USER DOCUMENTATION :

TRUST-TrioCFD : Multiphase Module v1.9.4

Version	Date	Code manager	Authors
v1.9.4	24 juin 2024	A DU CLUZEAU	A BURLOT C BAZIN A PEITAVY C REISS A GERSCHENFELD F PECQUERY
DES/ISAS/DM2S **CEA SACLAY** **91191 GIF-SUR-YVETTE**		*Input file :* TrioCFD_Pb_multiphase.tex *Software :* TrioCFD	
		DES/ISAS/DM2S/STMF/LMSF/UD	

Contents

Definition of dimensional and dimensionless variables

Variable name	Expression
Gas phase	g
Liquid phase	l
Distance to the wall	y
Normal unit vector from the wall	$\overrightarrow{n}$
Velocity of phase k	$\overrightarrow{u_k}$
Dynamic viscosity of phase k	μ_k
Volume mass of phase k	ρ_k
Kinematic viscosity of phase k	$\nu_k = \mu_k/\rho_k$
Turbulent dynamic viscosity of the carrying phase	μ_t
Turbulent kinematic viscosity of the carrying phase	$\nu_t = \mu_t/\rho$
Reynold's stress tensor of the carrying phase	$\overline{\overline{\tau_R}} = - < u_i u_j >$
Turbulent kinetic energy of the carrying phase	k
Turbulent dissipation of the carrying phase	ε
Temperature of phase k	T_k
Temperature of the wall	T_{wall}
Thermal conductivity of phase k	λ_k
Heat capacity at constant volume of phase k	Cp_k
Forced convection heat transfer coefficient	h_{fc}
Liquid fraction	α_l
Void fraction	α_g
Internal energy of phase k	e_k
Mass transfer to phase k	Γ_k
Mass transfer to phase k	Γ_k
Interfacial forces applied to phase k	$\overrightarrow{F}_{ki}$
Volume forces applied to phase k (i.e. gravity)	$\overrightarrow{F}_k$
Interfacial heat flux to phase k	q_{ki}
Wall heat flux to phase k	q_{kp}
Interfacial area between phase l and v	a_i
Bubble diameter of phase v	$d_b = \frac{6\alpha_g}{a_i}$
Bubble nucleation and departure from wall diameter	d_{dep}
Vapor/liquid saturation temperature	T_{sat}
Vapor/liquid latent heat	L_{vap}
Vapor/liquid surface tension	σ
Percentage of wall surface occupied by sliding bubbles	S_{sl}
Bubble Reynold's number	$Re_b = \frac{\|\|\overrightarrow{u_v} - \overrightarrow{u_l}\|\| d_b}{\nu_l}$
Bubble turbulent fluctuation Weber number	$We = \frac{(\varepsilon d_b)^{2/3} \rho_l d_b}{\sigma}$
Eotvos number	$Eo = \frac{g(\rho_l - \rho_g) d^2}{\sigma}$
Turbulent Prandtl number of the liquid	$Pr_t \sim 1$
Superheat Jacob number	$Ja_{\text{sup}} = \frac{\rho_l Cp_l \cdot (T_{\text{wall}} - T_{\text{sat}})}{\rho_v L_{\text{vap}}}$
Subcooling Jacob number	$Ja_{\text{sub}} = \frac{\rho_l Cp_l \cdot (T_{\text{sat}} - T_l)}{\rho_v L_{\text{vap}}}$

Introduction

This document serves as a theory guide for the multiphase framework of TRUST/TrioCFD. In brief, using the HPC platform TRUST, TrioCFD allows to perform two-phase flow RANS computation in the modelling paradigm where phases are considered continuous. It allows the computation of the flow of an arbitrary number of phases. Continuous equations for mass, momentum and energy are solved for each phase. For the classical case of two phases, one continuous and the other discrete, this system gives the classical six-equations-single-pressure Euler-Euler model. Using a special operator which allows to handle vanishing phases, this system can easily be simplified to handle any equilibrium. As a result, the same framework allows to simulate, for instance, a homogeneous equilibrium model. Currently, three approaches are available and documented: the six-equation-single-pressure model, the drift-velocity model and the HEM model. It consists on a set of source terms to the previously mentioned conservation equations. Single-phase RANS modelling is obviously available. Turbulence can currently be taken into account for the continuous phase only. The document describes the available time and spatial schemes. Details are given on the boundary conditions. Being a shared computing platform for several applications, a list of validation test cases is provided for each module based on the multiphase framework.

Technicalities are given on the links between mathematical formulation of algorithms and their implementation. To assist development, some details are provided about the coding standard. The documentation tries to follow the implementation in the code to described what is actually performed in the code. The origin of physical models is given in bibliography when available, or details about the intention of the developer are provided for in-house models.

In its current version, details are given about the shape of the input file. Some aspects about the `C++ architecture` are also provided to guide the user and developer in the file location between TRUST and TrioCFD.

Any remark or recommendation of improvment is welcome and can be emailed at trust@cea.fr.

Chapter 1

Governing equations

This chapter gives a general introduction to the theoretical framework of the multiphase system of averaged equations (Section 1.1), the two-fluid and drift-flux approches (Section 1.2) and a few details on turbulence equation (Section 1.3). The curious reader can find extensive details on the multiphase Navier-Stokes framework in the books of **LivreIH**, **Morel2015**, **LivreDrewPassman**.

1.1 Multiphase system averaged equations

1.1.1 General balance equations

The balance equations express the conservation of a quantity ϕ_k of a phase k with a velocity $\mathbf{u}_k$ in a reference frame (X,t) by integrating its changes in a control volume Ω. Assuming then the continuity of the quantity ϕ_k related to any control volume, we can derive the integral balance equations into differential balance equations. Then we must introduce volume variables as the fluid density ρ_k, the flux $\underline{\underline{J}}$ and the body source g_k. The general differential balance equation of ϕ_k is given by:

$$\underbrace{\frac{\partial}{\partial t}\rho_k\phi_k}_{\text{Time changes}} + \underbrace{\nabla\cdot(\rho_k\phi_k u_k)}_{\text{Convection}} = \underbrace{-\nabla\cdot\underline{\underline{J}}}_{\text{Diffusion}} + \underbrace{\rho_k g_k}_{\text{Source}} \tag{1.1}$$

Considering a physical system the quantities ϕ_k are the mass, momentum and energy. The table 1.1 summarizes the particular cases of the general differential balance equation with Γ_k the source of mass flow rate, P_k the pressure, $\underline{\underline{\tau}}_k$ the viscous stress tensor, $\mathbf{g}$ the body forces, e_k the internal energy, q_k the heat flux and $\dot{q}_k$ the body heat. The subscript k refers to each phase.

Quantity	ϕ_k	$\underline{\underline{J}}$	g_k
Mass	1	0	Γ_k
Momentum	$\mathbf{u}_k$	$-\underline{\underline{T}}_k = P_k\underline{\underline{I}} - \underline{\underline{\tau}}_k$	$\mathbf{g}$
Energy	$e_k + \frac{u_k^2}{2}$	$q_k - \underline{\underline{T}}_k\cdot\mathbf{u}$	$\mathbf{g}\cdot\mathbf{u}_k + \frac{\dot{q}_k}{\rho_k}$

Table 1.1: Conserved quantities in balance equations

1.1.2 Multiphase balance equations

The simulation of an averaged multiphase system requires the computation of the averaged local Navier-Stokes equations for each phase k. The averaged Navier-Stokes equations predict the average behavior of fluid motion and properties by filtering out local instant fluctuations.

