008006; //0.008; //0.035
        nu0  [0 2 -1 0 0 0 0]  0.00016012; // 0.0000179;
        numax   [0 2 -1 0 0 0 0]  1000e-2;
}
phases (water air);

water
{
        transportModel  Talmon;
        nu              [0 2 -1 0 0 0 0]  1e-02;
        rho             [1 -3 0 0 0 0 0] 1249;

        TalmonCoeffs
        {
                Phasename csand;

                coef [0 0 1 0 0 0 0]     50;
                cmax       cmax [0 0 0 0 0 0 0]     0.6;
                alpha0       [0 0 0 0 0 0 0]     0.27;
                tau0 [0 2 -2 0 0 0 0]   0.008006; //0.008; //0.035
            nu0      [0 2 -1 0 0 0 0]  0.00016012; // 0.0000179;
                numax   [0 2 -1 0 0 0 0]  1000e-2;
        }
}

air
{
    transportModel  CVRNewtonian;
    nu              [0 2 -1 0 0 0 0]  1.48e-05;
    rho             [1 -3 0 0 0 0 0] 1;
}

sigma           [1 0 -2 0 0 0 0] 0.07;

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

D.1.9. File constant/turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;

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

D.1.10.File system/blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
   (0 0 0)
   (20 0 0)
   (20 0.3 0)
   (0 0.3 0)
   (0 0 0.1)
   (20 0 0.1)
   (20 0.3 0.1)
   (0 0.3 0.1)
   (-1 0 0)
   (-1 0.3 0)
   (-1 0 0.1)
   (-1 0.3 0.1)
);
blocks
(
    hex (0 1 2 3 4 5 6 7) (120 80 1) simpleGrading (3 5 1)
    hex (8 0 3 9 10 4 7 11) (120 80 1) simpleGrading (1 5 1)

);
boundary
(
    leftwall
    {
        type patch;
        faces
        (
            (8 10 11 9)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 10 8)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (1 2 6 5)
         );
    }
    bottom
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    atmosphere
    {
        type wall;
        faces
        (
            (2 3 7 6)
            (3 9 11 7)
        );
    }
    front
    {
        type empty;
        faces
        (
            (4 5 6 7)
            (10 4 7 11)
        );
    }
    back
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (0 8 9 3)
        );
    }
);
mergePatchPairs();
// ************************************************************************* //
```

D.1.11.File system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoamPeter;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         2000;

deltaT          0.01;

writeControl    adjustableRunTime;

writeInterval   1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression uncompressed;

timeFormat      general;

timePrecision   6;

runTimeModifiable yes;

adjustTimeStep  yes;

maxCo           1;
maxAlphaCo      1;
maxDeltaT       1;

functions
{
    #includeFunc  singleGraph
}
// ************************************************************************* //
```

D.1.12.File system/decomposeparDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
```

```
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 6;

method          simple;

simpleCoeffs
{
    n               (6 1 1);
    delta           0.001;
}

hierarchicalCoeffs
{
    n               (2 2 1);
    delta           0.001;
    order           xyz;
}

manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );

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

D.1.13.File system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}
gradSchemes
{
    default         Gauss linear;
}
divSchemes
{
    default             none;

    div(rhoPhi,U)       Gauss linearUpwind grad(U);
    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div((interpolate(Us)&S),csand) Gauss upwind;
    "div\(phi,(k|omega)\)"      Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         corrected;
}
```

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

D.1.14.File system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 5;
        cAlpha          1.2;

        MULESCorr       yes;
        nLimiterIter    3;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }
    csand
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.1;
        smoother        GaussSeidel;
    }
    csandFinal
    {
        $csand;
        tolerance       5e-9;
        relTol          0;
    }

    "pcorr.*"
    {
        solver          PCG;
        preconditioner
        {
            preconditioner  GAMG;
            tolerance       1e-5;
            relTol          0;
            smoother        GaussSeidel;
        }
        tolerance       1e-5;
        relTol          0;
        maxIter         50;
    }

    p_rgh
    {
        solver           GAMG;
        tolerance        5e-9;
        relTol           0.01;
        smoother         GaussSeidel;
        maxIter          50;
    };

    p_rghFinal
    {
        $p_rgh;
        tolerance       5e-9;
        relTol          0;
    }

    "(U).*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        nSweeps         1;
        tolerance       1e-6;
        relTol          0.1;
    };
}

PIMPLE
{
    momentumPredictor no;
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    equations
    {
        ".*" 1;
    }
}
// ************************************************************************* //
```

D.1.15.File system/setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-0.25 0 0) (15 0.05 0.1);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
);
// ************************************************************************* //
```

D.1.16.File system/singlegraph

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Web:      www.OpenFOAM.org
     \\/     M anipulation  |
-------------------------------------------------------------------------------
Description
    Writes graph data for specified fields along a line, specified by start
    and end points.

\*---------------------------------------------------------------------------*/

start   (19.5 0 0.05);
end     (19.5 0.3 0.05);
fields  (U alpha.water);


// Sampling and I/O settings
#includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"

// Override settings here, e.g.
// setConfig { type midPoint; }

// Must be last entry
#includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
```

```

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

D.2. Simulation 2 and 3

D.2.1. File 0/alpha.water

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1;
    }
    bottom
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
        value           uniform 0;
    }
    leftwall
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.2.2. File 0/csand

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      csand;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.12;
    }
    leftwall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    bottom
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.2.3. File 0/p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

boundaryField
{
    atmosphere
    {
        type            totalPressure;
        p0              uniform 0;
    }
    ".*"
    {
        type            fixedFluxPressure;
        value           uniform 0;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.2.4. File 0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];
```

```

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.004;
    }
    bottom
    {
        type            noSlip;
    }
    leftwall
    {
        type            noSlip;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}


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

D.2.5. File 0/Us

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      Us;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.004;
    }
    leftwall
    {
        type            noSlip;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
        value           uniform (0 0 0);
    }
    bottom
    {
        type            noSlip;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}
```

```

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

D.2.6. File 0/wsvol

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      wsvol;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    leftwall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            zeroGradient;
    }
    bottom
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.2.7. File constant/g

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0.4898 -9.80 0);

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

D.2.8. File constant/transportProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Diam              Diam [ 0 1 0 0 0 0 0 ] 188e-06;

rhos              rhos [ 1 -3 0 0 0 0 0 ] 2650;

rhow              rhow [ 1 -3 0 0 0 0 0 ] 1000;

cfine             cfine [ 0 0 0 0 0 0 0 ] 0.151;

cmax              cmax [ 0 0 0 0 0 0 0 ] 0.6;

TalmonCoeffs
{
        Phasename csand;
        coef [0 0 1 0 0 0 0]     50;
        cmax      cmax [0 0 0 0 0 0 0]     0.6;
        alpha0    [0 0 0 0 0 0 0]     0.27;
        tau0       [0 2 -2 0 0 0 0]   0.037870; //=47.3/1249
        nu0        [0 2 -1 0 0 0 0]  1.71337e-5; // =0.0214/1249
        numax     [0 2 -1 0 0 0 0]  1000e-2;
}

phases (water air);

water
{
        transportModel  Talmon;
        nu              [0 2 -1 0 0 0 0]  1e-02;
        rho             [1 -3 0 0 0 0 0] 1249;

        TalmonCoeffs
        {
                Phasename csand;
                coef [0 0 1 0 0 0 0]     50;
                cmax       cmax [0 0 0 0 0 0 0]     0.6;
                alpha0        [0 0 0 0 0 0 0]     0.27;
                tau0  [0 2 -2 0 0 0 0]   0.037870; //=47.3/1249
                nu0  [0 2 -1 0 0 0 0]  1.71337e-5; // =0.0214/1249
                numax   [0 2 -1 0 0 0 0]  1000e-2;
        }
}
air
{
    transportModel  CVRNewtonian;
    nu              [0 2 -1 0 0 0 0]  1.48e-05;
    rho             [1 -3 0 0 0 0 0] 1;
}

sigma           [1 0 -2 0 0 0 0] 0.07;

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

D.2.9. File constant/turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;

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

D.2.10.File system/blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
   (0 0 0)
   (20 0 0)
   (20 0.3 0)
   (0 0.3 0)
   (0 0 0.1)
   (20 0 0.1)
   (20 0.3 0.1)
   (0 0.3 0.1)
   (-1 0 0)
   (-1 0.3 0)
   (-1 0 0.1)
   (-1 0.3 0.1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (120 80 1) simpleGrading (3 5 1)
    hex (8 0 3 9 10 4 7 11) (120 80 1) simpleGrading (1 5 1)

);

boundary
(
    leftwall
    {
        type patch;
        faces
        (
            (8 10 11 9)
        );
    }
    inlet
    {
        type patch;
        faces
        (
            (0 4 10 8)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (1 2 6 5)
        );
    }
    bottom
    {
        type wall;
        faces
        (
            (1 5 4 0)
        );
    }
    atmosphere
    {
        type wall;
        faces
        (
            (2 3 7 6)
            (3 9 11 7)
        );
    }
    front
    {
        type empty;
        faces
        (
            (4 5 6 7)
            (10 4 7 11)
        );
    }
    back
    {
        type empty;
        faces
        (
            (0 3 2 1)
            (0 8 9 3)
        );
    }
);

mergePatchPairs();

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

D.2.11.File system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoamPeter;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1200;

deltaT          0.01;

writeControl    adjustableRunTime;

writeInterval   5;

purgeWrite      0;

writeFormat     ascii;

writePrecision  8;

writeCompression uncompressed;

timeFormat      general;

timePrecision   8;

runTimeModifiable yes;

adjustTimeStep  yes;

maxCo           1;
maxAlphaCo      1;
maxDeltaT       1;

functions
{
    #includeFunc  singleGraph

    writeFields
    {
        type writeObjects;
        functionObjectLibs ("libutilityFunctionObjects.so");
        objects
        (
            rho
        );
        writeControl outputTime;
        writeInterval 5;
    }
}

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

D.2.12.File system/decomposeparDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 6;

method          simple;

simpleCoeffs
{
    n               (6 1 1);
    delta           0.001;
}

hierarchicalCoeffs
{
    n               (2 2 1);
    delta           0.001;
    order           xyz;
}

manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );


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

D.2.13.File system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

