Cite as: Zabaleta, F.: Incorporation of Greimann and Holly interparticle stress model to sedFoam. In Proceedings of CFD with OpenSource Software, 2018, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2018

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Incorporation of Greimann and Holly interparticle stress model to sedFoam

Developed for OpenFOAM-6.0.x
Requires: swak4Foam and sedFoam

Author:
Federico Zabaleta
University of California, Davis
fzabaleta@ucdavis.edu

Peer reviewed by:
Ebrahim Ghahramani
Jiangyuan Zhang

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 3, 2019

Learning outcomes

The reader will learn:

How to use it:

- How the sedFoam solver works.
- How to use the solver.

The theory of it:

- Two-fluid flow theory.
- Interparticle stress modelling.

How it is implemented:

- How sedFoam is implemented.

How to modify it:

- How to implement new inter-particle stress models.
- The effect of the new model on the simulation of sediment distribution in an open channel.

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- Fundamentals of fluid mechanics.
- Fundamentals of two-phase flows.
- Run standard document tutorials like damBreak tutorial
- Be able to read and understand C++.

Contents

Chapter 1

Theoretical Framework

1.1 Introduction

Sediment laden flows consist of two phases (water and sediment) moving and interacting with each other. The carrier phase (water) is responsible of the motion of the dispersed phase (sediment), which at the same time affects the water flow. An ideal numerical model for the particle phase would resolve the dynamics of each individual particle in the system. However, typical engineering applications involve millions of particles, and the time and resources to account for the dynamics of every particle exceeds the computational capabilities nowadays.

First approaches to model suspended sediment in open channels considered the dispersed phase as a scalar transported by the carrier phase. This approach considers the sediment moving at the same stream-wise velocity of the fluid, and the vertical velocities of the sediment as a result of the settling velocity and the turbulent diffusion. Under these considerations, assuming a vertical sediment diffusivity proportional to the momentum diffusivity, and describing the fluid velocity distribution with the law of the wall, Rouse (1957) [4] derived an equation that describes the vertical distribution of sediment under equilibrium conditions in an open-channel, known as Rousean distribution. This equation represents a very good approximation of the shape of the sediment distribution, but fails to predict the concentration of sediment in the wall-normal direction. Good representations can be achieved by adjusting the Rouse number in order to adjust the measured data, but this equation is not capable by itself to predict the wall-normal sediment distribution. Several authors proposed methods to correct the Rousean distribution. These attempts included modification of the von-Karman constant, adjusting the values of the Schmidt number (ratio between the eddy viscosity of the flow and the suspended sediment diffusion coefficient) and including mechanisms to account for different velocities of the flow and the sediment.

A more complex method is the particle cloud approach, in which "bulk" properties are identified and interpreted in terms of mixing. In particular we are interested in the eulerian version of the particle cloud approach, also called two fluid model, where the properties of the particles are assumed to be continuous like those of a fluid. Equations are developed for the conservation of mass and momentum, and discretized into algebraic equations and then solved using the same procedures used for the carrier phase.

1.2 Two Fluid Model

Two-phase flow models consist of a set of equations of mass, momentum and energy conservations for both phases. These models treat both phases as a continuum, no matter if they really are. To use this assumption, ensemble or volume averaged variables are considered for both phases (for more information see Crowe (2012) [3]).

For a water sediment flow, mass and momentum equations account for the interaction between both phases (the energy equation is decoupled from the other two). Assuming that the pressure at

both phases is the same, averaging over turbulence, and using the Boussinesq model to represent the Reynolds stresses of the carrier and disperse phases, we can write the mass and momentum conservation equations for both phases like it is presented in Equations 1.1 and 1.2. The overbars for turbulence averaging have been dropped for simplicity.

$$\frac{\partial \alpha^k \rho^k}{\partial t} + \nabla \cdot (\alpha^k \rho^k \mathbf{U}^k) = 0 \tag{1.1}$$

$$\frac{\partial \alpha^k \rho^k \mathbf{U}^k}{\partial t} + \nabla \cdot (\alpha^k \rho^k \mathbf{U}^k \mathbf{U}^k) = -\alpha^k \nabla p + \nabla \cdot \left(\alpha^k \left[\mathbf{T}^k + \boldsymbol{\xi}^k \right] \right) + \alpha^k \rho^k \mathbf{g} + \mathbf{F}^k \tag{1.2}$$

In this equations, k indicates the phases. These are the carrier phase (water, with superscript c) and the disperse phase (sediment, with superscript d); α^k is the volume fraction of the phase k; ρ^k is the density of the phase k; $\mathbf{U}^k$ is the velocity of the phase k; t indicates time; p denotes the pressure; $\mathbf{T}^k$ represent the viscous and Reynolds stresses, $\boldsymbol{\xi}^k$ represent the stresses due to interparticle collisions ($\boldsymbol{\xi}^c = \mathbf{0}$); $\mathbf{g}$ is the gravity acceleration, and $\mathbf{F}^k$ represent all the interaction forces among phases for the phase k. This forces are typically drag, lift, virtual mass and turbulent dispersion.

The total stresses ($\mathbf{T}^k$) can be represented as the sum of the viscous stress and the Reynolds stress (Equation 1.3). Assuming a newtonian fluid to represent the viscous stresses, and using the Boussinesq approximation to estimate the Reynold stresses, they can be expressed as follows,

$$\mathbf{T}^k = \boldsymbol{\tau}^k + \mathbf{R}^k \tag{1.3}$$

$$\boldsymbol{\tau}^k = \rho^k \nu^k \left(\nabla \mathbf{U}^k + \nabla \mathbf{U}^{k^T} \right) + \frac{2}{3} \rho^k \nu^k \left(\nabla \cdot \mathbf{U}^k \right) \mathbf{I} \tag{1.4}$$

$$\boldsymbol{R}^k = \rho^k \nu_t^k \left(\nabla \mathbf{U}^k + \nabla \mathbf{U}^{k^T} \right) + \frac{2}{3} \rho^k \nu_t^k \left(\nabla \cdot \mathbf{U}^k \right) \mathbf{I} + \frac{2}{3} \rho^k K^k \mathbf{I} \tag{1.5}$$

These two stresses can be combined using an efficient viscosity $\nu_{eff} = \nu^k + \nu_t^k$. Then the total stress can be rewritten as,

$$\mathbf{T}^k = \rho^k \nu_{eff}^k \left(\nabla \mathbf{U}^k + \nabla \mathbf{U}^{k^T} \right) + \frac{2}{3} \rho^k \nu_{eff}^k \left(\nabla \cdot \mathbf{U}^k \right) \mathbf{I} + \frac{2}{3} \rho^k K^k \mathbf{I} \tag{1.6}$$

1.2.1 Phase-Intensive Formulation

The solver sedFoam uses the phase-intesive formulations of the phase momentum equations [7]. This formulation is used to avoid instabilities when $\alpha^k \to 0$. To obtain this formulation, the continuity equation is multiplied by $\mathbf{U}^k$ and subtracted from the mommentum equation. Then the new equation is divided by α^k. Assuming an incompressible flow (e.g. ρ^k constant), the phase intensive formulation of the mommentum equation (of the carrier phase, $k = c$) can be expressed as,

$$\frac{\partial \mathbf{U}^k}{\partial t} + \mathbf{U}^k \cdot \nabla \mathbf{U}^k + \nabla \cdot \frac{\mathbf{T}^k}{\rho^k} + \frac{\nabla \alpha^k}{\alpha^k} \cdot \frac{\mathbf{T}^k}{\rho^k} = -\frac{\nabla p}{\rho^k} + \mathbf{g} + \frac{\mathbf{F}^k}{\alpha^k \rho^k} \tag{1.7}$$

Then the total stress can be divided into a diffusive component and a correction term.