However, precise modeling of all microscopic scales is still necessary for an accurate reproduction of the averaged flow. To handle multiple phases within the averaged framework, it becomes essential to introduce a variable that defines the phase mixture.

Let χ_k^{phase} be a tracer of phase k which defines whether or not we are in this phase. It takes the value 1 if we are in the phase, 0 otherwise. The fraction of phase k, denoted as α_k, in a controlled volume $V_{\Omega k}$ within a reference frame $(X,\ t)$, is defined as follows by

$$\alpha_k = \frac{1}{V_\Omega}\int_{V_\Omega} \chi_k^{\text{phase}}(X,\ t)dV_\Omega = \frac{V_{\Omega k}}{V_\Omega}. \tag{1.2}$$

From this equation arises the well-known axiom of continuity over all phases:

$$\sum_k \alpha_k = \sum_k \frac{V_{\Omega k}}{V_\Omega} = \frac{V_\Omega}{V_\Omega} = 1. \tag{1.3}$$

The averaged method introduces two different mean quantities for a property ϕ: the mean of a field k in the controlled volume $\bar{\phi}_k$ and the mean of a field in its proper volume in the controlled volume $\overline{\overline{\phi}}_k$. With the fraction α_k of the field k in a controlled volume $V_{\Omega k}$ in a reference frame $(X,\ t)$, they are defined by:

$$\bar{\phi}_k = \frac{1}{V_\Omega}\int_{V_\Omega} \phi_k(X,\ t)dV_\Omega, \tag{1.4}$$

$$\overline{\overline{\phi}}_k = \frac{1}{V_{\Omega k}}\int_{V_{\Omega k}} \phi_k(X,\ t)dV_\Omega = \frac{\bar{\phi}_k}{\alpha_k}. \tag{1.5}$$

$$\tag{1.6}$$

It means that we can replace the mean $\bar{\phi}_k$ from the general balance equation by the quantity $\alpha_k\overline{\overline{\phi}}_k$, with $\overline{\overline{\phi}}_k$ being the quantity seen as single phase. To illustrate this, Figure 1.1 gives an example of the different means and the link with the void fraction. In this perspective, we can

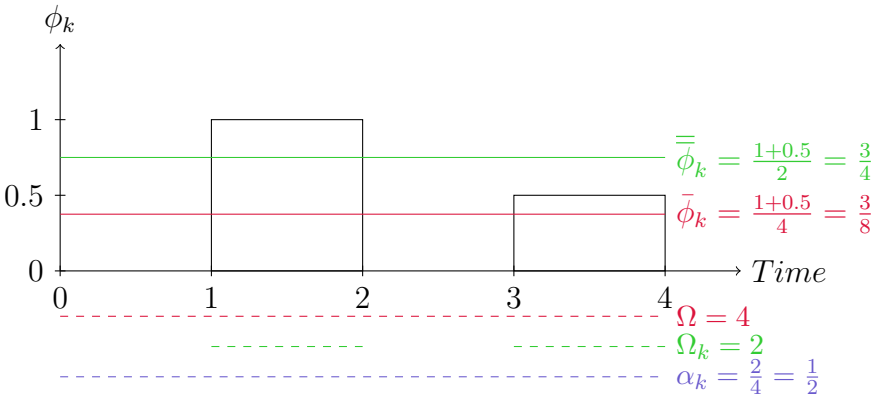


Figure 1.1: Example of means computation from a quantity ϕ_k.

interpret each quantity as follows:

- $\bar{\phi}_k$ is the true observed conserved mean quantity,
- $\overline{\overline{\phi}}_k$ is the easily modeled mean value as if it is a single phase,
- α_k is the respective presence given by the equilibrium between phases.

Furthermore, in order to ensure accuracy, the mean values must satisfy a fundamental assumption of regularity, which ensures that the average of a mean value remains consistent with the mean value itself. In order to account for the zero mean value of the fluctuation ϕ', another commonly used approach is the Favre averaging. The Favre average of the quantity ϕ for the field k is defined as $\tilde{\phi}_k = \frac{\overline{\rho\phi_k}}{\bar{\rho}}$, where the overline represents the averaging operation. For the sake of clarity, the variables in the following equations are assumed averaged even if bar notations are not used.
Then conservation equations for mass in each phase k is given by:

$$\underbrace{\frac{\partial}{\partial t}\rho_k\alpha_k}_{\text{Time changes}} + \underbrace{\nabla\cdot(\rho_k\alpha_k\mathbf{u}_k)}_{\text{Convection}} = \underbrace{\Gamma_k}_{\text{Source}}, \tag{1.7}$$

with t the time, ρ_k the density of phase k, $\mathbf{u}_k$ the velocity of phase k and Γ_k a mass source term of phase k.

The conservation equation for momentum in each phase is:

$$\underbrace{\frac{\partial}{\partial t}\rho_k\alpha_k\mathbf{u}_k}_{\text{Time changes}} + \underbrace{\nabla\cdot(\rho_k\alpha_k\mathbf{u}_k\otimes\mathbf{u}_k)}_{\text{Convection}} = \underbrace{-\alpha_k\nabla P_k}_{\text{Pressure}} + \underbrace{\nabla\cdot\left(\alpha_k\underline{\underline{\tau}}_k\right)}_{\text{Diffusion}} - \underbrace{\nabla\cdot\left(\alpha_k\underline{\underline{\tau}}_k^t\right)}_{\text{Turbulence}} + \underbrace{\rho_k\alpha_k\mathbf{g}}_{\text{Gravity}} + \underbrace{\mathbf{I_k}}_{\text{Sources}}, \tag{1.8}$$

with P_k the pressure, $\underline{\underline{\tau}}_k$ and $\underline{\underline{\tau}}_k^t$ respectively the viscous and turbulent stress tensors, $\mathbf{g}$ gravity and $\mathbf{I_k}$ an interfacial transfer.

The conservation equation for the internal energy in each phase is:

$$\underbrace{\frac{\partial}{\partial t}\rho_k\alpha_k e_k}_{\text{Time changes}} + \underbrace{\nabla\cdot(\rho_k\alpha_k e_k\mathbf{u}_k)}_{\text{Convection}} = \underbrace{-P_k\frac{D\alpha_k}{Dt}}_{\text{Pressure}} + \underbrace{\nabla\cdot\psi_k}_{\text{Diffusion}} - \underbrace{\nabla\cdot\psi_k^t}_{\text{Turbulence}} + \underbrace{q_k}_{\text{Sources}}, \tag{1.9}$$

with ψ_k and ψ_k^t respectively the heat and turbulent fluxes, and $\mathbf{q_k}$ a source of heat.

1.1.3 Interfacial momentum transfers with dispersed hypothesis

According to **LivreIH**, the interfacial transfer between two phases in the momentum balance equation can be expressed as follows:

$$\begin{aligned}\mathbf{I}_k &= -\frac{1}{\Delta t}\sum_j \frac{1}{u_{ni}}\left(\rho_k\mathbf{n}_k\left(\mathbf{u}_k-\mathbf{u}_i\right)\mathbf{u}_k - \underline{\underline{T}}_k\mathbf{n}_k\right)\\ &= -\sum_j \frac{1}{\Delta t}\frac{1}{u_{nij}}\left(\rho_k\mathbf{n}_k\left(\mathbf{u}_k-\mathbf{u}_i\right)\mathbf{u}_k - \left(\underline{\underline{T}}_k - \underline{\underline{T}}_k^{\text{interface}}\right)\mathbf{n}_k\right) + \sum_j \underline{\underline{T}}_k^{\text{interface}}\frac{1}{\Delta t}\frac{1}{u_{nij}}\mathbf{n}_k,\end{aligned} \tag{1.10}$$

with u_{ni} the interfacial normal velocity, $\mathbf{n}_k$ the interfacial normal vector, $\mathbf{u}_i$ the interfacial velocity and $\underline{\underline{T}}^{\text{interface}}$ interfacial stress tensor.