```

```
divSchemes
{
    default             none;

    div(rhoPhi,U)       Gauss linearUpwind grad(U);
    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div((interpolate(Us)&S),csand) Gauss upwind;
    "div\(phi,(k|omega)\)"      Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

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

D.2.14.File system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 5;
        cAlpha          1.2;

        MULESCorr       yes;
        nLimiterIter    3;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }
    csand
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.1;
        smoother        GaussSeidel;
    }
    csandFinal
    {
        $csand;
        tolerance       5e-9;
        relTol          0;
    }

    "pcorr.*"
    {
        solver          PCG;
        preconditioner
        {
            preconditioner  GAMG;
            tolerance       1e-5;
            relTol          0;
            smoother        GaussSeidel;
        }
        tolerance       1e-5;
        relTol          0;
        maxIter         50;
    }

    p_rgh
    {
        solver           GAMG;
        tolerance        5e-9;
        relTol           0.01;
        smoother         GaussSeidel;
        maxIter          50;
    };

    p_rghFinal
    {
        $p_rgh;
        tolerance       5e-9;
        relTol          0;
    }

    "(U).*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        nSweeps         1;
        tolerance       1e-6;
        relTol          0.1;
    };
}

PIMPLE
{
    momentumPredictor no;
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    equations
    {
        ".*" 1;
    }
}
// ************************************************************************* //
```

D.2.15.File system/setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-0.25 0 0) (15 0.05 0.1);
        fieldValues
        (
            volScalarFieldValue alpha.water 0
        );
    }
);


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

D.2.16.File system/singlegraph

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Web:      www.OpenFOAM.org
     \\/     M anipulation  |
-------------------------------------------------------------------------------
Description
    Writes graph data for specified fields along a line, specified by start
    and end points.

\*---------------------------------------------------------------------------*/

start   (15 0 0.05);
end     (15 0.3 0.05);
fields  (U alpha.water csand);


// Sampling and I/O settings
#includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"

// Override settings here, e.g.
// setConfig { type midPoint; }

// Must be last entry
#includeEtc "caseDicts/postProcessing/graphs/graph.cfg"

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

D.3. Simulation 4

D.3.1. File 0/alpha.water

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1;
    }
    leftWall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.3.2. File 0/csand – Simulation 4a

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      csand;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    leftWall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.3.3. File 0/csand – Simulation 4b and 4c

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      csand;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.12;
    }
    leftWall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.3.4. File 0/p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

boundaryField
{
    outlet
    {
        type            prghTotalPressure;
        p0              uniform 0;
    }
    ".*"
    {
        type            fixedFluxPressure;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.3.5. File 0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            noSlip;
    }
    outlet
    {
        type            pressureInletOutletVelocity;
        phi             phi;
        value           uniform (0 0 0);
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.3.6. File 0/Us

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      Us;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            noSlip;
    }
    outlet
    {
        type            pressureInletOutletVelocity;
        phi             phi;
        value           uniform (0 0 0);
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.3.7. File 0/wsvol

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      wsvol;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    leftWall
    {
        type            noSlip;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.3.8. File constant/g

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.76646 0.92320);

// 5.4 deg (0 -9.76646 0.92320)
// 5.0 deg (0 -9.77267 0.85499)
// 4.5 deg (0 -9.77976 0.76968)

// 2.86deg (0 -9.79778 0.48948)

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

D.3.9. File constant/transportProperties – Simulation 4a and 4b

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Diam              Diam [ 0 1 0 0 0 0 0 ] 188e-06;    //188

rhos              rhos [ 1 -3 0 0 0 0 0 ] 2650;

rhow              rhow [ 1 -3 0 0 0 0 0 ] 1000;

cfine             cfine [ 0 0 0 0 0 0 0 ] 0.151;

cmax              cmax [ 0 0 0 0 0 0 0 ] 0.6;

TalmonCoeffs
{
       Phasename csand;
    coef    [0 0 1 0 0 0 0]     50;
```

```
        cmax       cmax [0 0 0 0 0 0 0]     0.6;
        alpha0       [0 0 0 0 0 0 0]     0.27;
    tau0     [0 2 -2 0 0 0 0]   0.0363008; //=47.3/1303
    nu0      [0 2 -1 0 0 0 0]  1.64236e-5; // =0.0214/1303
        numax   [0 2 -1 0 0 0 0]  1000e-2;
}

phases (water air);

water
{
        transportModel  Talmon;
        nu              [0 2 -1 0 0 0 0] 1e-02;
        rho             [1 -3 0 0 0 0 0] 1303;

        TalmonCoeffs
        {
                Phasename csand;
                coef [0 0 1 0 0 0 0]     50;
                cmax       cmax [0 0 0 0 0 0 0]     0.6;
                alpha0       [0 0 0 0 0 0 0]     0.27;
                tau0 [0 2 -2 0 0 0 0]   0.0363008; //=47.3/1303
                nu0  [0 2 -1 0 0 0 0]  1.64236e-05; // =0.0214/1303
                numax   [0 2 -1 0 0 0 0]  1000e-2;
        }
}

air
{
    transportModel  CVRNewtonian;
    nu              [0 2 -1 0 0 0 0] 1.48e-05;
    rho             [1 -3 0 0 0 0 0] 1;
}

sigma           [1 0 -2 0 0 0 0] 0.07;

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

D.3.10.File constant/transportProperties – Simulation 4c

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Diam              Diam [ 0 1 0 0 0 0 0 ] 188e-06;    //188

rhos              rhos [ 1 -3 0 0 0 0 0 ] 2650;

rhow              rhow [ 1 -3 0 0 0 0 0 ] 1000;

cfine             cfine [ 0 0 0 0 0 0 0 ] 0.151;

cmax              cmax [ 0 0 0 0 0 0 0 ] 0.6;

TalmonCoeffs
{
       Phasename csand;
    coef     [0 0 1 0 0 0 0]     50;
        cmax       cmax [0 0 0 0 0 0 0]     0.6;
        alpha0       [0 0 0 0 0 0 0]     0.27;
    tau0     [0 2 -2 0 0 0 0]   0.0313245; //=47.3/1510
    nu0      [0 2 -1 0 0 0 0]  1.4172185e-05; // =0.0214/1510
        numax   [0 2 -1 0 0 0 0]  1000e-2;
}

phases (water air);

water
{
        transportModel  Talmon;
        nu              [0 2 -1 0 0 0 0] 1e-02;
        rho             [1 -3 0 0 0 0 0] 1510;