$$\frac{\mathbf{T}^k}{\rho^k} = -\nu_{eff}^k \nabla \mathbf{U}^k + \mathbf{T}_C^k \tag{1.8}$$

$$\mathbf{T}_C^k = -\nu_{eff}^k \nabla \mathbf{U}^{k^T} + \frac{2}{3} \nu_{eff}^k \left(\nabla \cdot \mathbf{U}^k \right) \mathbf{I} + \frac{2}{3} K^k \mathbf{I} \tag{1.9}$$

Then Equation 1.7 can be rewritten as,

$$\frac{\partial \mathbf{U}^k}{\partial t} + \mathbf{U}^k \cdot \nabla \mathbf{U}^k - \nabla \cdot (\nu_{eff}^k \nabla \mathbf{U}^k) - \frac{\nabla \alpha^k}{\alpha^k} \cdot (\nu_{eff}^k \nabla \mathbf{U}^k) + \nabla \cdot \mathbf{T}_C^k + \frac{\nabla \alpha^k}{\alpha^k} \cdot \mathbf{T}_C^k = -\frac{\nabla p}{\rho^k} + \mathbf{g} + \frac{\mathbf{F}^k}{\alpha^k \rho^k} \quad (1.10)$$

Taking into account the following identities,

$$\mathbf{U}^k \cdot \nabla \mathbf{U}^k = \nabla \cdot (\mathbf{U}^k \mathbf{U}^k) - \mathbf{U}^k \left(\nabla \cdot \mathbf{U}^k \right) \quad (1.11)$$

$$\frac{\nabla \alpha^k}{\alpha^k} \cdot (\nu_{eff}^k \nabla \mathbf{U}^k) = \nu_{eff}^k \nabla \cdot (\nabla \alpha^k \mathbf{U}^k) - \mathbf{U}^k \left(\nabla \cdot [\nu_{eff}^k \nabla \alpha^k] \right) \quad (1.12)$$

Equation 1.10 can be finally rewritten as,

$$\frac{\partial \mathbf{U}^k}{\partial t} + \nabla \cdot (\mathbf{U}^k \mathbf{U}^k) - \mathbf{U}^k \left(\nabla \cdot \mathbf{U}^k \right) - \nu_{eff}^k \nabla^2 \mathbf{U}^k - \nu_{eff}^k \nabla \cdot (\nabla \alpha^k \mathbf{U}^k) - \mathbf{U}^k \left(\nabla \cdot [\nu_{eff}^k \nabla \alpha^k] \right) = -\nabla \cdot \mathbf{T}_C^k - \frac{\nabla \alpha^k}{\alpha^k} \cdot \mathbf{T}_C^k - \frac{\nabla p}{\rho^k} + \mathbf{g} + \frac{\mathbf{F}^k}{\alpha^k \rho^k} \quad (1.13)$$

This is the final expression used in sedFoam. The left hand side is calculated implicitly, while the parts on the right hand side are calculated explicitly and transferred to the pressure Equation. An additional term that comes from the drag model is also calculated implicitly. This equation can be found in the code on the file `solvers/UEqn.H`. A similar approach can be done with the dispersed phase. A piece of that code is reproduced here,

```
surfaceScalarField phiRb =
(
  - fvc::interpolate(nuEffb)*mesh.magSf()*(fvc::snGrad(beta))
   /fvc::interpolate(beta)
);
fvVectorMatrix UbEqn
(
  fvm::ddt(Ub)
  + fvm::div(phib, Ub, "div(phib,Ub)")
  - fvm::Sp(fvc::div(phib), Ub)
  - fvm::laplacian(nuEffb, Ub)
  + fvm::div(phiRb, Ub, "div(phiRb,Ub)")
  - fvm::Sp(fvc::div(phiRb), Ub)
    ==
    ... (Corresponding to interaction forces)
);
```

1.2.2 Phase continuity equation

The continuity equation for both phases should be solved in a way that guarantee boundedness for each phase (e.g α^k should be in between 0 and 1). In order to do this, the continuity equation for both phases are combined. The total and relative velocities are defined as follows,

$$\mathbf{U} = \alpha^c \mathbf{U}^c + \alpha^d \mathbf{U}^d \quad ; \quad \mathbf{U}^r = \mathbf{U}^d - \mathbf{U}^c \quad (1.14)$$

Then the velocity of the dispersed phase can be rewritten as,

$$\mathbf{U}^d = \mathbf{U} + \alpha^c \mathbf{U}^r \tag{1.15}$$

Substituting the last expression in the continuity equation of the dispersed phase, this one can be rewritten as,

$$\frac{\partial \alpha^d}{\partial t} + \nabla \cdot (\alpha^d \mathbf{U}) + \nabla \cdot (\alpha^d (1 - \alpha^d) \mathbf{U}^r) = 0 \tag{1.16}$$

This expression is used in sedFoam to solve the continuity equation of the dispersed phase. A piece of the code showing this equation is presented below. If this equation is solved fully implicitly, then the solution should be bounded. Finally, the phase fraction of the carrier phase is calculated as $\alpha^c = 1 - \alpha^d$.

```
surfaceScalarField phi("phi", phia + phib);
surfaceScalarField phir("phir", phia - phib);

fvScalarMatrix alphaEqn
(
     fvm::ddt(alpha)
   + fvm::div(phi, alpha, scheme)
   + fvm::div(-fvc::flux(-phir, beta, schemer), alpha, schemer)
);

alphaEqn.relax();
alphaEqn.solve();

alpha.min(alphaMax);
alpha.max(0);

beta = scalar(1.0) - alpha;
```

1.3 Interaction forces

The interaction forces $\mathbf{F}^k$ can be calculated as,

$$\mathbf{F}^k = \mathbf{F}_D + \mathbf{F}_L + \mathbf{F}_{VM} + \mathbf{F}_{SUS}$$

Where the drag force ($\mathbf{F}_D$) can be calculated as [1],

$$\mathbf{F}_D = \alpha^d \beta (\mathbf{U}^c - \mathbf{U}^d) \tag{1.17}$$

$$\beta = \begin{cases} 150 \dfrac{\alpha^d \nu^c \rho^c}{\alpha^c d^2} + 1.75 \dfrac{\rho^c |\mathbf{U}^c - \mathbf{U}^d|}{d} & \alpha^c \geq 0.2 \\ \dfrac{0.75 * C_D \rho^c |\mathbf{U}^c - \mathbf{U}^d| (1 - \alpha^d)^{-1.65}}{d} & \alpha^c < 0.2 \end{cases} \tag{1.18}$$

The lift force ($\mathbf{F}_L$) can be calculated as [2],

$$\mathbf{F}_L = -C_L \alpha^d \rho^c (\mathbf{U}^c - \mathbf{U}^d) \times (\nabla \times \mathbf{U}^c) \tag{1.19}$$

$$C_L = \frac{1}{4} \tag{1.20}$$

The virtual mass force, for a spherical particle submerged in an inviscid, incompressible fluid is given by [3],

$$\mathbf{F}_{VM} = \frac{1}{2}\alpha^d \rho^c \left(\frac{D\mathbf{U}^c}{Dt} - \frac{\partial \mathbf{U}^d}{\partial t} \right) \tag{1.21}$$

$\mathbf{F}_{SUS}$ represents the turbulent resuspension term [1], and can be calculated as,

$$\mathbf{F}_{SUS} = \frac{1}{\sigma_c} \beta K \nu_t^d \nabla \alpha^d \tag{1.22}$$

1.4 Interparticle-stress

The interparticle stress occurs due to collision between particles. Probably the most used approach to model this stress is the Kinetic Theory of Granular Flows. This model includes a parameter called granular temperature which is obtained solving a partial differential equation. SedFoam recently introduced the Dense Granular Flow Rheology model to represent the interparticle stress. This model is not going to be discussed here, but more information can be found on Chauchat et. al (2017) [1].