A new variable, called interfacial area concentration a_i, can be introduced to simplify the formulation:

$$a_{ij} = \frac{1}{\Delta t u_{ni\,j}}, \tag{1.11}$$

with Δt a time interval.

We can also notice that:

$$\frac{1}{\Delta t}\sum_j \frac{1}{u_{nij}}\mathbf{n}_k = -\nabla\alpha \tag{1.12}$$

By decomposing the stress into pressure and shear-stress components, subscripted i at the interface, and incorporating the mass loss rate $\dot{m}$, we obtain the following expression:

$$\begin{aligned}\mathbf{I}_k = & \underbrace{\Gamma_k \tilde{\mathbf{u}}_{ki}}_{\text{Mass transfer}} + \underbrace{\sum_j a_{ij}\left(\overline{\overline{P_{ki}}} - P_k\right)\mathbf{n}_k}_{\text{Interfacial pressure imbalance}} + \underbrace{\sum_j a_{ij}\left(\underline{\underline{\tau}}_k - \overline{\overline{\underline{\underline{\tau}}_{ki}}}\right)\mathbf{n_k}}_{\text{Interfacial shear-stress imbalance}} \\ & + \underbrace{\overline{\overline{P_{ki}}}\nabla\alpha_k}_{\substack{\text{Interfacial pressure} \\ \text{dispersion}}} - \underbrace{\overline{\overline{\underline{\underline{\tau}}_{ki}}}\nabla\alpha_k}_{\substack{\text{Interfacial shear-stress} \\ \text{dispersion}}},\end{aligned} \tag{1.13}$$

with quantities denoted with an i as quantities at the interface and Γ_k the mass transfer term.

When one fluid can be supposed to be dispersed, the expression for the averaged interfacial momentum transfers depends on the hydrodynamical models, denoted as M^k:

$$\mathbf{I}_k = \underbrace{\sum_j a_{ij}\mathbf{M}^k}_{\text{Hydrodynamical forces}} + \underbrace{\mathbf{M}^k\nabla\alpha_k}_{\text{Interfacial dispersion force}}. \tag{1.14}$$

When a phase can be assumed to be dispersed, we can represent it as a population of bubbles/-droplet with varying diameters, which serves as a topological characteristic for the continuous fluid. The dispersed fluid is characterized by two interconnected attributes: a distribution of bubble diameters and the concentration of interfaces.

We can define the Sauter-mean diameter of the distribution f_d of sizes D as:

$$D_{sm} = \frac{\int f_d D^3 \mathrm{d}D}{\int f_d D^2 \mathrm{d}D}, \tag{1.15}$$

One can write a relation between the Sauter-mean diameter and the area concentration per unit of volume A_i so that in dispersed bubbly hypothesis $A_i = \pi D^2$, the interfacial area concentration $a_i = \int f_d A_i \mathrm{d}V$ and the void fraction α. The relation between those variables is:

$$D_{sm} = \frac{\int f_d D^3 \mathrm{d}D}{\int f_d D^2 \mathrm{d}D} = 6\frac{\int f_d V \mathrm{d}V}{\int f_d A_i \mathrm{d}V} = \frac{6\alpha}{a_i}. \tag{1.16}$$

This relation arises from the dispersed hypothesis. It means that we have two options:

- either we take a user-defined diameter D_{sm} and let the void fraction play the crucial role of modeling the interfacial area concentration, presented in section 7.6.1,
- or we model directly the interfacial area concentration. It is often done by introducing a new transport equation for a_i, presented in 7.6.2.

1.2 Two-fluid and Drift-flux models

1.2.1 Equilibrium between phases

A new fundamental relation arises by summing the differential equations of each phase to get the differential equations for the mixture. Assuming that the subscript m denotes the quantity for the mixture, the relation can be expressed as follows:

$$\begin{aligned}\sum_k \overline{\rho_k\phi_k\mathbf{u}_k} &= \sum_k \alpha_k(\overline{\overline{\rho_k}}\,\overline{\overline{\phi_k\mathbf{u}_k}} + \overline{\overline{\rho'_k\phi_k\mathbf{u}_k}}) = \sum_k \alpha_k\left(\overline{\overline{\rho_k}}\tilde{\phi}_k\tilde{\mathbf{u}}_k + \overline{\overline{\rho_k\phi'_k\mathbf{u}'_k}}\right) \\ &= \underbrace{\rho_m\phi_m\mathbf{u}_m}_{\text{Mean transport}} + \underbrace{\sum_k \alpha_k\overline{\overline{\rho_k}}\tilde{\phi}_k\left(\mathbf{u}_k - \mathbf{u}_m\right)}_{\text{Phase transport}} + \underbrace{\sum_k \alpha_k\overline{\overline{\rho_k\phi'_k\mathbf{u}'_k}}}_{\substack{\text{Turbulent and} \\ \text{interfacial transport}}}.\end{aligned} \tag{1.17}$$

This relation provides the basis for separating the mixture and individual phases into distinct sets of equations. Two approaches stem from this separation: simulating only the mixture while modeling phase phenomena with a drift model, known as the *drift-flux* model, or simulating the behavior of each phase while incorporating turbulent and interfacial phenomena, known as the *two-fluid* model. According to **LivreIH**, the last approach offers several advantages, including the ability to simulate segregated dynamics and non-equilibrium interactions between phases. However, the drift-flux main advantage is that it drastically decrease the number of equations to only one set of equation for the mixture while the two-fluid must handle a set of equation for each phase.

To summarize:

- the *two-fluid* model handles a set of conservative equations for each phase. In TRUST/TrioCFD, it is referred as `Pb_multiphase`. Only the last term of equation 1.17 is modeled, i.e. turbulence and interfacial transport.

- The *drift-flux* model supposes mechanical equilibrium. When thermal equilibrium is also supposed in order to handle only one set of equations for the mixture, it degenerates to the well-known Homogeneous Equilibrium Model. In the framework of TRUST/TrioCFD, this model is referred as `Pb_multiphase_HEM`. The two last terms of equation 1.17 are modeled i.e. the drift between the phases, turbulence and interfacial transport along with a thermal jump condition between phases.

1.2.2 Example of modeling for liquid-gaz systems

Modeling the dynamics of two-phase flows presents significant challenges due to the diverse range of scenarios encountered, including steady and transient behaviors, as well as losses in homogeneity and thermal effects. One notable approach is the four-fields model, initially introduced by **Lahey2001**. The four-fields model partitions the flow into distinct regions, each associated with specific characteristics. These regions include the liquid phase, bubbly phase, droplet phase, and gas phase. Within this model, a continuous liquid field (lc) is managed to represent the presence of liquid, a continuous gas field (gc) captures the behavior of large gas pockets, a dispersed gas field (gd) models the dispersed bubbles, and a dispersed liquid field (ld) represents the behavior of liquid droplets (as illustrated in Figure 1.2).