        TalmonCoeffs
        {
                Phasename csand;
                coef [0 0 1 0 0 0 0]     50;
                cmax       cmax [0 0 0 0 0 0 0]     0.6;
                alpha0       [0 0 0 0 0 0 0]     0.27;
                tau0 [0 2 -2 0 0 0 0]   0.0313245; //=47.3/1510
                nu0  [0 2 -1 0 0 0 0]  1.4172185e-05; // =0.0214/1510
                numax   [0 2 -1 0 0 0 0]  1000e-2;
        }
}

air
{
    transportModel  CVRNewtonian;
    nu              [0 2 -1 0 0 0 0] 1.48e-05;
    rho             [1 -3 0 0 0 0 0] 1;
}

sigma           [1 0 -2 0 0 0 0] 0.07;

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

D.3.11.File constant/turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;

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

D.3.12.File system/blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (-0.0261166666666667 0.0805 0)  // 0
    (0.0261166666666667 0.0805 0)  // 1
    (-0.0261166666666667 0.0522333333333333 0)  // 2
    (0.0261166666666667 0.0522333333333333 0)  // 3
    (-0.055401816305966 0.022948183694034 0)  // 4
    (0.055401816305966 0.022948183694034 0)  // 5
    (0.0783204954019061 0.0805 0)  // 6
    (-0.0783204954019061 0.0805 0)  // 7
    (-0.0261166666666667 0.108766666666667 0)  // 8
    (0.0261166666666667 0.108766666666667 0)  // 9
    (0.055401816305966 0.133751816305966 0)  // 10
    (-0.055401816305966 0.133751816305966 0)  // 11
    (-0.0261166666666667 0.0805 15)  // 12
    (0.0261166666666667 0.0805 15)  // 13
    (-0.0261166666666667 0.0522333333333333 15)  // 14
    (0.0261166666666667 0.0522333333333333 15)  // 15
    (-0.055401816305966 0.022948183694034 15)  // 16
    (0.055401816305966 0.022948183694034 15)  // 17
    (0.0783204954019061 0.0805 15)  // 18
    (-0.0783204954019061 0.0805 15)  // 19
    (-0.0261166666666667 0.108766666666667 15)  // 20
    (0.0261166666666667 0.108766666666667 15)  // 21
    (0.055401816305966 0.133751816305966 15)  // 22
    (-0.055401816305966 0.133751816305966 15)  // 23
    (-0.0261166666666667 0.0805 -2)  // 24
    (0.0261166666666667 0.0805 -2)  // 25
    (-0.0261166666666667 0.0522333333333333 -2)  // 26
    (0.0261166666666667 0.0522333333333333 -2)  // 27
    (-0.055401816305966 0.022948183694034 -2)  // 28
    (0.055401816305966 0.022948183694034 -2)  // 29
    (0.0783204954019061 0.0805 -2)  // 30
    (-0.0783204954019061 0.0805 -2)  // 31
    (-0.0261166666666667 0.108766666666667 -2)  // 32
    (0.0261166666666667 0.108766666666667 -2)  // 33
    (0.055401816305966 0.133751816305966 -2)  // 34
    (-0.055401816305966 0.133751816305966 -2)  // 35
);

edges
(
    arc 7 4 (-0.0730410843293006 0.05 0)  //
    arc 4 5 (0 0 0)  //
    arc 5 6 (0.0730410843293006 0.05 0)  //
    arc 6 10 (0.0730410843293006 0.1067 0)  //
    arc 10 11 (0 0.1567 0)  //
    arc 11 7 (-0.0730410843293006 0.1067 0)  //
    arc 19 16 (-0.0730410843293006 0.05 15)  //
    arc 16 17 (0 0 15)  //
    arc 17 18 (0.0730410843293006 0.05 15)  //
    arc 18 22 (0.0730410843293006 0.1067 15)  //
    arc 22 23 (0 0.1567 15)  //
    arc 23 19 (-0.0730410843293006 0.1067 15)  //
    arc 31 28 (-0.0730410843293006 0.05 -2)  //
    arc 28 29 (0 0 -2)  //
    arc 29 30 (0.0730410843293006 0.05 -2)  //
    arc 30 34 (0.0730410843293006 0.1067 -2)  //
    arc 34 35 (0 0.1567 -2)  //
    arc 35 31 (-0.0730410843293006 0.1067 -2)  //
);

blocks
(
    //main pipe
    hex (4 2 0 7 16 14 12 19) (10 5 40) simpleGrading (3 1 1)
    hex (5 3 2 4 17 15 14 16) (10 10 40) simpleGrading (3 1 1)
    hex (6 1 3 5 18 13 15 17) (10 5 40) simpleGrading (3 1 1)
    hex (10 9 1 6 22 21 13 18) (10 5 40) simpleGrading (3 1 1)
    hex (11 8 9 10 23 20 21 22) (10 10 40) simpleGrading (3 1 1)
    hex (7 0 8 11 19 12 20 23) (10 5 40) simpleGrading (3 1 1)
    hex (0 2 3 1 12 14 15 13) (5 10 40) simpleGrading (1 1 1)
    hex (8 0 1 9 20 12 13 21) (5 10 40) simpleGrading (1 1 1)
    //inlet

    hex (28 26 24 31 4 2 0 7) (10 5 40) simpleGrading (3 1 1)
    hex (29 27 26 28 5 3 2 4) (10 10 40) simpleGrading (3 1 1)
    hex (30 25 27 29 6 1 3 5) (10 5 40) simpleGrading (3 1 1)
    hex (34 33 25 30 10 9 1 6) (10 5 40) simpleGrading (3 1 1)
    hex (35 32 33 34 11 8 9 10) (10 10 40) simpleGrading (3 1 1)
    hex (31 24 32 35 7 0 8 11) (10 5 40) simpleGrading (3 1 1)
    hex (24 26 27 25 0 2 3 1) (5 10 40) simpleGrading (1 1 1)
    hex (32 24 25 33 8 0 1 9) (5 10 40) simpleGrading (1 1 1)

);

boundary
(
    inlet
    {
        type patch;
        faces
        (
            (29 5 4 28)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (12 14 15 13)
            (13 21 20 12)
            (16 14 12 19)
            (17 15 14 16)
            (18 13 15 17)
            (22 21 13 18)
            (23 20 21 22)
            (19 12 20 23)
        );
    }
    pipeWall
    {
        type wall;
        faces
        (
            (11 7 19 23)
            (7 4 16 19)
            (4 5 17 16)
            (5 6 18 17)
            (6 10 22 18)
            (35 31 7 11)
            (31 28 4 7)
            (29 30 6 5)
            (30 34 10 6)
            (34 35 11 10)
            (10 11 23 22)
        );
    }
    leftWall
    {
        type wall;
        faces
        (
            (24 25 27 26)
            (24 26 28 31)
            (26 27 29 28)
            (27 25 30 29)
            (24 32 33 25)
            (30 25 33 34)
            (24 31 35 32)
            (34 33 32 35)
        );
    }
);

mergePatchPairs();