The interparticle stress is divided into the normal (p) and off-diagonal (τ) components. These components can be divided into a collisional kinetic component (superscript ^{sc}) and a frictional component (superscript ^{sf}). They can be expressed as,

$$\boldsymbol{\xi}^d = p^d \mathbf{I} + \boldsymbol{\tau}^d = \boldsymbol{\xi}^{dc} + \boldsymbol{\xi}^{df} \tag{1.23}$$

$$p^d = p^{dc} + p^{df} \quad ; \quad \boldsymbol{\tau}^d = \boldsymbol{\tau}^{dc} + \boldsymbol{\tau}^{df} \tag{1.24}$$

The normal components can be calculated as follows,

$$p^{dc} = \begin{cases} 0 & \alpha^d < \alpha_L^d \\ F \frac{\left(\alpha^d - \alpha_L^d\right)^m}{\left(\alpha_{max}^d - \alpha^d\right)^n} & \alpha^d > \alpha_L^d \end{cases} \quad ; \quad p^{df} = \rho^d \alpha^d \left[1 + 2(1+e)\alpha^d g_{s0}\right] \Phi \tag{1.25}$$

where F, m and n are empirical coefficients, Φ is the granular temperature, and g_{s0} is defined as,

$$g_{s0} = \frac{2 - \alpha^d}{2(1-\alpha^d)^3} \tag{1.26}$$

The tangential stress components can be calculated as:

$$\boldsymbol{\tau}^{dc} = 2\mu^{dc} \mathbf{S}^d \tag{1.27}$$

$$\boldsymbol{\tau}^{df} = 2\mu^{df} \mathbf{S}^d + \lambda (\nabla \cdot \mathbf{U}^d) \tag{1.28}$$

and,

$$\mathbf{S}^d = \frac{1}{2}\left(\nabla \mathbf{U}^d + \nabla \mathbf{U}^{d^T}\right) - \frac{1}{3}\nabla \cdot \mathbf{U}^d \tag{1.29}$$

where μ^{df} is the particle frictional viscosity, μ^{dc} is the particle collisional viscosity and λ is the bulk viscosity. The first one is a function of the friction angle and the last two are functions of the granular temperature (Φ). They can be calculated as follows [1],

$$\mu^{df} = \frac{p^{df} \sin(\theta_f)}{\sqrt{||\mathbf{S}^d||}} \tag{1.30}$$

$$\mu^{dc} = \rho^d d_s \sqrt{\Phi} \left(\frac{4{\alpha^d}^2 g_{s0}(1+e)}{5\pi} + \frac{\sqrt{\pi} g_{s0}(1+e)(3e-1){\alpha^d}^2}{15(3-e)} + \frac{\sqrt{\pi}\alpha^d}{6(3-e)} \right) \tag{1.31}$$

$$\lambda = \frac{4}{3}\alpha^{d^2}\rho^d d_s g_{s0}(1+e)\sqrt{\frac{\Phi}{\pi}} \tag{1.32}$$

where θ_f is the friction angle, e is the is the coefficient of restitution during the collision and d_s is the particle diameter. The granular temperature (Φ) can be calculated with the balance equation [1],

$$\frac{3}{2}\left(\frac{\partial \alpha^d \rho^d \Phi}{\partial t} + \nabla \cdot (\alpha^d \rho^d \mathbf{U}^d \Phi)\right) = \left(-p^{dc}\mathbf{I} + \boldsymbol{\tau}^{dc}\right)\nabla \mathbf{U}^d - \nabla \cdot \mathbf{q} - \gamma_s + J_{int} \tag{1.33}$$

The closure of granular temperature flux (q_j) is assumed to be analogous to Fourier's law of conduction,

$$\mathbf{q} = -\kappa^{dc}\nabla\Phi \tag{1.34}$$

where κ^{dc} is the conductivity of granular temperature, calculated as [1],

$$\kappa^{dc} = \rho^d d_s \sqrt{\Phi}\left(\frac{2\alpha^{d^2} g_{s0}(1+e)}{\pi} + \frac{9\sqrt{\pi} g_{s0}(1+e)(2e-1)\alpha^{d^2}}{2(49-33e)} + \frac{5\sqrt{\pi}\alpha^d}{2(49-33e)}\right) \tag{1.35}$$

The dissipation rate due to inelastic collision is calculated based on that proposed by Ding and Gidaspow (1990),

$$\gamma_s = 3\left(1-e^2\right)(\alpha^d)^2 \rho^d g_{s0}\Phi\left(\frac{4}{d_s}\sqrt{\frac{\Phi}{\pi}} - \frac{\partial U_j^d}{\partial x_j}\right) \tag{1.36}$$

Due to the presence of the carrier fluid phase, carrier-flow turbulence can also induce particle fluctuations. Following Hsu et al (2004), the fluid-particle interaction term can be expressed as,

$$J_{int} = \alpha^d K\left(2t_m k^c - 3\Phi\right) \tag{1.37}$$

where t_m represents the correlation between particle and fluid velocity fluctuations and k^c the turbulent kinetic energy of the carrier phase.

1.4.1 Simplification by Greimann and Holly

In this project we are going to implement the equations proposed by Greimann and Holly [5]. These equations are a simplification of the Kinetic Theory model obtained using a mixing length approach for the particle phase. These equations have the advantage that they do not need to solve a partial differential equation to obtain the granular temperature. In this model an algebraic model for the granular temperature is obtained. They express the frictional interparticle stress ($\boldsymbol{\xi}^{dc}$) as,

$$\boldsymbol{\xi}^{dc} = -2\alpha^d\rho^d g_o(1+e)\Phi\mathbf{I} - \frac{4}{5}\alpha^d\rho^d g_0(1+e)\left(\mathbf{M}^d - \Phi\mathbf{I}\right) + \frac{4}{5}\alpha^d\rho^d d_p g_0(1+e)\sqrt{\frac{\tau}{\pi}}\left(2\mathbf{S}^d + (\nabla\cdot\mathbf{U}^d)\mathbf{I}\right) \tag{1.38}$$

where Φ is the granular temperature, $\mathbf{M}^d$ is the second order moment of the particle phase and $\mathbf{S}^d$ is the strain rate tensor. $\mathbf{M}^d$ is defined as,

$$\mathbf{M}^d = \Phi\mathbf{I} - \mathbf{S}^d \tag{1.39}$$

the rest of the parameters are defined as follows,

$$g_0 = \left(1 - \frac{\alpha^d}{\alpha^d_{max}}\right)^{-2.5\alpha^d_{max}} \quad ; \quad \Phi = \frac{2K^d}{3} \quad ; \quad K^d = \begin{array}{c}\text{Turbulent Kinetic Energy (TKE)}\\ \text{of disperse phase}\end{array} \tag{1.40}$$

Substituting Equation 1.39 in Equation 1.38 and rearranging we obtain Equation 1.41.

$$\boldsymbol{\xi}^{dc} = -2\alpha^d\rho^d g_o(1+e)\Phi\mathbf{I} + \left[\frac{4}{5}\alpha^d\rho^d g_0(1+e)\nu_T^d + \frac{4}{5}\alpha^d\rho^d d_p g_0(1+e)\sqrt{\frac{\Phi}{\pi}}\right]\left(\nabla\mathbf{U}^d + \nabla\mathbf{U}^{dT}\right) + \left[\frac{8}{15}\alpha^d\rho^d g_0(1+e)\nu_T^d + \frac{4}{5}\alpha^d\rho^d d_p g_0(1+e)\sqrt{\frac{\Phi}{\pi}}\right]\left((\nabla\cdot\mathbf{U}^d)\mathbf{I}\right) \quad (1.41)$$

Chapter 2

Description of sedFoam

SedFoam is a solver derived from twoPhaseEulerFoam of OpenFOAM version 2.1.0. SedFoam adds to twoPhaseEulerFoam specific models designed for sediment transport simulations. It also includes new subroutines to increase the model's stability. The solver can be obtained from the following website: http://github.com/sedfoam/sedfoam [1]. In this chapter we are going to describe the elements that compose a sedFoam case, and how the code has been implemented and structured.