While the four-fields model offers a comprehensive approach to capture the complex dynamics of two-phase flows, it can be computationally intensive. This model requires the solution of twelve balance equations, accounting for mass, momentum, and energy conservation. Additionally, closure laws are needed to describe the interfacial transfers and interactions between the different fields. Implementing the four-fields model accurately demands careful consideration of the closure laws and numerical techniques to ensure reliable predictions of the system behavior. To reduce the number of equations, **Morel2007** proposed two degenerate four-fields to two-fields models that combines the two-fluid and drift-flux approaches. The first proposition involves using two dispersed fields of bubbles carried by liquid (bm) and droplets carried by gas (dm), and then drifting the mixture made by the dispersed field. Another proposition is to use two separated fields for gas (g) and liquid (l), with each field being a hybrid of dispersed and continuous phases. The benefit is the reduction of the number of closure laws to 6 balance equations. However, these approaches have limitations, such as the inability to handle non-equilibrium physical phenomena.

Table 1.2 provides a summary of the current possibilities in TrioCFD two-phase flow computation.

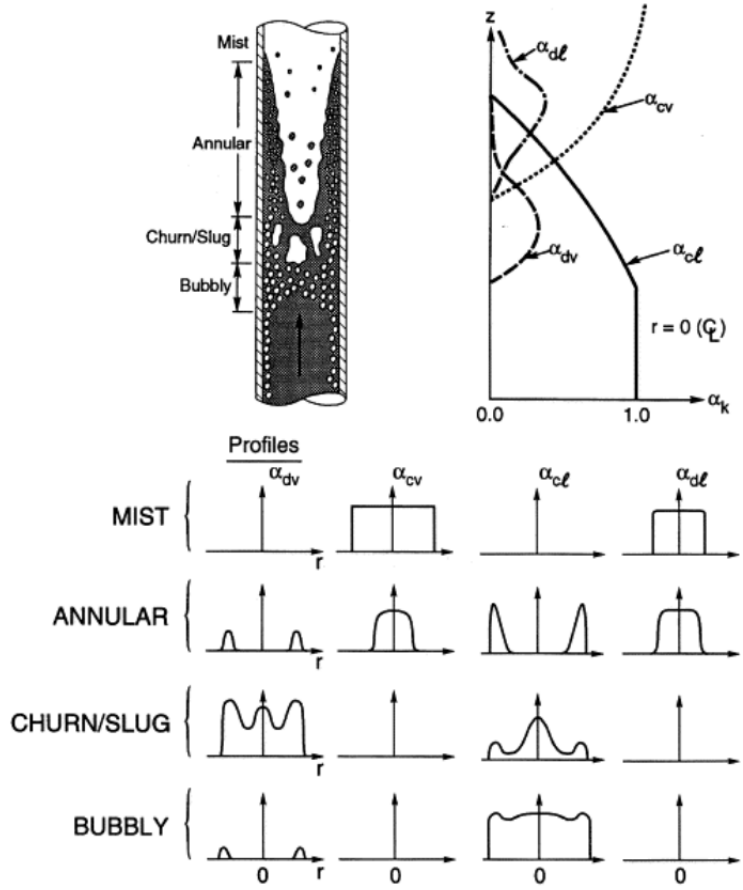


Figure 1.2: Principle of a four-field model handling any kind of topology of interest. From **Lahey2001**. The subscripts v and l refer respectively to vapor and liquid whereas d and c denote respectively the dispersed and the continuous phases.

Model	Set 1	Set 2	Set 3	Set 4	TrioCFD
Four Fields	α_{gd}	α_{gc}	α_{ld}	α_{lc}	Pb_multiphase
Hybrid Drift-flux	$\alpha_{bm} = \alpha_{gd} + \alpha_{lc}$	$\alpha_{dm} = \alpha_{ld} + \alpha_{gc}$			Pb_multiphase
Hybrid Two-fluid	$\alpha_g = \alpha_{gd} + \alpha_{gc}$	$\alpha_l = \alpha_{ld} + \alpha_{lc}$			✗
Bubbly Two-fluid	α_{gd}	α_{lc}			Pb_multiphase
Droplet Two-fluid	α_{ld}	α_{gc}			Pb_multiphase
Bubbly Drift-flux	$\alpha_{bm} = \alpha_{gd} + \alpha_{lc}$				Pb_multiphase_HEM
Droplet Drift-flux	$\alpha_{dm} = \alpha_{ld} + \alpha_{gc}$				Pb_multiphase_HEM

Table 1.2: Table of the different approaches. g denotes for gas, l denotes for liquid, d denotes for dispersed, c denotes for continuous and m denotes for mixture.

1.3 Turbulence equations

1.3.1 Single-phase Reynolds Averaged Navier-Stokes equations

The turbulent stress term is derived from the process of averaging the momentum equation of the Navier-Stokes equations. While the turbulence tends to expand the range of length scale, the averaged approach tends to erase the smallest scales. The Reynolds Averaged Navier-Stokes (RANS) simulates only the macroscopic scale and models the other scales. It is the cheapest

approach in terms of computational power, but it needs a high proportion of model to get complex behaviors.

For simplicity, we will focus on a single-phase, incompressible problem without any external sources. The momentum equation for this example can be expressed as follows:

$$\frac{\partial v_i}{\partial t} + \underbrace{v_i \frac{\partial v_i}{\partial x_k}}_{\text{Advection}} = \underbrace{-\frac{1}{\rho}\frac{\partial P}{\partial x_i}}_{\text{Pressure}} + \underbrace{\nu \frac{\partial^2 v_i}{\partial x_k \partial x_k}}_{\text{Viscosity}}. \tag{1.18}$$

In order to apprehend the averaging, we stand that the quantities v_i and P can be decomposed into a mean value U_i, $\bar{P}$ and a fluctuation u_i, p value so that $v_i = U_i + u_i$. By applying the mean to the previous equation we obtain:

$$\frac{\partial U_i}{\partial t} + \underbrace{U_i \frac{\partial U_i}{\partial x_k}}_{\text{Advection}} + \underbrace{\frac{\partial \overline{u_i u_k}}{\partial x_k}}_{\text{Turbulent stress}} = \underbrace{-\frac{1}{\rho}\frac{\partial \bar{P}}{\partial x_i}}_{\text{Pressure}} + \underbrace{\nu \frac{\partial^2 U_i}{\partial x_k \partial x_k}}_{\text{Viscosity}}. \tag{1.19}$$

The operating average filter preserves the fundamental structure of the instantaneous equation, with one notable exception: the emergence of the turbulent stress term, also called Reynolds stress, represented as $\overline{u_i u_j}$. This term captures the effects of turbulent fluctuations that occur within the flow. Thus, it becomes crucial to compute this term to properly account for these fluctuations.