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

D.3.13.File system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoamPeter;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         1000;

deltaT          0.01;

writeControl    adjustableRunTime;

writeInterval   1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  8;

writeCompression uncompressed;

timeFormat      general;

timePrecision   8;
```

```

runTimeModifiable no;

adjustTimeStep no;

maxCo           1;
maxAlphaCo      1;
maxDeltaT       1;

functions
{
    writeFields
    {
        type writeObjects;
        functionObjectLibs ("libutilityFunctionObjects.so");
        objects
        (
            nu
            nuws
            rho
        );
        writeControl outputTime;
        writeInterval 1;
    }
    interfaceHeight1
    {
        type           interfaceHeight;
        libs           ("libfieldFunctionObjects.so");
        alpha          alpha.water;
        locations      ((0 0 0) (0 0 10) (0 0 12.5) (0 0 15));
    }
}
// ************************************************************************* //
```

D.3.14.File system/decomposeparDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 5;

method          simple;

simpleCoeffs
{
    n               (1 1 5);
    delta           0.001;
}

hierarchicalCoeffs
{
    n               (1 1 1);
    delta           0.001;
    order           xyz;
}

manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );


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

D.3.15.File system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default             none;
    div(rhoPhi,U)       Gauss linearUpwind grad(U);
    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div((interpolate(Us)&S),csand) Gauss upwind;
    "div\(phi,(k|omega)\)"      Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

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

D.3.16.File system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 5;
        cAlpha          1.2;

        MULESCorr       yes;
        nLimiterIter    3;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }
    csand
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.1;
        smoother        GaussSeidel;
    }
    csandFinal
    {
        $csand;
        tolerance       5e-9;
        relTol          0;
    }
    "pcorr.*"
    {
        solver          PCG;
        preconditioner
        {
            preconditioner  GAMG;
            tolerance       1e-5;
            relTol          0;
            smoother        GaussSeidel;
        }
        tolerance       1e-5;
        relTol          0;
        maxIter         50;
    }
    p_rgh
    {
        solver           GAMG;
        tolerance        5e-9;
        relTol           0.01;

        smoother         GaussSeidel;



        maxIter          50;
    };
    p_rghFinal
    {
        $p_rgh;
        tolerance       5e-9;
        relTol          0;
    }
    "(U).*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        nSweeps         1;
        tolerance       1e-6;
        relTol          0.1;
    };
}

PIMPLE
{
    momentumPredictor no;
    nCorrectors     2;
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    equations
    {
        ".*" 1;
    }
}

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

D.3.17.File system/setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-0.07835 0 0) (0.07835 0.07835 15);
        fieldValues
        (
            volScalarFieldValue alpha.water 0
        );
    }
);

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

D.4. Simulation 5

D.4.1. File 0/alpha.water

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1;
    }
    leftWall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.4.2. File 0/csand

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      csand;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.154;
        type            codedFixedValue;
        value           uniform 0;

        // Name of generated boundary condition
        redirectType    rampedFixedValue;

        code
        #{
            const scalar t = this->db().time().value();
            operator==(min(0.154, 0.154/100*t));
        #};
    }
    leftWall
    {
        type            zeroGradient;
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.4.3. File 0/p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedFluxPressure;
    }
    outlet
    {
        type            prghPressure;
        p               uniform 0;
    }
    leftWall
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.4.4. File 0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            noSlip;
    }
    outlet
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.4.5. File 0/Us

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      Us;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            noSlip;
    }
    outlet
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.4.6. File 0/wsvol

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      wsvol;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    leftWall
    {
        type            noSlip;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.4.7. File constant/g – Simulation 5a

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.76646 0.92320);

// 6.4 deg (0 -9.74886 1.09351)
// 5.4 deg (0 -9.76646 0.92320)
// 5.0 deg (0 -9.77267 0.85499)
// 4.5 deg (0 -9.77976 0.76968)
// 2.86deg (0 -9.79778 0.48948)

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

D.4.8. File constant/g – Simulation 5b

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.74886 1.09351);

// 6.4 deg (0 -9.74886 1.09351)
// 5.4 deg (0 -9.76646 0.92320)
// 5.0 deg (0 -9.77267 0.85499)
// 4.5 deg (0 -9.77976 0.76968)
// 2.86deg (0 -9.79778 0.48948)

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

D.4.9. File constant/g – Simulation 5c

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.72829 1.26348);

// 7.4 deg (0 -9.72829 1.26348)
// 6.4 deg (0 -9.74886 1.09351)
// 5.4 deg (0 -9.76646 0.92320)
// 5.0 deg (0 -9.77267 0.85499)
// 4.5 deg (0 -9.77976 0.76968)
// 2.86deg (0 -9.79778 0.48948)

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

D.4.10.File constant/transportProperties – Simulation 5a

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Diam              Diam [ 0 1 0 0 0 0 0 ] 188e-06;    //188

rhos              rhos [ 1 -3 0 0 0 0 0 ] 2650;

rhow              rhow [ 1 -3 0 0 0 0 0 ] 1000;

cfine             cfine [ 0 0 0 0 0 0 0 ] 0.151; // this is not getting used

cmax              cmax [ 0 0 0 0 0 0 0 ] 0.6;

TalmonCoeffs
{
        Phasename csand;
    coef     [0 0 1 0 0 0 0]     50;
        cmax       cmax [0 0 0 0 0 0 0]     0.6;
        alpha0       [0 0 0 0 0 0 0]     0.27;
    tau0     [0 2 -2 0 0 0 0]   0.036301; // =47.3/1303
    nu0      [0 2 -1 0 0 0 0]  1.64236e-5; // =0.0214/1303
        numax   [0 2 -1 0 0 0 0]  1000e-2;
}

phases (water air);

water
{
        transportModel  Talmon;
        nu              [0 2 -1 0 0 0 0] 1e-02;
        rho             [1 -3 0 0 0 0 0] 1303;

        TalmonCoeffs
        {
                Phasename csand;
                coef [0 0 1 0 0 0 0]     50;
                cmax       cmax [0 0 0 0 0 0 0]     0.6;
                alpha0       [0 0 0 0 0 0 0]     0.27;
                tau0 [0 2 -2 0 0 0 0]   0.036301; // =47.3/1303
                nu0  [0 2 -1 0 0 0 0]  1.64236e-5; // =0.0214/1303
                numax   [0 2 -1 0 0 0 0]  1000e-2;
        }
}

air
{
    transportModel  CVRNewtonian;
    nu              [0 2 -1 0 0 0 0] 1.48e-05;
    rho             [1 -3 0 0 0 0 0] 1;
}

sigma           [1 0 -2 0 0 0 0] 0.07;

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

D.4.11.File constant/turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;

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

D.4.12.File system/blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
(-0.0261166666666667 0.0805 0)  // 0
(0.0261166666666667 0.0805 0)  // 1
(-0.0261166666666667 0.0522333333333333 0)  // 2
(0.0261166666666667 0.0522333333333333 0)  // 3
(-0.055401816305966 0.022948183694034 0)  // 4
(0.055401816305966 0.022948183694034 0)  // 5
(0.0783204954019061 0.0805 0)  // 6
(-0.0783204954019061 0.0805 0)  // 7
(-0.0261166666666667 0.108766666666667 0)  // 8
(0.0261166666666667 0.108766666666667 0)  // 9
(0.055401816305966 0.133751816305966 0)  // 10
(-0.055401816305966 0.133751816305966 0)  // 11
(-0.0261166666666667 0.0805 15)  // 12
(0.0261166666666667 0.0805 15)  // 13
(-0.0261166666666667 0.0522333333333333 15)  // 14
(0.0261166666666667 0.0522333333333333 15)  // 15
(-0.055401816305966 0.022948183694034 15)  // 16
(0.055401816305966 0.022948183694034 15)  // 17
(0.0783204954019061 0.0805 15)  // 18
(-0.0783204954019061 0.0805 15)  // 19
(-0.0261166666666667 0.108766666666667 15)  // 20
(0.0261166666666667 0.108766666666667 15)  // 21
(0.055401816305966 0.133751816305966 15)  // 22
(-0.055401816305966 0.133751816305966 15)  // 23
);

edges
(
    arc 7 4 (-0.0730410843293006 0.05 0)  //
    arc 4 5 (0 0 0)  //
    arc 5 6 (0.0730410843293006 0.05 0)  //
    arc 6 10 (0.0730410843293006 0.1067 0)  //
    arc 10 11 (0 0.1567 0)  //
    arc 11 7 (-0.0730410843293006 0.1067 0)  //
    arc 19 16 (-0.0730410843293006 0.05 15)  //
    arc 16 17 (0 0 15)  //
    arc 17 18 (0.0730410843293006 0.05 15)  //
    arc 18 22 (0.0730410843293006 0.1067 15)  //
    arc 22 23 (0 0.1567 15)  //
    arc 23 19 (-0.0730410843293006 0.1067 15)  //
);

blocks
(
    hex (4 2 0 7 16 14 12 19) (10 5 40) simpleGrading (3 1 1)
    hex (5 3 2 4 17 15 14 16) (10 10 40) simpleGrading (3 1 1)
    hex (6 1 3 5 18 13 15 17) (10 5 40) simpleGrading (3 1 1)
    hex (10 9 1 6 22 21 13 18) (10 5 40) simpleGrading (3 1 1)
    hex (11 8 9 10 23 20 21 22) (10 10 40) simpleGrading (3 1 1)
    hex (7 0 8 11 19 12 20 23) (10 5 40) simpleGrading (3 1 1)
    hex (0 2 3 1 12 14 15 13) (5 10 40) simpleGrading (1 1 1)
    hex (8 0 1 9 20 12 13 21) (5 10 40) simpleGrading (1 1 1)
);

boundary
(
    inlet
    {
        type patch;
        faces
        (
            (0 1 3 2)
            (0 2 4 7)
            (2 3 5 4)
            (3 1 6 5)
        );
    }
    leftWall
    {
        type patch;
        faces
        (
            (0 8 9 1)
            (6 1 9 10)
            (0 7 11 8)
            (10 9 8 11)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (12 14 15 13)
            (13 21 20 12)