2.1 SedFoam Installation

For OpenFOAM 5.0 (or version 6 or OpenFOAM Plus version 1806), download the official SedFoam-3.0 package:

```
git clone http://github.com/sedfoam/sedfoam/
```

access to the sedFoam folder,

```
cd sedfoam
```

compile it with,

```
./Allwmake
```

For OpenFOAM-6.0.x the following line in `solver/sedFoam.C` should be commented,

```
if (pimple.corrPISO()<pimple.nCorrPISO())
```

and the following line uncommented,

```
if (!pimple.finalPISOIter())
```

2.2 Swak4foam installation

Swak4foam is a library that offers the user the possibility to specify expressions involving the fields and evaluates them. It offers a number of utilities (for instance funkySetFields to set fields using expression), boundary conditions (groovyBC to specify arbitrary boundary conditions based on expressions) and function objects that allow doing many things that would otherwise require programming. In order to install it the following steps should be followed,

```
hg clone http://hg.code.sf.net/p/openfoam-extend/swak4Foam -u develop
cd swak4Foam
./maintainanceScripts/compileRequirements.sh
./Allwmake
```

2.3 Code structure

SedFoam is organized in three main directories. The first one contains the solver, the second one the new turbulence closures implemented for sediment transport, and the third one several tutorials. The solver directory contains the main files, and five other directories: `granularRheologyModels`, `griemannHolly`, `kinetic TheoryModels`, `interfacialModels`, and `phaseModel`. The first three contain the models for the interparticle stress (included the new one) and all the submodels inside them. The directory `interfacialModels` contains the models to calculate the interaction forces and lastly `phaseModel` contains the class that defines the two phases.

```
sedFoam
|
|--- solver
|    |--- granularRheologyModels
|    |     |--- FluidViscosityModel
|    |     |--- FrictionModel
|    |     |--- granularRheologyModel
|    |     |--- PPressureModel
|    |
|    |--- griemannHollyModel
|    |--- interfacialModels
|    |--- kineticTheoryModels
|    |     |--- conductivityModel
|    |     |--- frictionalStressModel
|    |     |--- granularPressureModel
|    |     |--- kineticTheoryModel
|    |     |--- radialModel
|    |     |--- viscosityModel
|    |
|    |--- phaseModel
|
|--- TurbulenceModels
|    |--- turbulenceModels
|
|--- Tutorials
```

2.4 Main code

A part of the main code is included below (`sedFoam.C`). During the runTime the equations mentioned in previous chapters are solved as follows,

1. Solve continuity equation (`alphaEqn.H`).
2. Calculate lift and drag coefficients (`liftDragCoeffs.H`).
3. Compute parameters from the Kinetic Theory. Solve granular temperature equation (`callKineticTheory.H`).
4. Compute contact pressure and frictional stress (`callFrictionStress.H`).
5. Create mommentum equations (`UEqns.H`).
6. Solves pressure Equation (`pEqns.H`).
7. At the end of each PIMPLE correction loop recalculates 1-4.

8. Solves turbulence model.

```
while (runTime.run())
{
    ..
    while (pimple.loop())
    {
        #include "alphaEqn.H"
        #include "liftDragCoeffs.H"
        #include "callKineticTheory.H"
        #include "callFrictionStress.H"
        #include "UEqns.H"

     while (pimple.correct())
     {
            #include "pEqn.H"

            if (!pimple.finalPISOIter())
            {
                if (correctAlpha)
                {
                    #include "alphaEqn.H"
                }
                #include "liftDragCoeffs.H"
                #include "callKineticTheory.H"
                #include "callFrictionStress.H"
            }

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

        }

        #include "DDtU.H"
    }
    ...
}
```

2.5 Class phaseModel

The main class implemented in sedFoam is the phaseModel class, located at sedFoam/phaseModel. An object of this class is defined for each phase we are modeling (in this case 2), and the members of the object include all information of the class such as the name of the phase, diameter of the phase, viscosity, density, velocity, concentration, between others. It does not contains any member function other than the constructor and functions to return the members of the class. The members are define as follows,

```
class phaseModel
{
    // Private data
```

```
        dictionary dict_;

        //- Name of phase
        word name_;

        //- Characteristic diameter of phase
        dimensionedScalar d_;

        //- shape factor for non-sperical particles
        dimensionedScalar sF_;

        //- exponent of the hindrance function for drag coeeficient
        dimensionedScalar hExp_;

        //- kinematic viscosity
        dimensionedScalar nu_;

        //- density
        dimensionedScalar rho_;

        //- Velocity
        volVectorField U_;

        //- Concentration
        volScalarField alpha_;

        //- Fluxes
        autoPtr<surfaceScalarField> phiPtr_;

        ...

}
```

2.6 Kinetic Theory class

The kinetic theory class manages all the aspects related to the kinetic theory model. The class members are composed of the different parameters that are needed to compute shear and normal stresses. The values that have to be specified by the user are defined in dictionary called `kineticTheoryProperties` located in the `constant` directory. The class members include five pointers that point to different submodels. These submodels are:

1. Viscosity Model
   (a) Gidaspow
   (b) Hrenya Sinclair
   (c) Syamlal
   (d) none
2. Conductivity Model
   (a) Gidaspow
   (b) Hrenya Sinclair
   (c) Syamlal

3. Radial Model
   (a) Carnahan Starling
   (b) Gidaspow
   (c) Lun Savage
   (d) Sinclair Jackson
   (e) Torquato
4. Granular Pressure Model
   (a) Lun
   (b) Syamlal Rogers O'Brien
   (c) Torquato
5. Frictional Stress Model
   (a) Johnson Jackson
   (b) Schaeffer
   (c) Srivastava Sundaresan

These models will define how some of the variables in the Kinetic Theory Model are defined. In the previous section the Kinetic Theory Model was explained using Syamlal for the Viscosity and Conductivity models, Lun for the Granular Pressure model, Srivastava Sundaresan for the frictional Stress model and Carnahan Starling for the radial model.

2.7 Inter-granular stress

The Kinetic Theory Model and the Granular Rheology Model are manipulated using objects called kineticTheory from the class kineticTheoryModel and granularRheology from the class granularRheologyModel. These objects are created in `createFields.H`,

```
kineticTheoryModel kineticTheory
(
    phasea,
    Ub,
    draga
);

granularRheologyModel granularRheology
(
    phasea,
    phaseb,
    pa
);
```

One big disadvantage of this method is that both files `kineticTheoryProperties` and also `granularRheologyProperties` have to be fully defined, independently of which model is going to be used. This comes from version of twoPhaseEulerFoam from which sedFoam was developed. Newer versions use pointers to select the interparticle stress model. The computation of the granular temperature and derived parameters is performed in `callKineticTheory.H` and `callFrictionStress.H` using conditional statements and members functions,

```
if (kineticTheory.on())
{
    // Compute Kinetic Theory including granular temperature solution
    kineticTheory.solve
    (
        galpha, gradUaT, turbulenceb->k(), turbulenceb->epsilon(), tur$
        B, runTime
    );
    ...
}

if (granularRheology.on())
{
    // Solving granular rheology
    granularRheology.solve(gradUaT,pff,alphaSmall,runTime.deltaT());
    ...
}
```

Chapter 3

Implementations

3.1 Greimann and Holly Model

In order to implement the Greimann and Holly Model to sedFoam, Equation 1.41 has to be adapted to the framework in which sedFoam was implemented. As we saw in the previous section, the interparticle stress is divided into frictional and contact component. These at the same time are divided in normal and shear stresses. These components are introduced individually in the equations, and therefore Equation 1.41 cannot be introduced all together into the model. Equation 1.41 can be rewritten as follows,

$$\boldsymbol{\xi}^{dc} = K_1\mathbf{I} + K_2\left(\nabla\mathbf{U}^d + \nabla\mathbf{U}^{d^T}\right) + K_3\left((\nabla\cdot\mathbf{U}^d)\mathbf{I}\right) \tag{3.1}$$