If we subtract equation 1.19 to 1.18, we can get the equation Eq_i of the fluctuating velocity u_i. The Reynolds stress equation can be obtained by performing the operation $\overline{u_j Eq_i + u_i Eq_j}$:

$$\begin{aligned}\frac{\partial \overline{u_i u_j}}{\partial t} &+ \underbrace{U_i \frac{\partial \overline{u_i u_j}}{\partial x_k}}_{\text{Advection}} + \underbrace{\overline{u_i u_k}\frac{\partial \overline{U_j}}{\partial x_k} + \overline{u_j u_k}\frac{\partial U_i}{\partial x_k}}_{\text{Exchange mean-fluctuations}} + \underbrace{\frac{\partial \overline{u_i u_j u_k}}{\partial x_k}}_{\text{Triple correlation transport}} \\ &= \underbrace{\overline{\frac{1}{\rho} p \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)}}_{\text{Pressure redistribution}} - \underbrace{\frac{1}{\rho}\frac{\partial}{\partial x_k}\left(\overline{u_i p}\delta_{jk} + \overline{u_j p}\delta_{ik}\right)}_{\text{Pressure diffusion}} - \underbrace{2\nu \overline{\frac{\partial u_i}{\partial x_k}\frac{\partial u_j}{\partial x_k}}}_{\text{Pseudo-dissipation}} + \underbrace{\nu \frac{\partial^2 \overline{u_i u_j}}{\partial x_k \partial x_k}}_{\text{Molecular dissipation}}.\end{aligned} \tag{1.20}$$

This expression is an analytical equation for the Reynolds stress. However, 4 non-linear terms need models because they cannot be computed because they are not directly depending on the Reynolds stress or the mean flow. We then have 6 equations with 24 new unknowns. In order to get rid of some terms, one idea is to consider the turbulent kinetic energy $k = \frac{1}{2}\overline{u_i u_i}$. The equation then becomes:

$$\begin{aligned}\frac{\partial k}{\partial t} &+ \underbrace{U_i \frac{\partial k}{\partial x_k}}_{\text{Advection}} + \underbrace{\overline{u_i u_k}\frac{\partial U_i}{\partial x_k}}_{\text{Exchange mean-fluctuations}} + \underbrace{\frac{\partial \frac{1}{2}\overline{u_i u_i u_k}}{\partial x_k}}_{\text{Triple correlation transport}} \\ &= \underbrace{-\frac{1}{\rho}\frac{\partial}{\partial x_k}\overline{u_k p}}_{\text{Pressure diffusion}} \underbrace{-\nu \overline{\frac{\partial u_i}{\partial x_k}\frac{\partial u_i}{\partial x_k}}}_{\text{Dissipation}} + \underbrace{\nu \frac{\partial^2 k}{\partial x_k \partial x_k}}_{\text{Molecular dissipation}}.\end{aligned} \tag{1.21}$$

Three fundamental quantities can be notified from those equations and are presented in table 1.3. In this example, the instantaneous pressure also depends on the mean flow and the fluctuations. The Poisson equation gives the instantaneous value for the pressure. With incompressible

Quantity	Symbol	Expression
Turbulent kinetic energy	k	$\frac{1}{2}\overline{u_i u_i}$
Dissipation	ε	$\nu\overline{\frac{\partial u_i}{\partial x_k}\frac{\partial u_i}{\partial x_k}}$
Anisotropy	b_{ij}	$\frac{\overline{u_i u_j}}{2k} - \frac{1}{3}\delta_{ij}$

Table 1.3: Table of the fundamental turbulent quantities.

hypothesis and without external forces, the equation is as follows:

$$\frac{1}{\rho}\frac{\partial^2 P}{\partial x_i \partial x_i} = -\frac{\partial v_i}{\partial x_j}\frac{\partial v_j}{\partial x_i} = \underbrace{-S_{ij}S_{ji}}_{\substack{\text{Pure deformation}\\ \text{sink term}}} + \underbrace{\frac{\Omega^2}{2}}_{\substack{\text{Rotation}\\ \text{source term}}}, \tag{1.22}$$

with $S_{ij} = \frac{1}{2}(\frac{\partial v_j}{\partial x_i} + \frac{\partial v_i}{\partial x_j})$ and $\Omega = \nabla \times \mathbf{v}$.

By applying the mean to the previous equation we get:

$$\frac{1}{\rho}\frac{\partial^2 \overline{P}}{\partial x_i \partial x_i} = \underbrace{-\overline{S}_{ij}\overline{S}_{ji} + \frac{\overline{\Omega}^2}{2}}_{\text{Mean contribution}} \underbrace{-\overline{s}_{ij}\overline{s}_{ji} + \frac{\overline{\omega}^2}{2}}_{\substack{\text{Fluctuating}\\ \text{contribution}}}, \tag{1.23}$$

with s_{ij} the pure deformation from the fluctuating velocity and ω_{ij} the rotational of the fluctuating velocity.

As for the velocity, the operating average filter preserves the fundamental structure of the instantaneous equation, with one notable exception: the emergence of the fluctuating pressure.

Then by subtracting this equation to the instantaneous Poisson equation, we get:

$$\frac{1}{\rho}\frac{\partial^2 p}{\partial x_i \partial x_i} = \underbrace{-2\frac{\partial^2 u_i U_j}{\partial x_i \partial x_j}}_{\text{Slow linear term}} \underbrace{-\frac{\partial^2 (u_i u_j - \overline{u_i u_j})}{\partial x_i \partial x_j}}_{\text{Fast quadratic term}}. \tag{1.24}$$

The equation reveals a fundamental propriety: both the mean and fluctuating pressures depend on the velocity field's values at any given point and time. It emphasizes the relevance of investigating pressure effects to fully capture the proprieties of turbulence. Actually, advanced Reynolds stress models try to model those phenomena.

1.3.2 Single-phase RANS model equations

First order models

For industrial purposes, two types of model are commonly used based on the previous equations. The first type is the first-order models, which aim to estimate turbulence by computing the equations for the scalars k and ε, the turbulent kinetic energy and its dissipation respectively. They are often referred as the $k-\varepsilon$ models and follow a classical formulation for k:

$$\frac{\partial k}{\partial t} + \underbrace{U_i\frac{\partial k}{\partial x_i}}_{\text{Advection}} = \underbrace{\text{Diffusion}}_{\text{Model}} + \underbrace{\text{Production}}_{\text{Model}} - \underbrace{\text{Dissipation}}_{\text{Model}} + \underbrace{\nu\frac{\partial^2 k}{\partial x_k \partial x_k}}_{\substack{\text{Molecular}\\ \text{diffusion}}}. \tag{1.25}$$

To close the problem, we use an analogy between the shear stress and the turbulent shear stress. As Stokes did for the shear stress, the turbulent shear stress is determined using a diagonal

component equal to the turbulent kinetic energy (analogy with the pressure) and an extra diagonal component using the gradient of the *mean* velocity. A new proportionality coefficient arises: the turbulent viscosity ν_t. With this analogy, turbulence is seen as a strong diffusive phenomenon with a viscosity ν_t. This analogy has been proposed by Joseph BOUSSINESQ and is written as:

$$R_{ij} = \underbrace{-2\nu_t S_{ij}}_{\substack{\text{Diffusion} \\ \text{Phenomenon}}} + \underbrace{\frac{2}{3}k\delta_{ij}}_{\substack{\text{Fluctuacting} \\ \text{contribution} \\ \text{to pressure}}}, \tag{1.26}$$

with $S_{ij} = \frac{1}{2}\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)$ and $k = \frac{1}{2}\overline{u_i u_i}$. By putting this expression in equation 1.19, we obtain:

$$\frac{\partial U_i}{\partial t} + \underbrace{U_i \frac{\partial U_i}{\partial x_k}}_{\text{Advection}} = \underbrace{-\frac{\partial}{\partial x_i}\left(\frac{\bar{P}}{\rho} + \frac{2}{3}k\right)}_{\text{Modified pressure}} + \underbrace{(\nu + \nu_t)\frac{\partial^2 U_i}{\partial x_k \partial x_k}}_{\text{Diffusion}}. \tag{1.27}$$

The turbulent viscosity is then evaluated by:

$$\nu_t = C_\mu \frac{k^2}{\varepsilon}, \tag{1.28}$$

with C_μ a constant often taken as equal to 0.09.

The advantage of this model is its reduced reliance on differential equations compared to second-order models, resulting in lower CPU costs. It offers ease of computation and excellent stability. It accurately predicts shear stress in free sheared flows and free flows with strong turbulence, addressing well-documented errors. However, due to its linear behavior, this model cannot effectively capture the interaction between a wake and a mix layer, flows with pronounced curvature, and boundary layers. Additionally, a significant drawback of this model is the inherent positivity of the production term, leading to the well-known issue of overestimating turbulent kinetic energy in boundary layers prior to a stagnation point.