            (16 14 12 19)
            (17 15 14 16)
            (18 13 15 17)
            (22 21 13 18)
            (23 20 21 22)
            (19 12 20 23)
        );
    }
    pipeWall
    {
        type wall;
        faces
        (
            (11 7 19 23)
            (7 4 16 19)
            (4 5 17 16)
            (5 6 18 17)
            (6 10 22 18)
            (10 11 23 22)
        );
    }
);

mergePatchPairs();

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

D.4.13.File system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoamPeter;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         600;

deltaT          0.005;

writeControl    adjustableRunTime; // adjustableRunTime  // timeStep

writeInterval   1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  8;

writeCompression uncompressed;

timeFormat      general;
```

```

timePrecision   8;

runTimeModifiable no;

adjustTimeStep no;

maxCo           0.5;
maxAlphaCo      0.5;
maxDeltaT       0.5;

functions
{
    writeFields
    {
        type writeObjects;
        functionObjectLibs ("libutilityFunctionObjects.so");
        objects
        (
            nu
            nuws
            rho
            rho_cf
        );
        writeControl outputTime;
        writeInterval 1;
    }
    interfaceHeight1
    {
        type           interfaceHeight;
        libs           ("libfieldFunctionObjects.so");
        alpha          alpha.water;
        locations      ((0 0 0) (0 0 10) (0 0 12.5) (0 0 15));
    }
}

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

D.4.14.File system/decomposeparDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 5;

method          simple;

simpleCoeffs
{
    n               (1 1 5);
    delta           0.001;
}
hierarchicalCoeffs
{
    n               (1 1 1);
    delta           0.001;
    order           xyz;
}
manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );

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

D.4.15.File system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default             none;

    div(rhoPhi,U)       Gauss linearUpwind grad(U);
    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div((interpolate(Us)&S),csand) Gauss upwind;
    "div\(phi,(k|omega)\)"      Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

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

D.4.16.File system/fvSolution – Simulation 5a and 5b

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 5;
        cAlpha          1.2;

        MULESCorr       yes;
        nLimiterIter    3;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }
    csand
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.1;
        smoother        GaussSeidel;
    }
    csandFinal
    {
        $csand;
        tolerance       5e-9;
        relTol          0;
    }
    "pcorr.*"
    {
        solver          PCG;
        preconditioner
        {
            preconditioner  GAMG;
            tolerance       1e-5;
            relTol          0;
            smoother        GaussSeidel;
        }
        tolerance       1e-5;
        relTol          0;
        maxIter         50;
    }
    p_rgh
    {
        solver           GAMG;
        tolerance        5e-9;
        relTol           0.01;

        smoother         GaussSeidel;

        maxIter          50;
    };
    p_rghFinal
    {
        $p_rgh;
        tolerance       5e-9;
        relTol          0;
    }
    "U"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        nSweeps         1;
        tolerance       1e-6;
        relTol          0.1;
    };
}

PIMPLE
{
    momentumPredictor no;
    nCorrectors     2;
    //nOuterCorrectors 2;
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    equations
    {
        ".*" 1;
    }
}

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

D.4.17.File system/fvSolution – Simulation 5c

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 5;
        cAlpha          1.2;

        MULESCorr       yes;
        nLimiterIter    3;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }
    csand
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.1;
        smoother        GaussSeidel;
    }
    csandFinal
    {
        $csand;
        tolerance       5e-9;
        relTol          0;
    }
    "pcorr.*"
    {
        solver          PCG;
        preconditioner
        {
            preconditioner  GAMG;
            tolerance       1e-5;
            relTol          0;
            smoother        GaussSeidel;
        }
        tolerance       1e-5;
        relTol          0;
        maxIter         50;
    }
    p_rgh
    {
        solver           GAMG;
        tolerance        5e-9;
        relTol           0.01;

        smoother         GaussSeidel;

        maxIter          50;
    };
    p_rghFinal
    {
        $p_rgh;
        tolerance       5e-9;
        relTol          0;
    }
    "U"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        nSweeps         1;
        tolerance       1e-6;
        relTol          0.1;
    };
}

PIMPLE
{
    momentumPredictor no;
    nCorrectors     2;
    nOuterCorrectors 2;
    nNonOrthogonalCorrectors 0;
}

relaxationFactors
{
    equations
    {
        ".*" 1;
    }
}

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

D.4.18.File system/setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-0.07835 0 0) (0.07835 0.07835 15);
        fieldValues
        (
            volScalarFieldValue alpha.water 0
        );
    }
);


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

D.5. Simulation 6

D.5.1. File 0/alpha.water

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1;
    }
    leftWall
    {
        type            fixedValue;
        value           uniform 0;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.5.2. File 0/csand

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      csand;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }
    leftWall
    {
        type            fixedValue;
        value           uniform 0;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.5.3. File 0/p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

boundaryField
{
    leftWall
    {
        type            prghTotalPressure;
        p0              uniform 0;
    }
    ".*"
    {
        type            fixedFluxPressure;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.5.4. File 0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.5.5. File 0/Us

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      Us;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.5.6. File 0/wsvol

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      wsvol;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    leftWall
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.5.7. File constant/g

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.76646 0.92320);

// 6.4 deg (0 -9.74886 1.09351)
// 5.4 deg (0 -9.76646 0.92320)
// 5.0 deg (0 -9.77267 0.85499)
// 4.5 deg (0 -9.77976 0.76968)
// 2.86deg (0 -9.79778 0.48948)

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

D.5.8. File constant/transportProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Diam              Diam [ 0 1 0 0 0 0 0 ] 188e-06;    //188

rhos              rhos [ 1 -3 0 0 0 0 0 ] 2650;

rhow              rhow [ 1 -3 0 0 0 0 0 ] 1000;

cfine             cfine [ 0 0 0 0 0 0 0 ] 0.151; // this is not getting used

cmax              cmax [ 0 0 0 0 0 0 0 ] 0.6;

TalmonCoeffs
{
        Phasename csand;
    coef     [0 0 1 0 0 0 0]      50;
        cmax       cmax [0 0 0 0 0 0 0]     0.6;
        alpha0       [0 0 0 0 0 0 0]     0.27;
    tau0     [0 2 -2 0 0 0 0]   0.036301; // =47.3/1303
    nu0      [0 2 -1 0 0 0 0]  1.64236e-5; // =0.0214/1303
        numax   [0 2 -1 0 0 0 0]  1000e-2;
}

phases (water air);

water
{
        transportModel  Talmon;
        nu              [0 2 -1 0 0 0 0] 1e-02;
        rho             [1 -3 0 0 0 0 0] 1303;

        TalmonCoeffs
        {
                Phasename csand;
                coef [0 0 1 0 0 0 0]     50;
                cmax       cmax [0 0 0 0 0 0 0]     0.6;
                alpha0       [0 0 0 0 0 0 0]     0.27;
                tau0 [0 2 -2 0 0 0 0]   0.036301; // =47.3/1303
                nu0  [0 2 -1 0 0 0 0]  1.64236e-5; // =0.0214/1303
                numax   [0 2 -1 0 0 0 0]  1000e-2;
        }
}
air
{
    transportModel  CVRNewtonian;
    nu              [0 2 -1 0 0 0 0] 1.48e-05;
    rho             [1 -3 0 0 0 0 0] 1;
}

sigma           [1 0 -2 0 0 0 0] 0.07;

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

D.5.9. File constant/turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;

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

D.5.10.File system/blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
    (-0.0261166666666667 0.0805 0)  // 0
    (0.0261166666666667 0.0805 0)  // 1
    (-0.0261166666666667 0.0522333333333333 0)  // 2
    (0.0261166666666667 0.0522333333333333 0)  // 3
    (-0.055401816305966 0.022948183694034 0)  // 4
    (0.055401816305966 0.022948183694034 0)  // 5
    (0.0783204954019061 0.0805 0)  // 6
    (-0.0783204954019061 0.0805 0)  // 7
    (-0.0261166666666667 0.108766666666667 0)  // 8
    (0.0261166666666667 0.108766666666667 0)  // 9
    (0.055401816305966 0.133751816305966 0)  // 10
    (-0.055401816305966 0.133751816305966 0)  // 11
    (-0.0261166666666667 0.0805 15)  // 12
    (0.0261166666666667 0.0805 15)  // 13
    (-0.0261166666666667 0.0522333333333333 15)  // 14
    (0.0261166666666667 0.0522333333333333 15)  // 15
    (-0.055401816305966 0.022948183694034 15)  // 16
    (0.055401816305966 0.022948183694034 15)  // 17
    (0.0783204954019061 0.0805 15)  // 18
    (-0.0783204954019061 0.0805 15)  // 19
    (-0.0261166666666667 0.108766666666667 15)  // 20
    (0.0261166666666667 0.108766666666667 15)  // 21
    (0.055401816305966 0.133751816305966 15)  // 22
    (-0.055401816305966 0.133751816305966 15)  // 23
);

edges
(
    arc 7 4 (-0.0730410843293006 0.05 0)  //
    arc 4 5 (0 0 0)  //
    arc 5 6 (0.0730410843293006 0.05 0)  //
    arc 6 10 (0.0730410843293006 0.1067 0)  //
    arc 10 11 (0 0.1567 0)  //
    arc 11 7 (-0.0730410843293006 0.1067 0)  //
    arc 19 16 (-0.0730410843293006 0.05 15)  //
    arc 16 17 (0 0 15)  //
    arc 17 18 (0.0730410843293006 0.05 15)  //
    arc 18 22 (0.0730410843293006 0.1067 15)  //
    arc 22 23 (0 0.1567 15)  //
    arc 23 19 (-0.0730410843293006 0.1067 15)  //
);