Where K_1, K_2 and K_3 can be found in Equation 1.41. From the Kinetic Theory of Granular Flows, and combining Equations 1.23, 1.25 and 1.28 we can rewrite the frictional component of the interparticle stress as,

$$\boldsymbol{\xi}^{df} = p^{df}\mathbf{I} + \mu^{df}\left(\nabla\mathbf{U}^d + \nabla\mathbf{U}^{d^T}\right) + \left(\lambda - \frac{1}{3}\mu^{df}\right)\nabla\cdot\mathbf{U}^d \tag{3.2}$$

Therefore by analogy of Equations 3.1 and 3.2 we can see that the model can be implemented by using,

$$p^{df} = K_1 \quad ; \quad \mu^{df} = K_2 \quad ; \quad \lambda = K_3 + \frac{1}{3}\mu^{df} \tag{3.3}$$

or

$$p^{df} = -2\alpha^d\rho^d g_o(1+e)\Phi \tag{3.4}$$

$$\mu^{df} = \frac{4}{5}\alpha^d\rho^d g_0(1+e)\nu_T^d + \frac{4}{5}\alpha^d\rho^d d_p g_0(1+e)\sqrt{\frac{\Phi}{\pi}} \tag{3.5}$$

$$\lambda = \frac{8}{15}\alpha^d\rho^d g_0(1+e)\nu_T^d + \frac{4}{5}\alpha^d\rho^d d_p g_0(1+e)\sqrt{\frac{\Phi}{\pi}} - \frac{1}{3}\mu^{df} \tag{3.6}$$

SedFoam does not take into account any turbulence closure for the disperse phase. Nevertheless, ν_T^d can be assumed equal to zero for the disperse phase [6], and the TKE of the disperse phase in equilibrium with the carrier phase. Using these assumptions the previous equations can be simplified as follows,

$$p^{df} = -2\alpha^d\rho^d g_o(1+e)\Phi \tag{3.7}$$

$$\mu^{df} = \frac{4}{5}\alpha^d \rho^d d_p g_0 (1+e)\sqrt{\frac{\tau}{\pi}} \tag{3.8}$$

$$\lambda = \frac{4}{5}\alpha^d \rho^d d_p g_0 (1+e)\sqrt{\frac{\tau}{\pi}} - \frac{1}{3}\mu^{df} = \frac{2}{3}\mu^{df} \tag{3.9}$$

3.2 Implementation

Following the same logic used in sedFoam, a new class called griemannHollyModel is created that contains all the information of the Greimann Holly Model. Assuming that we are located at the sedFoam main folder, we can proceed as follows: Create a copy of the solver directory called mySolver,

```
cp -r solver mySolver
```

edit `Allwmake` adding the following line at the end,

```
mySolver/Allwmake
```

rename `sedFoam.C`,

```
mv mySolver/sedFoam.C mySolver/mySedFoam.C
```

modify `mySolver/Make/files` changing it by,

```
mySedFoam.C
EXE = $(FOAM_USER_APPBIN)/mySedFoam
```

compile and test that no error arises by using,

```
./Allwmake
```

If everything went without errors then we can continue by creating a new class that will represent the new model. Because this model is a simplification of the Kinetic Theory Model, we can start with a copy of the kineticTheoryModel class and then modify it. Create the folder,

```
mkdir mySolver/griemannHollyModel
```

copy the class declaration and definition from the kinneticTheoryModels directory,

```
cp -r mySolver/kineticTheoryModels/Make/  mySolver/griemannHollyModel/
```

```
cp mySolver/kineticTheoryModels/kineticTheoryModel/kineticTheoryModel.*
mySolver/griemannHollyModel/
```

rename the files,

```
mv mySolver/griemannHollyModel/kineticTheoryModel.C
mySolver/griemannHollyModel/griemannHollyModel.C
```

```
mv mySolver/griemannHollyModel/kineticTheoryModel.H
mySolver/griemannHollyModel/griemannHollyModel.H
```

add to `mySolver/Allwmake` before `wmake` the following line,

```
wmake libso griemannHollyModel
```

add to `mySolver/Allwclean` before `wclean` the following line,

```
wclean libso griemannHollyModel
```

edit `mySolver/griemannHolly/Make/files` so it look as follows,

```
griemannHollyModel.C
LIB = $(FOAM_USER_LIBBIN)/libgriemannHollyModel
```

Finally the following lines in `mySolver/griemannHolly/Make/options` can be removed,

```
-I$(LIB_SRC)/foam/lnInclude \
-I../interfacialModels/lnInclude
```

Because several members of the class are the same, the class should be edited by removing and adding some members and functions. The final version of the class header is,

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2010 OpenCFD Ltd.
     \\/     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::griemannHollyModel

Description

SourceFiles
    griemannHollyModel.C

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

#ifndef griemannHollyModel_H
#define griemannHollyModel_H

#include "phaseModel.H"
#include "autoPtr.H"
#include "fvCFD.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
```

```
namespace Foam
{

/*---------------------------------------------------------------------------*\
                          Class griemannHollyModel Declaration
\*---------------------------------------------------------------------------*/

class griemannHollyModel
{
    // Private data

        const phaseModel& phasea_;
        const volVectorField& Ua_;
        const volScalarField& alpha_;
        const surfaceScalarField& phia_;
        const dimensionedScalar& rhoa_;
        const dimensionedScalar& da_;
        const dimensionedScalar& nua_;

        //- dictionary holding the modeling info
        IOdictionary griemannHollyProperties_;

        //- use kinetic theory or not.
        Switch griemannHolly_;

        //- Use nuta = nutb.
        Switch turbulentViscosityb_;

        //- coefficient of restitution
        const dimensionedScalar e_;

        //- angle of internal friction
        const dimensionedScalar phi_;

        //- The granular energy/temperature
        volScalarField Theta_;

        //- The granular viscosity
        volScalarField mua_;

        volScalarField muf_;

        //- The granular bulk viscosity
        volScalarField lambda_;

        //- The granular pressure
        volScalarField pa_;

        //- frictional stress
        volScalarField pf_;

        //- The radial distribution function
        volScalarField gs0_;
```

```
    // Private Member Functions

        //- Disallow default bitwise copy construct
        griemannHollyModel(const griemannHollyModel&);

        //- Disallow default bitwise assignment
        void operator=(const griemannHollyModel&);


public:

    // Constructors

        //- Construct from components
        griemannHollyModel
        (
            const phaseModel& phasea
        );


    //- Destructor
    virtual ~griemannHollyModel();


    // Member Functions

        void update(const volScalarField& kb, const volScalarField& nutb);

        bool on() const
        {
            return griemannHolly_;
        }

        const volScalarField& Theta() const
        {
            return Theta_;
        }

        const volScalarField& mua() const
        {
            return mua_;
        }

        const volScalarField& muf() const
        {
            return muf_;
        }

        const volScalarField& pa() const
        {
            return pa_;
        }
```

```
        const volScalarField& pf() const
        {
            return pf_;
        }

        const volScalarField& lambda() const
        {
            return lambda_;
        }

        const dimensionedScalar& phi() const
        {
            return phi_;
        }

};


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

} // End namespace Foam

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

#endif

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

The update function is used to update the values of the differently parameters each time step. It calculate the parameters different if $nu_T^d = 0$ or $nu_T^d = nu_T^c$. It is defined in `griemannHollyModel.C`. The final version of the file is:

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 1991-2010 OpenCFD Ltd.
     \\/     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 "griemannHollyModel.H"
#include "mathematicalConstants.H"
#include "fvCFD.H"

// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //

Foam::griemannHollyModel::griemannHollyModel
(
    const Foam::phaseModel& phasea
)
:
    phasea_(phasea),
    Ua_(phasea.U()),
    alpha_(phasea.alpha()),
    phia_(phasea.phi()),
    rhoa_(phasea.rho()),
    da_(phasea.d()),
    nua_(phasea.nu()),

    griemannHollyProperties_
    (
        IOobject
        (
            "griemannHollyProperties",
            Ua_.time().constant(),
            Ua_.mesh(),
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    ),
    griemannHolly_(griemannHollyProperties_.lookup("griemannHolly")),
    turbulentViscosityb_(griemannHollyProperties_.lookup("turbulentViscosityb")),
    e_(griemannHollyProperties_.lookup("e")),
    phi_(dimensionedScalar(griemannHollyProperties_.lookup("phi"))*M_PI/180.0),
    Theta_
    (
        IOobject
        (
            "Theta",
            Ua_.time().timeName(),
            Ua_.mesh(),
            IOobject::MUST_READ,
            IOobject::AUTO_WRITE
        ),
        Ua_.mesh()
    ),
    mua_
    (
        IOobject
        (
```

```
            "mua",
            Ua_.time().timeName(),
            Ua_.mesh(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        Ua_.mesh(),
        dimensionedScalar("zero", dimensionSet(1, -1, -1, 0, 0), 0.0)
    ),
    muf_
    (
        IOobject
        (
            "muf",
            Ua_.time().timeName(),
            Ua_.mesh(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        Ua_.mesh(),
        dimensionedScalar("zero", dimensionSet(1, -1, -1, 0, 0), 0.0)
    ),
    lambda_
    (
        IOobject
        (
            "lambda",
            Ua_.time().timeName(),
            Ua_.mesh(),
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        Ua_.mesh(),
        dimensionedScalar("zero", dimensionSet(1, -1, -1, 0, 0), 0.0)
    ),
    pa_
    (
        IOobject
        (
            "pa",
            Ua_.time().timeName(),
            Ua_.mesh(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        Ua_.mesh(),
        dimensionedScalar("zero", dimensionSet(1, -1, -2, 0, 0), 0.0)
    ),
    pf_
    (
        IOobject
        (
            "pf_",
            Ua_.time().timeName(),
```

```
            Ua_.mesh(),
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        Ua_.mesh(),
        dimensionedScalar("zero", dimensionSet(1, -1, -2, 0, 0), 0.0)
    ),
    gs0_
    (
        IOobject
        (
            "gs0",
            Ua_.time().timeName(),
            Ua_.mesh(),
            IOobject::NO_READ,
            IOobject::NO_WRITE
        ),
        Ua_.mesh(),
        dimensionedScalar("zero", dimensionSet(0, 0, 0, 0, 0), 1.0)
    )
{}


// * * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * * * //

Foam::griemannHollyModel::~griemannHollyModel()
{}


// * * * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //

// Update parameters

void Foam::griemannHollyModel::update(const volScalarField& kb, const volScalarField& nutb)
{
    Info << "Updating Griemann and Holly parameters" << endl;

    gs0_ = (2 - alpha_)/(2*pow((1-alpha_),3));

    Theta_ = 2.0/3.0 * kb;

    if(turbulentViscosityb_)
    {
        mua_=nutb*4.0/5.0*alpha_*rhoa_*gs0_*(1+e_)
        +4.0/5.0*alpha_*rhoa_*da_* gs0_*(1+e_)*sqrt(Theta_/M_PI);
        lambda_=- 8.0/15.0*alpha_*rhoa_*gs0_*(1+e_)*nutb
        +4.0/5.0*alpha_*rhoa_*da_*gs0_*(1+e_)*sqrt(Theta_/M_PI)-1.0/3.0*mua_;
    }

    else
    {
        mua_=4.0/5.0*alpha_*rhoa_*da_*gs0_*(1+e_) *sqrt(Theta_/M_PI);
        lambda_=2.0/3.0*mua_;
    }
```

```
    pa_=-2.0*alpha_*rhoa_*gs0_*(1+e_)*Theta_;

}



//}

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

In order to include the new model in the code, a new object is created in `createFields.H`. The following lines should be added after the creations of the objects of kineticTheoryModel and granularRheologyModel,

```
griemannHollyModel griemannHolly
(
    phasea
);
```

also the following header should be added to `mySolver/mySedFoam.C`,

```
#include "griemannHollyModel.H"
```

Then a conditional statement is included in `callFrictionStress.H`:

```
else  if(griemannHolly.on())
{
    volTensorField dU = fvc::grad(Ua);
    volSymmTensorField dUU = symm(dU);
    volSymmTensorField devS = dUU - (scalar(1.0)/scalar(3.0))*tr(dUU)*I;
    dimensionedScalar I2Dsmall
    (
        "small",
        dimensionSet(0 , 0 ,-2 ,0 , 0, 0, 0),
        scalar(1.0e-40)
    );

    griemannHolly.update(turbulenceb->k(),turbulenceb->nut());
    nuEffa = griemannHolly.mua()/((alpha + alphaSmall)*rhoa);
    lambdaUa = griemannHolly.lambda();
    pa = griemannHolly.pa();
    volScalarField muEff_f =
    (
        pff*Foam::sin(kineticTheory.phi())
       /sqrt(scalar(2.0)*(devS && devS) + I2Dsmall)
    );
    nuFra = muEff_f/rhoa;
    nuEffb = turbulenceb->nut() + nub;
}
```

In order to compile the new model, the `mySolver/Make/options` file has to be modified. The following lines should be added,

```
-IgriemannHollyModel/lnInclude \
-lgriemannHollyModel\
```

Finally, the model can be compiled with `./Allwmake`.

Chapter 4

Tutorial sedFoam

This chapter will describe an openChannel simulation with two different interparticle stress models: Granular Rheology model and Griemann and Holly model. The Griemann and Holly model will be simulated with $\nu_T^d = 0$ and with $\nu_T^d = \nu_T^c$. The simulation will consist of a turbulent flow over a flat erodible bed. The averaged flow velocity is about U =0.52 m/s. The height of the flow is 0.17 m.

4.1 Case structure

A general case is structured in the usual way: a `0/`, `constant/` and `system/` folders. The contentes of `0/` and `constant/` folders are described in tables 4.1 and 4.2. The contents of the `system\` folder do not differ from other solvers.

File	Content
`alpha_a`	Sediment phase fraction
`alpha_b`	Water phase fraction
`epsilon`	Dissipation of turbulence energy
`k`	Turbulent kinetic energy
`mua`	Collisional dynamic viscosity (μ^{dc})
`muf`	Frictional dynamic viscosity (μ^{df})
`muI`	Friction coefficient (Granular Rheology model)
`nuEffa`	Effective kinematic viscosity of the sediment phase (ν^d_{eff})
`nuEffb`	Effective kinematic viscosity of the water phase (ν^c_{eff})
`nuFra`	Frictional kinematic viscosity of the sediment phase (ν^c_{eff})
`nut`	Turbulent kinematic viscosity of the water phase(ν^c_t)
`nuvb`	Fluid effective viscosity (Granular Rheology model)
`p`	Pressure
`pa`	Normal collisional stress (p^{dc})
`pff`	Normal frictional stress (p^{df})
`phia`	Sediment phase flux
`phib`	Water phase flux
`p_rbgh`	Pressure minus $\rho^b gh$
`SUStilde`	Turbulent suspension force
`Theta`	Granular temperature (Φ)
`U`	Total velocity ($\mathbf{U}$)
`Ua`	Sediment phase velocity ($\mathbf{U}^d$)
`Ub`	Sediment phase velocity ($\mathbf{U}^c$)

Table 4.1: Contents of 0/ folder.