Second order models

The second type of model is known as the second-order model or Reynolds Stress Model (RSM). These models directly incorporate equations for the Reynolds tensor and the dissipation rate. One key advantage of the RSM is its ability to accurately capture nonlinear phenomena and avoid stagnation point anomalies, thanks to an analytical production term. This term plays a crucial role in various turbulent phenomena. For instance, in turbulent flows around a cylinder, it induces the redistribution of velocities from turbulence to the mean flow. However, the RSM has some limitations compared to the linear model. It is numerically less robust, meaning it may encounter stability issues during computations. Calibration of the RSM is also more challenging due to its nonlinear behavior, requiring more intricate adjustments and fine-tuning. The general equation of the model is given by :

$$\begin{aligned}\frac{\partial}{\partial t}R_{ij} + \underbrace{U_k \frac{\partial R_{ij}}{\partial x_k}}_{\text{Advection}} = \underbrace{-R_{ik}\frac{\partial U_j}{\partial x_k} - R_{jk}\frac{\partial U_i}{\partial x_k}}_{\text{Exchange with mean flow}} + \underbrace{\nu \frac{\partial^2 R_{ij}}{\partial x_k \partial x_k}}_{\text{Molecular diffusion}} \\ - \underbrace{\text{Dissipation}}_{\text{Model}} - \underbrace{\text{Triple correlation transport}}_{\text{Model}} + \underbrace{\text{Pressure redistribution}}_{\text{Model}}.\end{aligned} \tag{1.29}$$

Near-wall region

In the simulation of turbulent flows, the near-wall region is a crucial area that raises challenges for modeling. In this region, turbulence is weaker compared to viscosity, involving standard turbulent models failure. To address this issue, two approaches are commonly used: near-wall functions and near-wall damping effects.

The first approach involves implementing a law that describes the behavior of velocity in the near-wall region. It assumes the presence of a viscous boundary layer with a thickness denoted as δ for the dimensionless distance to the wall, $y^+ = \frac{y}{\delta}$. Typically for $k - \epsilon$ models, it is recommended to ensure that the first cell size in the near-wall region satisfies the condition of $30 < y^+ < 100$, as the model cannot be directly integrated all the way to the wall. This condition is mandatory to be in the logarithmic layer shown in Figure 1.3. These models are

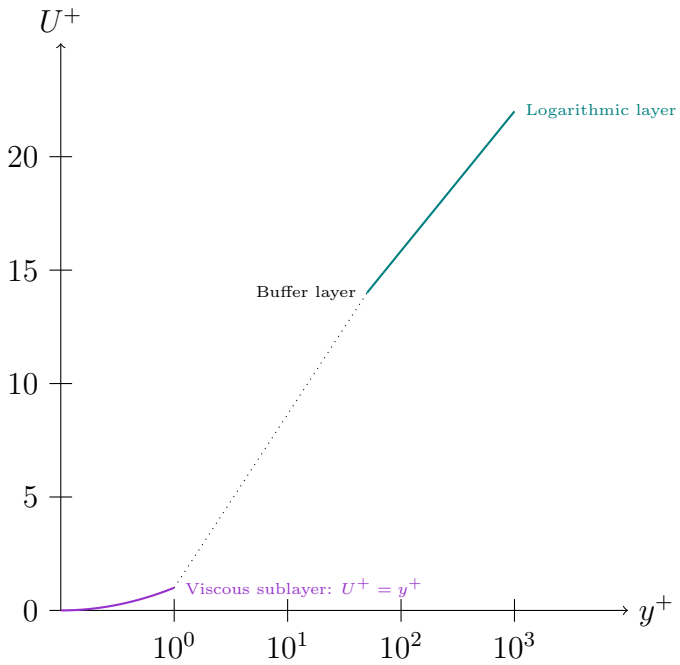


Figure 1.3: Principle of universal wall law.

typically referred to as high-Reynolds models. However, a significant challenge associated with this approach is the constraint it imposes on the flow's behavior near the wall, and the question of its universality for all flow configurations is still a topic of debate.

The second solution involves introducing a damping function or a new term that becomes active near the wall to damp the turbulence in that region. This approach allows achieving $y^+ \approx 1$, enabling the models to be integrated closer to the wall. Models that can be integrated all the way to the wall are typically referred to as Low-Reynolds models.

Chapter 2

Setting up data for a multiphase computation

This chapter describes how to build and complete the data file in order to perform a TrioCFD-multiphase computation. This chapter and the data file share the same structure, starting with a description of the domain (section 2.1), then the choice of discretizations in space (section 2.2) and time (section 2.3), which is then associated with the global problem 2.4 and the previously instantiated objects (section 2.5), to pursue with the description of the chosen model (section 2.6) before solving it (section 2.7).

Trust/TrioCFD allows to perform a multiphase flow computation using its submodule CMFD. For this reason, CFMD simulations are run with the same command, after having sourced TrioCFD.

```
$ trust filename.data
```

The file with the extension '.data' contains all the parameters that define the calculation. This file must comply with the TRUST data input file writing conventions [**trustonline**]. It is composed of keywords that will allow the creation of C++ objects. As a result, no C++ programming or compilation process is required to define a computation.
To run a parallel simulation, the same command as in Trust is used:

```
$ trust -partition filename.data [nprocs]
$ trust PAR_filename.data [nprocs]
```

For clarity, this section refers to a test case available in TrioCFD library: `Tube_solution_analytique_turbulent`.

2.1 Computation domain and mesh

The keywords have to be declared in a specific order. The first one is corresponding to the spatial dimension of the simulation

```
# Dimension can be 2D or 3D #
Dimension n
```

where **n** can be equal to 2 or 3. Pseudo 1D simulations can be performed with a 2D domain one cell wide.
In a second step, the user needs to create an instance of the `Domaine` class.

```
# Domain definition #
Domaine dom
```

In this example, the domain is named **dom**.
Note that the case of the object names is important, but not the case of the object keywords.
After the `Domaine` declaration, it must be associated with a mesh.
This can be done using the class Read_MED to use an external mesh file defined in the MED format.

```
# Read the mesh from MED file #
read_med { domain dom mesh mesh file 1_tube_analytique.med }
```

It is also possible to use the internal meshing tool from TRUST.
To perform 2D axi-symmetrical simulations, the keyword `bidim_axi` must be used before the mesh construction.