```

```
blocks
(
    hex (4 2 0 7 16 14 12 19) (10 5 40) simpleGrading (3 1 1)
    hex (5 3 2 4 17 15 14 16) (10 10 40) simpleGrading (3 1 1)
    hex (6 1 3 5 18 13 15 17) (10 5 40) simpleGrading (3 1 1)
    hex (10 9 1 6 22 21 13 18) (10 5 40) simpleGrading (3 1 1)
    hex (11 8 9 10 23 20 21 22) (10 10 40) simpleGrading (3 1 1)
    hex (7 0 8 11 19 12 20 23) (10 5 40) simpleGrading (3 1 1)
    hex (0 2 3 1 12 14 15 13) (5 10 40) simpleGrading (1 1 1)
    hex (8 0 1 9 20 12 13 21) (5 10 40) simpleGrading (1 1 1)
);

boundary
(
    inlet
    {
        type patch;
        faces
        (
            (0 1 3 2)
            (0 2 4 7)
            (2 3 5 4)
            (3 1 6 5)
        );
    }
    leftWall
    {
        type patch;
        faces
        (
            (0 8 9 1)
            (6 1 9 10)
            (0 7 11 8)
            (10 9 8 11)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (12 14 15 13)
            (13 21 20 12)
            (16 14 12 19)
            (17 15 14 16)
            (18 13 15 17)
            (22 21 13 18)
            (23 20 21 22)
            (19 12 20 23)
        );
    }
    pipeWall
    {
        type wall;
        faces
        (
            (11 7 19 23)
            (7 4 16 19)
            (4 5 17 16)
            (5 6 18 17)
            (6 10 22 18)
            (10 11 23 22)
        );
    }
);

mergePatchPairs();

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

D.5.11.File system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
```

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

application     interFoamPeter;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         300;

deltaT          0.005;

writeControl    timeStep; // adjustableRunTime  // timeStep

writeInterval   1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  8;

writeCompression uncompressed;

timeFormat      general;

timePrecision   8;

runTimeModifiable no;

adjustTimeStep no;

maxCo           0.5;
maxAlphaCo      0.5;
maxDeltaT       0.5;

functions
{
    writeFields
    {
        type writeObjects;
        functionObjectLibs ("libutilityFunctionObjects.so");
        objects
        (
            nu
            nuws
            rho
            rho_cf
        );
        writeControl outputTime;
        writeInterval 1;
    }
    interfaceHeight1
    {
        type            interfaceHeight;
        libs            ("libfieldFunctionObjects.so");
        alpha           alpha.water;
        locations       ((0 0 0) (0 0 10) (0 0 12.5) (0 0 15));
    }
}
// ************************************************************************* //
```

D.5.12.File system/decomposeparDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 5;

method          simple;

```

```
simpleCoeffs
{
    n               (1 1 5);
    delta           0.001;
}
hierarchicalCoeffs
{
    n               (1 1 1);
    delta           0.001;
    order           xyz;
}
manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );

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

D.5.13.File system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default             none;

    div(rhoPhi,U)       Gauss linearUpwind grad(U);
    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div((interpolate(Us)&S),csand) Gauss upwind;
    "div\(phi,(k|omega)\)"      Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
    default         Gauss linear corrected;
}
interpolationSchemes
{
    default         linear;
}
snGradSchemes
{
    default         corrected;
}

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

D.5.14.File system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
```

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 5;
        cAlpha          1.2;

        MULESCorr       yes;
        nLimiterIter    3;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }
    csand
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.1;
        smoother        GaussSeidel;
    }
    csandFinal
    {
        $csand;
        tolerance       5e-9;
        relTol          0;
    }

    "pcorr.*"
    {
        solver          PCG;
        preconditioner
        {
            preconditioner  GAMG;
            tolerance       1e-5;
            relTol          0;
            smoother        GaussSeidel;
        }
        tolerance       1e-5;
        relTol          0;
        maxIter         50;
    }
    p_rgh
    {
        solver           GAMG;
        tolerance        5e-9;
        relTol           0.01;

        smoother         GaussSeidel;
        maxIter          50;
    };

    p_rghFinal
    {
        $p_rgh;
        tolerance       5e-9;
        relTol          0;
    }

    "U"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        nSweeps         1;
        tolerance       1e-6;
        relTol          0.1;
    };
}

PIMPLE
{
    momentumPredictor no;
    nCorrectors     5;
    nNonOrthogonalCorrectors 0;
}

```