File	Content
`forceProperties`	Define virtual mass, lift and eddy diffusivity coefficients
`g`	Definition of gravity
`granularRheologyProperties`	Includes a switch on/off the Granular Rheology model and the definition of all the parameters needed for the model
`griemannHollyProperties`	Includes a switch on/off the Greimann and Holly model and the definition of all the parameters needed for the model
`interfacialProperties`	Define drag models
`kineticTheoryProperties`	Includes a switch on/off the Kinetic Theory of Granular Flows model and the definition of all the parameters needed for the model
`ppProperties`	Define parameters for normal frictional stress
`transportProperties`	Define properties of sediment and water phases
`turbulenceProperties`	Selection of turbulence model and related parameters.
`twophaseRASProperties`	Definition of additional turbulence parameters specific for two-phase flows

Table 4.2: Contents of constant/ folder.

4.2 Mesh generation and boundary conditions

The numerical domain consists of unidimensional domain with 1x1x30 cells in the x, y and z directions respectively (Figure 4.1). The inlet and outlet boundaries are defined with cyclic boundary conditions. The top and bottom of the channel are wall type boundaries while the back and front (y-direction) are empty type boundaries.

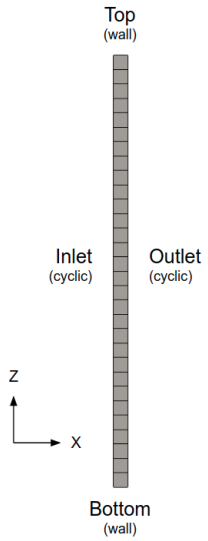

Figure 4.1: Geometry of the sedFoam tutorial case.

The mesh is generated with blockMesh. The domain size in the x and y direction is defined in such a way to make the shape of the cells cubic. The boundary conditions are defined in Table 4.3.

Variable	Bottom	Top	Inlet/Outlet
alpha_a	zeroGradient	fixedValue (0)	cyclic
k	zeroGradient	kqRWallFunction	cyclic
epsilon	zeroGradient	epsilonWallFunction	cyclic
nut	zeroGradient	nutWallFunction	cyclic
pa	zeroGradient	slip	cyclic
Theta	zeroGradient	zeroGradient	cyclic
Ua	zeroGradient	zeroGradient	cyclic
Ub	zeroGradient	zeroGradient	cyclic
p_rbgh	fixedFluxPressure	fixedValue (0)	cyclic
muI	zeroGradient	zeroGradient	cyclic

Table 4.3: Boundary conditions used in the tutorial

4.3 Case configuration

The phases properties are specified in `constant/transportProperties`. They are summarized in Table 4.4.

Variable	Sediment	Water
ρ	1190 kg/m^3	1000 kg/m^3
ν	N.A.	1e-6 m^2/s
diameter	1e-3 m	N.A
shape factor	0.5	N.A.
hidrance exponent	2.65	N.A.
ν_{max}	0.1 m^2/s	0.1 m^2/s
α^d_{max}	1e-6	N.A.

Table 4.4: Phases properties used in the tutorial

The properties of the Greimann Holly model are specified in `constant/griemannHollyProperties`. They are specified as follows,

```
griemannHolly   on;
turbulentViscosityb off;
e               e [ 0 0 0 0 0 0 0 ] 0.8;
phi             phi [ 0 0 0 0 0 0 0 ] 28.0;
```

for the $\nu^d_T = 0$ case, and

```
griemannHolly   on;
turbulentViscosityb on;
e               e [ 0 0 0 0 0 0 0 ] 0.8;
phi             phi [ 0 0 0 0 0 0 0 ] 28.0;
```

for the $\nu^d_T = \nu^c_T$ case. The properties of the Granular Rheology model are specified in the file `constant/granularRheologyProperties`. They are specified as follows,

```
granularRheology   on;
alphaMaxG       alphaMaxG [ 0 0 0 0 0 0 0 ] 0.55;
mus             mus  [ 0 0 0 0 0 0 0 ] 0.52;
mu2             mu2  [ 0 0 0 0 0 0 0 ] 0.96;
I0              I0  [ 0 0 0 0 0 0 0 ] 0.6;
Bphi            Bphi [ 0 0 0 0 0 0 0 ] 0.66;
n               n [ 0 0 0 0 0 0 0 ] 2.5;
Dsmall          Dsmall [ 0 0 -1 0 0 0 0 ] 1e-6;
```

```
relaxPa         relaxPa [0 0 0 0 0 0 0] 0.005;
FrictionModel  MuI;
PPressureModel MuI;
FluidViscosityModel  BoyerEtAl;
```

4.4 Running the tutorial

The tutorial is provided with four scripts: `Allrun`, `postProcess`, `plot` and `Allclean`. The first script (`Allrun`) creates the mesh, set the initial conditions and run the three cases. The second one (`postProcess`) postProcess the results, extracting sediment concentration and velocity for both phases. To do this it uses the function object singleGraph. The third script (`plot`) plots the results using gnuplot, and the last script (`Allclean`) cleans the three cases.
The `Allrun` script executes the following commands,

```
#!/bin/sh
# Create the mesh
blockMesh -case 1DSheetFlow_Original
blockMesh -case 1DSheetFlow_GriemannHolly
blockMesh -case 1DSheetFlow_GriemannHollyNut
# create the intial time folder
cp -r 1DSheetFlow_Original/0_org 1DSheetFlow_Original/0
cp -r 1DSheetFlow_GriemannHolly/0_org 1DSheetFlow_GriemannHolly/0
cp -r 1DSheetFlow_GriemannHolly2/0_org 1DSheetFlow_GriemannHollyNut/0
# Initialize the alpha field
funkySetFields -time 0 -case 1DSheetFlow_Original
funkySetFields -time 0 -case 1DSheetFlow_GriemannHolly
funkySetFields -time 0 -case 1DSheetFlow_GriemannHollyNut
# Run sedFoam
mySedFoam -case 1DSheetFlow_Original > log.Original&
mySedFoam -case 1DSheetFlow_GriemannHolly > log.GriemannHolly&
mySedFoam -case 1DSheetFlow_GriemannHollyNut > log.GriemannHollyNut&
```

The `postProcess` script executes the following commands,

```
#!/bin/sh
# Create the mesh
postProcess -func singleGraph -case 1DSheetFlow_Original
postProcess -func singleGraph -case 1DSheetFlow_GriemannHolly
postProcess -func singleGraph -case 1DSheetFlow_GriemannHollyNut
```

The `plot` script executes the following commands,

```
#!/usr/bin/gnuplot -persist
set term x11 0
set logscale x
set xrange [1e-7:1]
set format x "%1.0E"
set yrange [0:0.14]
set xlabel "Sediment fraction (alpha_a)"
set ylabel "y [m]"
plot "1DSheetFlow_Original/postProcessing/singleGraph/50/line_alpha_a.xy" u 2:1 title "Granular Rheolo

set term x11 1
unset logscale x
set xrange [0:1]
```