2.2 Discretization

The discretization method is defined as follows.

```
# Example of discretization on unstructured mesh #
PolyMAC_P0 dis option_PolyMAC_P0 { traitement_axi }
```

Multiple different discretizations can be used in the TrioCFD multiphase solver. More details on the mathematical formulations of these discretizations can be found in chapter 6.
The `VDF` scheme can be used with cartesian meshes and for 2D axi-symmetrical simulations. It is recommended to add `option_VDF  all_options`.
The `PolyMAC_P0` scheme can be used with any polyhedral mesh. Stability issues can arise if meshes contain deformed tetrahedrons. The scheme is also less costly when using hexahedrons than when using tetrahedrons. `option_PolyMAC_P0 { traitement_axi }` can be used to ignore no-slip boundary conditions in the calculation of the time step.
This can greatly increase the calculated time step in some situations.
To launch parallel computations, a partitionning step is required to split the domain in several zones. This step is managed with an external tool called metis.
The following commented sections are required for the partition to happen without error, but must not be modified by the user:

```
# BEGIN PARTITION
Partition dom
{
  /* Choose Nb_parts so to have ~ 25000 cells per processor */
  Partition_tool metis { nb_parts 4 }
  Larg_joint 2
  zones_name DOM
}
End
END PARTITION #

# BEGIN SCATTER
Scatter DOM.Zones dom
END SCATTER #
```

2.3 Define the time integration scheme

The next step is the definition of the time scheme. In the example the Euler explicit scheme is used. An instance of `Scheme_euler_explicit` called **sch** is created.

```
Schema_euler_implicite sch
Read sch
{
  tinit 0
  tmax 8
  dt_impr 1e-8
  facsec 1
  facsec_max 1
  nb_pas_dt_max 100
  seuil_statio 1e-3
  dt_start dt_fixe 2.e-5
  solveur SETS
  {
    criteres_convergence { alpha 1e-6 pression 1. vitesse 1e-5 temperature 1e8
        k 1e-5 omega 1. }
    iter_min 2
    solveur petsc cholesky { quiet }
    seuil_convergence_implicite 1e30
    facsec_diffusion_for_sets 100
  }
}
```

Two solvers can be used in multiphase triocfd: `ICE` and `SETS`. The mathematical details of these solvers are given in chapter 5. In short, the time step in `ICE` is limited by diffusion and convection. The time step in `SETS` is limited by convection in multi-phase flow and doesn't have a clear limit in single-phase flow. For single-phase flow, the use of `SETS` with a `facsec` between 10 and 100 is recommended. For adiabatic multi-phase flow, the use of `SETS` with a `facsec` of 1 but a `facsec_diffusion_for_sets` of 100, as in the case that is presented, is recommended. This enables the solver to ignore the diffusion stability limit and take only into account the limits on convection. For boiling flow, the use of `ICE` with a `facsec` of 1 is recommended. There are still glitches to iron out in the thermal equation for it to be possible to use `SETS` with a `facsec` 1. It is essential to use a small `dt_start`: the time step isn't always well calculated for the first time step as all variables aren't initialized. The convergence criteria can be adjusted to the situation. For adiabatic flow the criterion on temperature is often very loose for example. In some cases, making the criteria stricter can be detrimental to convergence, in others it can help as it can reduce some numerical noise. This must be adjusted on a case-to-case basis.

2.4 Problem definition and object association

A Problem is an object that define the set of equations that is solved during the simulation. Mutliphase TrioCFD simulations require the following keyword to build a two-fluid Euler-Euler set of equations.

```
# Problem definition #
Pb_Multiphase pb
```

2.5 Assosciate the instantiated objects

At this point the mesh, the domain and the numerical schemes are defined. It is necessary to link them through in the C++ executable, and discretize the problem:

```
Associate pb dom
Associate pb sch
Discretize pb dis
```

2.6 Complete model description

The physical properties, choice of closure laws, equations to solve (including initial and boundary conditions) and post-treatments are associated with the problem through the C++ class **Read**.

```
# Problem description #
Read pb
{
```

2.6.1 Fluid properties

First, a composite medium is declared. In this example, it consists of two incompressible phases and an interface with constant properties.

```
  # Physical characteristcs of medium #
  milieu_composite
  {
    liquide_eau Fluide_Incompressible
    {
      # Dynamic viscosity [kg/m/s] #
      mu champ_uniforme 1 1.e-3
      # Volumic mass [kg/m3] #
      rho champ_uniforme 1 1.e3
      # thermal conductivity [W/m/K] #
      lambda Champ_Uniforme 1 0.604
      # Heat capacity [J/m3/K] #
      Cp Champ_Uniforme 1 75.366
      # Thermal expansion (K-1) #
      beta_th Champ_Uniforme 1 0
    }
    gaz_air Fluide_Incompressible
    {
      # Dynamic viscosity [kg/m/s] #
      mu champ_uniforme 1 1.e-5
      # Volumic mass [kg/m3] #
      rho champ_uniforme 1 1.
      # thermal conductivity [W/m/K] #
      lambda Champ_Uniforme 1 0.023
      # Heat capacity [J/m3/K] #
      Cp Champ_Uniforme 1 1006
      # Thermal expansion (K-1) #
```

```
    beta_th Champ_Uniforme 1 0
  }
  interface_eau_air interface_sigma_constant
  {
    # surface tension J/m2
    tension_superficielle 0.0728
  }
}
```

When no external fluid properties are used, the units are free, but it is strongly recommended to use the units of the International System to avoid coherence problems in two-phase correlations that are not always dimensionless. However, when external fluid properties are used IS units must be chosen, apart from the temperatures that are defined in °C.
More details on the external fluid properties that can be used can be found in section 7.1.

2.6.2 Choice of closure laws

The correlation bloc contains the list of all closure laws that can be used in the problem. In this example, we have:

- An adaptive wall law for the turbulent boundary conditions (see chapter 8),
- A constant bubble diameter equal to 1mm,
- A constant interfacial heat flux (liquide_eau defines the coefficient from the liquid to the interface and gaz_air from the gas phase to the interface),
- A constant coefficient virtual mass,
- A constant coefficient interfacial friction,
- A constant coefficient interfacial lift force,
- A constant coefficient turbulent dispersion.

Details on these physical models can be found in chapter 7.

```
correlations
{
  loi_paroi adaptative { }
  diametre_bulles champ champ_fonc_xyz dom 2 0 0.001
  flux_interfacial coef_constant { liquide_eau 1e10 gaz_air 1e10 }
  masse_ajoutee coef_constant { }
  frottement_interfacial bulles_constant { coeff_derive 0.1 }
  portance_interfaciale constante { Cl 0.03 }
  dispersion_bulles constante { D_td_star 0.003 }
}
```

2.6.3 Momentum equation

The momentum equation contains an evanescence operator, that manages the momentum of a vanishing phase (see section 4). For CFD applications, it is recommended to set alpha_res between 10^{-5} and 10^{-6}. The pressure solver is discussed below. An upwind convection scheme

(named "amont" in french) is required for code stability.
In VDF, the `amont_vpoly` option must be implemented to handle the added mass term.
In laminar flow, no option needs to be added to the diffusion term.
In turbulent flow, the turbulence model must be specified.
For two-phase liquid-gas flow, the initial condition on velocity follows the order u_{lx}, u_{gx}, u_{ly}, u_{gy}, u_{lz}, u_{gz}. The order of the phases is set by the fluid properties declaration.
For complex geometries, it can be challenging to define an initial condition on velocity. In this case, the initial condition can be set to 0 and a velocity ramp enforced as a boundary condition.
`Paroi_frottante_loi` boundary condition is used to enforce a velocity wall law. `Symetrie` is no-penetration slip boundary condition.
A no-slip boundary condition can be enforced using the `paroi_fixe` keyword. The sources that are listed here are gravity and the interfacial forces.
Gravity is a momentum source, that is multiplied by $\alpha_k \rho_k$ before being put in the solver.
The interfacial forces defined here all call the correlation bloc for the specific closures, apart from the Antal correction that is standalone (see chapter 7).

```
QDM_Multiphase
{
  evanescence { homogene { alpha_res 1.e-5 alpha_res_min 5.e-6 } }
  solveur_pression petsc cholesky { quiet }
  convection { amont }
  diffusion { turbulente k_omega { } }
  initial_conditions
  {
    vitesse champ_fonc_xyz dom 6 0 0 0 0 0.52 0.52
    pression Champ_Fonc_xyz dom 1 1e5
  }
  conditions_limites
  {
    wall paroi_frottante_loi { }
    bottom frontiere_ouverte_vitesse_imposee_sortie Champ_front_Fonc_xyz 6 0
       0 0 0 0.52 0.52
    top frontiere_ouverte_pression_imposee champ_front_uniforme 1 1e5
    symetrie symetrie
  }
  sources
  {
    source_qdm Champ_Fonc_xyz dom 6 0 0 0 0 -9.81 -9.81 ,
    frottement_interfacial { } ,
    portance_interfaciale { beta 1 } ,
    Dispersion_bulles { beta 1 } ,
    Correction_Antal { }
  }
}
```