```
relaxationFactors
{
    equations
    {
        ".*" 1;
    }
}

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

D.5.15.File system/setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-0.07835 0 0) (0.07835 0.07835 15);
        fieldValues
        (
            volScalarFieldValue alpha.water 0
        );
    }
);
// ************************************************************************* //
```

D.6. Simulation 7

D.6.1. File 0/alpha.water

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      alpha.water;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 1;
    }
    leftWall
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.6.2. File 0/csand

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      csand;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.154;
    }
    leftWall
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            zeroGradient;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.6.3. File 0/p_rgh

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                     |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    location    "0";
    object      p_rgh;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

boundaryField
{
    atmosphere
    {
        type            prghPressure;
        p               uniform 0;
    }
    ".*"
    {
        type            fixedFluxPressure;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.6.4. File 0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            noSlip;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.6.5. File 0/Us

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      Us;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            flowRateInletVelocity;
        volumetricFlowRate constant 0.005;
    }
    leftWall
    {
        type            noSlip;
    }
    atmosphere
    {
        type            pressureInletOutletVelocity;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.6.6. File 0/wsvol

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    location    "0";
    object      wsvol;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    leftWall
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            zeroGradient;
    }
    pipeWall
    {
        type            noSlip;
    }
    defaultFaces
    {
        type            empty;
    }
}

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

D.6.7. File constant/g

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       uniformDimensionedVectorField;
    location    "constant";
    object      g;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -2 0 0 0 0];
value           (0 -9.76646 0.92320);

// 6.4 deg (0 -9.74886 1.09351)
// 5.4 deg (0 -9.76646 0.92320)
// 5.0 deg (0 -9.77267 0.85499)
// 4.5 deg (0 -9.77976 0.76968)
// 2.86deg (0 -9.79778 0.48948)

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

D.6.8. File constant/transportProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

Diam              Diam [ 0 1 0 0 0 0 0 ] 188e-06;    //188

rhos              rhos [ 1 -3 0 0 0 0 0 ] 2650;

rhow              rhow [ 1 -3 0 0 0 0 0 ] 1000;

cfine             cfine [ 0 0 0 0 0 0 0 ] 0.151; // this is not getting used

cmax              cmax [ 0 0 0 0 0 0 0 ] 0.6;

TalmonCoeffs
{
       Phasename csand;
    coef     [0 0 1 0 0 0 0]     50;
        cmax       cmax [0 0 0 0 0 0 0]      0.6;
        alpha0       [0 0 0 0 0 0 0]     0.27;
    tau0     [0 2 -2 0 0 0 0]   0.036301; // =47.3/1303
    nu0      [0 2 -1 0 0 0 0]  1.64236e-5; // =0.0214/1303
        numax   [0 2 -1 0 0 0 0]  1000e-2;
}

phases (water air);

water
```

```
{
        transportModel  Talmon;
        nu              [0 2 -1 0 0 0 0] 1e-02;
        rho             [1 -3 0 0 0 0 0] 1303;

        TalmonCoeffs
        {
                Phasename csand;
                coef [0 0 1 0 0 0 0]     50;
                cmax       cmax [0 0 0 0 0 0 0]     0.6;
                alpha0        [0 0 0 0 0 0 0]     0.27;
                tau0 [0 2 -2 0 0 0 0]   0.036301; // =47.3/1303
                nu0  [0 2 -1 0 0 0 0]  1.64236e-5; // =0.0214/1303
                numax   [0 2 -1 0 0 0 0]  1000e-2;
        }
}

air
{
    transportModel  CVRNewtonian;
    nu              [0 2 -1 0 0 0 0] 1.48e-05;
    rho             [1 -3 0 0 0 0 0] 1;
}

sigma           [1 0 -2 0 0 0 0] 0.07;


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

D.6.9. File constant/turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  laminar;

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

D.6.10.File system/blockMeshDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1;

vertices
(
(-0.0261166666666667 0.07835 0)  // 0
(0.0261166666666667 0.07835 0)  // 1
(-0.0261166666666667 0.0522333333333333 0)  // 2
(0.0261166666666667 0.0522333333333333 0)  // 3
(-0.055401816305966 0.022948183694034 0)  // 4
(0.055401816305966 0.022948183694034 0)  // 5
(0.07835 0.07835 0)  // 6
(-0.07835 0.07835 0)  // 7
(-0.0261166666666667 0.1567 0)  // 8
(0.0261166666666667 0.1567 0)  // 9
(0.07835 0.1567 0)  // 10
(-0.07835 0.1567 0)  // 11
```

```
(-0.0261166666666667 0.07835 15)  // 12
(0.0261166666666667 0.07835 15)  // 13
(-0.0261166666666667 0.0522333333333333 15)  // 14
(0.0261166666666667 0.0522333333333333 15)  // 15
(-0.055401816305966 0.022948183694034 15)  // 16
(0.055401816305966 0.022948183694034 15)  // 17
(0.07835 0.07835 15)  // 18
(-0.07835 0.07835 15)  // 19
(-0.0261166666666667 0.1567 15)  // 20
(0.0261166666666667 0.1567 15)  // 21
(0.07835 0.1567 15)  // 22
(-0.07835 0.1567 15)  // 23
);

edges
(
    arc 7 4 (-0.0730410843293006 0.05 0)  //
    arc 4 5 (0 0 0)  //
    arc 5 6 (0.0730410843293006 0.05 0)  //
    arc 19 16 (-0.0730410843293006 0.05 15)  //
    arc 16 17 (0 0 15)  //
    arc 17 18 (0.0730410843293006 0.05 15)  //

);
blocks
(
    hex (0 2 3 1 12 14 15 13) (5 10 40) simpleGrading (1 1 1)
    hex (4 2 0 7 16 14 12 19) (10 5 40) simpleGrading (3 1 1)
    hex (5 3 2 4 17 15 14 16) (10 10 40) simpleGrading (3 1 1)
    hex (6 1 3 5 18 13 15 17) (10 5 40) simpleGrading (3 1 1)

    hex (10 9 1 6 22 21 13 18) (10 10 40) simpleGrading (3 1 1)
    hex (7 0 8 11 19 12 20 23) (10 10 40) simpleGrading (3 1 1)
    hex (8 0 1 9 20 12 13 21) (10 10 40) simpleGrading (1 1 1)
);
boundary
(
    inlet
    {
        type patch;
        faces
        (
            (0 1 3 2)
            (0 2 4 7)
            (2 3 5 4)
            (3 1 6 5)
        );
    }
    leftWall
    {
        type patch;
        faces
        (
            (0 8 9 1)
            (6 1 9 10)
            (0 7 11 8)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (12 14 15 13)
            (12 14 16 19)
            (14 15 17 16)
            (15 13 18 17)
            (12 20 21 13)
            (18 13 21 22)
            (12 19 23 20)
        );
    }
    pipeWall
    {
        type wall;
        faces
        (
            (11 7 19 23)
            (7 4 16 19)
            (4 5 17 16)
            (5 6 18 17)
            (6 10 22 18)
        );
    }
    atmosphere
    {
        type wall;
        faces
        (
            (8 11 23 20)
            (9 8 20 21)
            (10 9 21 22)
        );
    }
);

mergePatchPairs();

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

D.6.11.File system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     interFoamPeter;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         600;

deltaT          0.01;

writeControl    timeStep; // adjustableRunTime  // timeStep

writeInterval   1;

purgeWrite      0;

writeFormat     ascii;

writePrecision  8;

writeCompression uncompressed;

timeFormat      general;

timePrecision   8;

runTimeModifiable yes;

adjustTimeStep yes;

maxCo           1;
maxAlphaCo      1;
maxDeltaT       1;

functions
{
    writeFields
    {
        type writeObjects;
        functionObjectLibs ("libutilityFunctionObjects.so");
        objects
        (
            nu
            nuws
            rho
            rho_cf
        );
        writeControl outputTime;
        writeInterval 1;
    }
    interfaceHeight1
    {
        type           interfaceHeight;
        libs           ("libfieldFunctionObjects.so");
        alpha          alpha.water;
        locations      ((0 0 0) (0 0 10) (0 0 12.5) (0 0 15));
    }
}
// ************************************************************************* //
```

D.6.12.File system/decomposeparDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 5;

method          simple;

simpleCoeffs
{
    n               (1 1 5);
    delta           0.001;
}
hierarchicalCoeffs
{
    n               (1 1 1);
    delta           0.001;
    order           xyz;
}
manualCoeffs
{
    dataFile        "";
}

distributed     no;

roots           ( );

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

D.6.13.File system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes
{
    default             none;

    div(rhoPhi,U)       Gauss linearUpwind grad(U);
    div(phi,alpha)      Gauss vanLeer;
    div(phirb,alpha)    Gauss linear;
    div((interpolate(Us)&S),csand) Gauss upwind;
    "div\(phi,(k|omega)\)"      Gauss upwind;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

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

D.6.14.File system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  4.1                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "alpha.water.*"
    {
        nAlphaCorr      1;
        nAlphaSubCycles 5;
        cAlpha          1.2;

        MULESCorr       yes;
        nLimiterIter    3;

        solver          smoothSolver;
        smoother        GaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }
    csand
    {
        solver          GAMG;
        tolerance       1e-6;
        relTol          0.1;
        smoother        symGaussSeidel;
    }
    csandFinal
    {
        $csand;
        tolerance       5e-9;
        relTol          0;
    }
    "pcorr.*"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-5;
        relTol          0;

        //preconditioner
        //{
        //    preconditioner  GAMG;
        //    tolerance       1e-5;
        //    relTol          0;
        //    smoother        GaussSeidel;
        //}
        //tolerance       1e-5;
        //relTol          0;
        //maxIter         50;
    }
    p_rgh
    {
        //solver           GAMG;
        //tolerance        5e-9;
        //relTol           0.01;
        //smoother         GaussSeidel;
        //maxIter          50;

        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-07;
        relTol          0.05;
    }
    p_rghFinal
    {
        $p_rgh;
        relTol          0;
    }
    U
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-6;
        relTol          0;
    }
    UFinal
    {
        $U;
        tolerance       5e-7;
        relTol          0;
    }
}

PIMPLE
{
    momentumPredictor           yes;
    nCorrectors                 3;
    nNonOrthogonalCorrectors    0;
    nOuterCorrectors            50;

    residualControl
    {
        p_rgh
        {
            relTol 0;
            tolerance 1e-7;
        }
        U
        {
            relTol 0;
            tolerance 1e-6;
        }
    }
}

relaxationFactors
{
    equations
    {
        ".*" 1;
    }
    fields
    {
        ".*"  1;
    }
}

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

D.6.15.File system/setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  5                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-0.07835 0 0) (0.07835 0.07835 15);
        fieldValues
        (
            volScalarFieldValue alpha.water 0
        );
    }
);

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

Appendix E. Tips

E.1.1. parallelReconstructPar

Through bash scripting, it's actually possible to run a `reconstructPar` command in parallel. This particular script was found online and has been written by K. Wardle and later improved by H. Stadler, W. Bateman and A. Shafiee.