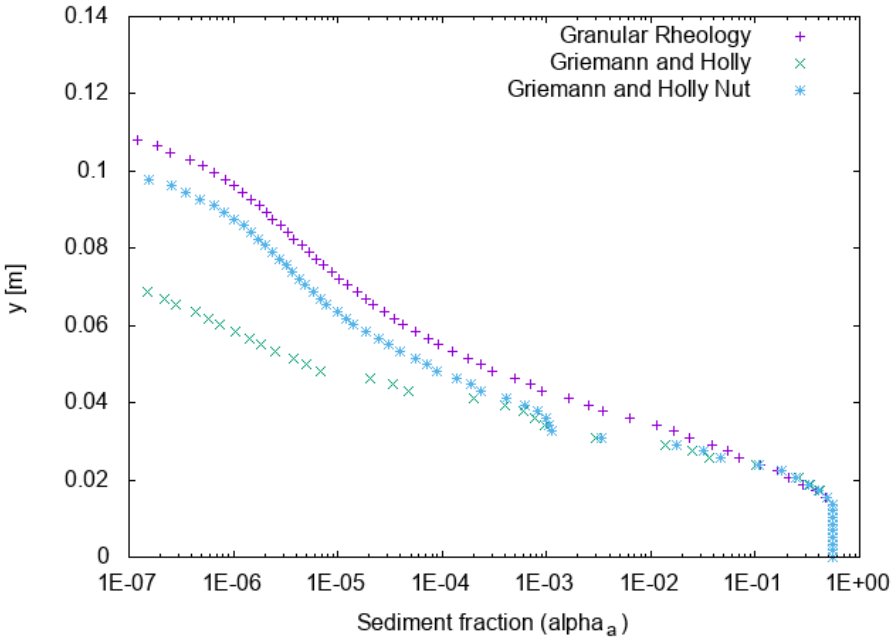

Figure 4.2: Sediment concentration for the Granular Rheology model and for the Greimann and Holly model under two configurations.

```
set yrange [0:0.14]
set xlabel "Water velocity (Ua)"
set ylabel "y [m]"
plot "1DSheetFlow_Original/postProcessing/singleGraph/50/line_Ua_Ub.xy" u 2:1 title "Granular Rheology

set term x11 2
set xrange [0:1]
set yrange [0:0.14]
set xlabel "Sediment velocity (Ub)"
set ylabel "y [m]"
plot "1DSheetFlow_Original/postProcessing/singleGraph/50/line_Ua_Ub.xy" u 5:1 title "Granular Rheology
```

The `Allclean` script executes the following commands,

```
#!/bin/sh

# Remove the mesh
foamCleanPolyMesh -case 1DSheetFlow_Original
foamCleanPolyMesh -case 1DSheetFlow_GriemannHolly
foamCleanPolyMesh -case 1DSheetFlow_GriemannHollyNut

# Remove time folders
foamListTimes -rm -time 0: -withZero -case 1DSheetFlow_Original
foamListTimes -rm -time 0: -withZero -case 1DSheetFlow_GriemannHolly
foamListTimes -rm -time 0: -withZero -case 1DSheetFlow_GriemannHollyNut

# Remove postProcessing folder

rm -r 1DSheetFlow_Original/postProcessing
```

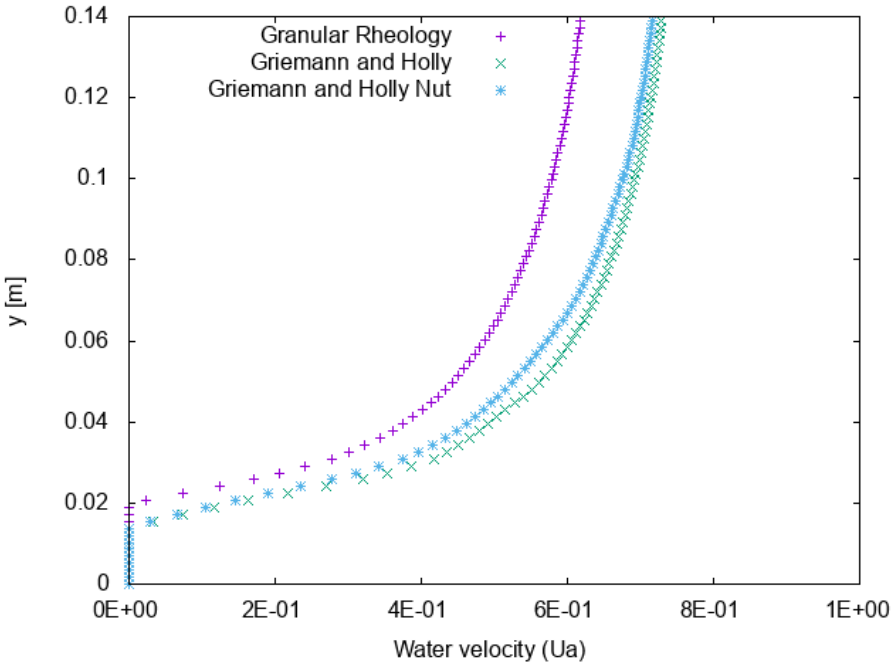

Figure 4.3: Water velocity for the Granular Rheology model and for the Greimann and Holly model under two configurations.

```
rm -r 1DSheetFlow_GriemannHolly/postProcessing
rm -r 1DSheetFlow_GriemannHollyNut/postProcessing

# Remove logs
rm -rf  gradPOSC.txt 1DSheetFlow_Original/log.*
rm -rf  gradPOSC.txt 1DSheetFlow_GriemannHolly/log.*
rm -rf  gradPOSC.txt 1DSheetFlow_GriemannHollyNut/log.*
```

The results of the simulations are presented in Figures 4.2, 4.3 and 4.4

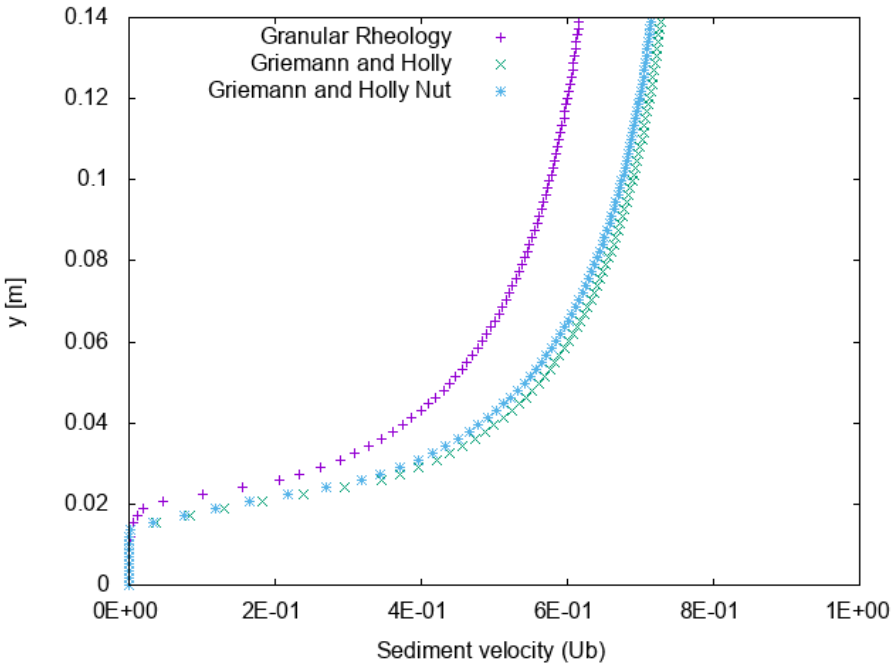

Figure 4.4: Sediment velocity for the Granular Rheology model and for the Griemann and Holly model under two configurations.

Study questions

1. How do you obtain the phase-intensive equation for the sediment phase?
2. What is the purpose of `callFrictionStress.H` file?
3. How do you select the submodels for the Kinetic Theory Model?
4. What would be the main steps to implement a new interparticle stress model?

Bibliography

[1] Chauchat, J., Cheng, Z., Nagel, T., and Cyrille Bonamy, a. T.-J. H. Sedfoam-2.0: a 3d two-phase flow numerical model for sediment transport. *Geoscientific Model Development Discuss. 10* (2017).

[2] Crowe, C. T. *Theory of Multicomponent Fluids*. CRC Press, 2012.

[3] Drew, D. A., and Passman, S. L. *Multiphase flows with droplets and particles*, vol. 135. Springer-Verlag New York, 1999.

[4] Garcia, M. H. Lecture notes: Sediment transport. Tech. rep., University of Illinois at Urbana-Champaign, Ven Te Chow Hydrosystems Lab., June 2006.

[5] Greimann, B. P., and Jr., F. M. H. Two-phase flow analysis of concentration profiles. *Journal of Hydraulic Engineering 127* (2001).

[6] Jha, S. K., and Bombardelli, F. A. Toward two-phase flow modeling of nondilute sediment transport in open channels. *Journal of Geophysical Research 115* (Aug 2010).

[7] Weller, H. Derivation, modelling and solution of the conditionally averaged two-phase flow equations. Tech. rep., OpenCFD, 02 2005.