From a command line or bash script, it can be called as follows. Note that logging the result to a separate file (`> logParallelReconstructPar`) is optional.

```
bash parallelReconstructPar.sh -n 5 > logParallelReconstructPar;
```

And this is the parallelReconstructPar.sh bash script:

```
#!/bin/bash
echo "
      K. Wardle 6/22/09, modified by H. Stadler Dec. 2013, minor fix Will Bateman Sep 2014, minor fix A. Shafiee Jul. 2017.
      bash script to run reconstructPar in pseudo-parallel mode
      by breaking time directories into multiple ranges
     "

USAGE="
      USAGE: $0 -n <NP> -f fields -o <OUTPUTFILE>
        -f (fields) is optional, fields given in the form T,U,p; option is passed on to reconstructPar
      -t (times) is optional, times given in the form tstart,tstop
        -o (output) is optional
"

#TODO: add flag to trigger deletion of original processorX directories after successful reconstruction
# At first check whether any flag is set at all, if not exit with error message
if [ $# == 0 ]; then
    echo "$USAGE"
    exit 1
fi

#Use getopts to pass the flags to variables
while getopts "f:n:o:t:" opt; do
  case $opt in
    f) if [ -n $OPTARG ]; then
  FIELDS=$(echo $OPTARG | sed 's/,/ /g')
  fi
      ;;
    n) if [ -n $OPTARG ]; then
  NJOBS=$OPTARG
  fi
      ;;
    o) if [ -n $OPTARG ]; then
  OUTPUTFILE=$OPTARG
       fi
      ;;
    t) if [ -n $OPTARG ]; then
  TLOW=$(echo $OPTARG | cut -d ',' -f1)
  THIGH=$(echo $OPTARG | cut -d ',' -f2)
  fi
      ;;
    \?)
      echo "$USAGE" >&2
      exit 1
      ;;
    :)
      echo "Option -$OPTARG requires an argument." >&2
      exit 1
      ;;
  esac
done

# check whether the number of jobs has been passed over, if not exit with error message
if [[ -z $NJOBS ]]
then
    echo "
      the flag -n <NP> is required!
       "
    echo "$USAGE"
    exit 1
fi

APPNAME="reconstructPar"
```

```

echo "running $APPNAME in pseudo-parallel mode on $NJOBS processors"

#count the number of time directories
NSTEPS=$(($(ls -d processor0/[0-9]*/ | wc -l)-1))
NINITAL=$(ls -d [0-9]*/ | wc -l) ##count time directories in case root dir, this will include 0

P=p
#find min and max time
TMIN=$(ls processor0 -1v | sed '/constant/d' | sort -g | sed -n 2$P) # modified to omit constant and first time directory
#TMIN=`ls processor0 | sort -nr | tail -1`
TMAX=$(ls processor0 -1v | sed '/constant/d' | sort -gr | head -1) # modified to omit constant directory
#TMAX=`ls processor0 | sort -nr | head -1`

#Adjust min and max time according to the parameters passed over
if [ -n "$TLOW" ]
  then
    TMIN=$(ls processor0 -1v | sed '/constant/d' | sort -g | sed -n 1$P) # now allow the first directory
    NLOW=2
    NHIGH=$NSTEPS
    # At first check whether the times are given are within the times in the directory
    if [ $(echo "$TLOW > $TMAX" | bc) == 1 ]; then
        echo "
      TSTART ($TLOW) > TMAX ($TMAX)
      Adjust times to be reconstructed!
      "
        echo "$USAGE"
        exit 1
    fi
    if [ $(echo "$THIGH < $TMIN" | bc) == 1 ]; then
        echo "
      TSTOP ($THIGH) < TMIN ($TMIN)
      Adjust times to be reconstructed!
      "
        echo "$USAGE"
        exit 1
    fi

    # Then set Min-Time
    until [ $(echo "$TMIN >= $TLOW" | bc) == 1 ]; do
      TMIN=$(ls processor0 -1v | sort -g | sed -n $NLOW$P)
      NSTART=$(($NLOW))
      let NLOW=NLOW+1
    done

    # And then set Max-Time
    until [ $(echo "$TMAX <= $THIGH" | bc) == 1 ]; do
      TMAX=$(ls processor0 -1v | sort -g | sed -n $NHIGH$P)
      let NHIGH=NHIGH-1
    done

    # Finally adjust the number of directories to be reconstructed
    NSTEPS=$(($NHIGH-$NLOW+3))

  else
    NSTART=2
fi

echo "reconstructing $NSTEPS time directories"

NCHUNK=$(($NSTEPS/$NJOBS))
NREST=$(($NSTEPS%$NJOBS))
TSTART=$TMIN

echo "making temp dir"
TEMPDIR="temp.parReconstructPar"
mkdir $TEMPDIR

PIDS=""
for i in $(seq $NJOBS)
do
  if [ $NREST -ge 1 ]
    then
      NSTOP=$(($NSTART+$NCHUNK))
      let NREST=$NREST-1
    else
      NSTOP=$(($NSTART+$NCHUNK-1))
  fi
  TSTOP=$(ls processor0 -1v | sort -g | sed -n $NSTOP$P)

  if [ $i == $NJOBS ]
  then
  TSTOP=$TMAX
  fi

  if [ $NSTOP -ge $NSTART ]
    then
    echo "Starting Job $i - reconstructing time = $TSTART through $TSTOP"
    if [ -n "$FIELDS" ]
      then
        $($APPNAME -fields "($FIELDS)" -newTimes -time $TSTART:$TSTOP > $TEMPDIR/output-$TSTOP &)
  echo "Job started with PID $(pgrep -n -x $APPNAME)"
  PIDS="$PIDS $(pgrep -n -x $APPNAME)" # get the PID of the latest (-n) job exactly matching (-x) $APPNAME
      else
        $($APPNAME -newTimes -time $TSTART:$TSTOP > $TEMPDIR/output-$TSTOP &)
  echo "Job started with PID $(pgrep -n -x $APPNAME)"
  PIDS="$PIDS $(pgrep -n -x $APPNAME)"
    fi
   fi
  let NSTART=$NSTOP+1
  TSTART=$(ls processor0 -1v | sort -g | sed -n $NSTART$P)
done

#sleep until jobs finish
#if number of jobs > NJOBS, hold loop until job finishes
NMORE_OLD=$(echo 0)
until [ $(ps -p $PIDS | wc -l) -eq 1 ]; # check for PIDS instead of $APPNAME because other instances might also be running
  do
    sleep 10
    NNOW=$(ls -d [0-9]*/ | wc -l) ##count time directories in case root dir, this will include 0
    NMORE=$(echo $NSTEPS-$NNOW+$NINITAL | bc) ##calculate number left to reconstruct and subtract 0 dir
    if [ $NMORE != $NMORE_OLD ]
      then
      echo "$NMORE directories remaining..."
    fi
    NMORE_OLD=$NMORE
  done

#combine and cleanup
if [ -n "$OUTPUTFILE" ]
  then
#check if output file already exists
  if [ -e "$OUTPUTFILE" ]
  then
    echo "output file $OUTPUTFILE exists, moving to $OUTPUTFILE.bak"
    mv $OUTPUTFILE $OUTPUTFILE.bak
  fi

  echo "cleaning up temp files"
  for i in $(ls $TEMPDIR)
  do
    cat $TEMPDIR/$i >> $OUTPUTFILE
  done
fi

rm -rf $TEMPDIR

echo "finished"
```