The pressure solver is used to solve the pressure reduction step in ICE and SETS numerical schemes (see 5). `Petsc cholesky`, used above, is a direct inversion method that yields results in parallel and sequential simulations that are identical to the numerical error, but is slower than iterative solvers.
To reduce calculation time, it is recommended to use the following options:

```
solveur_pression petsc cli_quiet
```

```
{
    -pc_type hypre
    -pc_hypre_type boomeramg
    -pc_hypre_boomeramg_strong_threshold 0.8
    -pc_hypre_boomeramg_agg_nl 4
    -pc_hypre_boomeramg_agg_num_paths 5
    -pc_hypre_boomeramg_max_levels 25
    -pc_hypre_boomeramg_coarsen_type PMIS
    -pc_hypre_boomeramg_interp_type ext+i
    -pc_hypre_boomeramg_P_max 2
    -pc_hypre_boomeramg_truncfactor 0.5
    -ksp_type fgmres
}
```

2.6.4 Mass equation

The mass equation doesn't contain a diffusion operator. The sum of volume fractions of each phase must be equal to one at all points in the initial and boundary conditions. A `flux_interfacial` source must be included in boiling flow.

```
Masse_Multiphase
{
  initial_conditions { alpha Champ_Fonc_xyz dom 2 0.9 0.1 }
  convection { amont }
  conditions_limites
  {
    wall paroi
    bottom frontiere_ouverte a_ext Champ_Front_fonc_xyz 2 0.9 0.1
    top frontiere_ouverte a_ext Champ_Front_fonc_xyz 2 0.9 0.1
    symetrie paroi
  }
  sources { }
}
```

2.6.5 Energy equation

The energy equation is a standard convection-diffusion equation. For now, it is always required in the dataset, even for adiabatic flows.

```
Energie_Multiphase
{
  equation_non_resolue 1
  diffusion { turbulente SGDH { sigma 0. } }
  convection { amont }
  initial_conditions { temperature Champ_Uniforme 2 0 0 }
  boundary_conditions
  {
    wall paroi_adiabatique
    bottom frontiere_ouverte T_ext Champ_Front_Uniforme 2 0 0
    top frontiere_ouverte T_ext Champ_Front_Uniforme 2 0 0
```

```
    symetrie paroi_adiabatique
  }
  sources
  {
    flux_interfacial
  }
}
```

Some recommendations about this setup. In turbulent thermal flow, it it recommended to use `turbulente SGDH { pr_t 0.9 }` for temperature diffusion. In boiling flow, the vapor temperature must be initialized at of over saturation temperature so that physical properties of the vapor can be calculated by the equations of state, even if the vapor fraction is zero at the beginning of the simulation. This is also true for boundary conditions. In multi-phase boiling flow, there are two possibilities for Neumann and Dirichlet boundary conditions at the wall. If no `flux_parietal` correlation is defined in the correlation bloc, the number of phases in the boundary condition must be the same as the total number of phases. If a `flux_parietal` correlation is defined, the boundary condition must have only one field though there are multiple phases. The `flux_parietal` condition then transfers the heat flux towards the different phases. A `flux_interfacial` source must be included in adiabatic flow for numerical stability, so that the temperature matrix has enough diagonal coefficients. In compressible or boiling flow, a `travail_pression` source must also be included.

2.6.6 Optional equations

Optional equations can be included in addition to the momemtum, mass and energy balances. These include turbulence (see chapter 8), interfacial area transport equations or population balance equations (see chapter 7.6).

```
taux_dissipation_turbulent
{
  diffusion { turbulente SGDH { sigma 0.5 } }
  convection { amont }
  initial_conditions { omega Champ_Fonc_xyz dom 1 13.85 }
  boundary_conditions
  {
    wall Cond_lim_omega_demi { }
    bottom frontiere_ouverte omega_ext Champ_Front_Uniforme 1 13.85
    top frontiere_ouverte omega_ext Champ_Front_Uniforme 1 13.85
    symetrie paroi
  }
  sources
  {
    Production_echelle_temp_taux_diss_turb { alpha_omega 0.5 } ,
    Dissipation_echelle_temp_taux_diss_turb { beta_omega 0.075 } ,
    Diffusion_croisee_echelle_temp_taux_diss_turb { sigma_d 0.5 }
  }
}
energie_cinetique_turbulente
{
  diffusion { turbulente SGDH { sigma 0.67 } }
  convection { amont }
```

```
  initial_conditions { k champ_fonc_xyz dom 1 0.0027 }
  boundary_conditions
  {
    wall Cond_lim_k_complique_transition_flux_nul_demi
    bottom frontiere_ouverte k_ext Champ_Front_Uniforme 1 0.0027
    top frontiere_ouverte k_ext Champ_Front_Uniforme 1 0.0027
    symetrie paroi
  }
  sources
  {
    Production_energie_cin_turb { } ,
    Terme_dissipation_energie_cinetique_turbulente { beta_k 0.09 }
  }
}
```

2.6.7 Post-processing

The fields and quantities that can be post-processed are described in detail in chapter 10. The post-processing bloc includes three sections. The first is used to define additional fields that can be used in the post-treatment. The distance to the edge and the pressure gradient are often useful to interpret simulation results. The `operateur_eqn` keyword can be used to record the value of source terms and operators. The output is divided by $\alpha_k \rho_k$. The first operator is diffusion and the second convection. The order of the sources is the same as in the sources bloc of the momentum equation. Fields can also be defined by arithmetic operations of other fields (see TRUST documentation [**trustonline**]). The second section concerns probes. The probes can be used to obtain the local values of any quantity. Their locations are written in the order x_{start}, y_{start}, x_{end}, y_{end}. The outputs are written in csv format in text files. The final section is the recording of complete fields. `lata`, `single_lata`, `cgns` and `lml` formats can be selected. These can be chosen depending on the reading software used and the size of the simulation. Interesting fields to capture include the unkowns, the unkowns per phase, the unknown residuals, physical properties, turbulent viscosity and any user-defined field.

```
Postraitement
{
  Definition_champs
  {
    d_paroi refChamp { Pb_champ pb distance_paroi_globale }
    gradient_p refChamp { Pb_champ pb gradient_pression }
    diff operateur_eqn { numero_op 0 sources { refChamp { pb_champ pb vitesse
        } } }
    grvv operateur_eqn { numero_source 0 sources { refChamp { pb_champ pb
       vitesse } } }
  }
  sondes
  {
    vitesse vitesse periode 1.e-1 segment 10 0 1 0.01 1
  }
  format lata champs binaire dt_post 1.e-1
  {
    alpha elem
```

```
    vitesse elem
    vitesse_liquide_eau elem
    vitesse_residu elem
    pression elem
    nu_turb_liquide_eau elem
    enthalpie elem
  }
}
```

The final brace of the read must not be forgotten!

```
}
```

2.7 Solve the problem

The last step consists in solving the problem. This is done by including the following line.

```
Solve pb
```

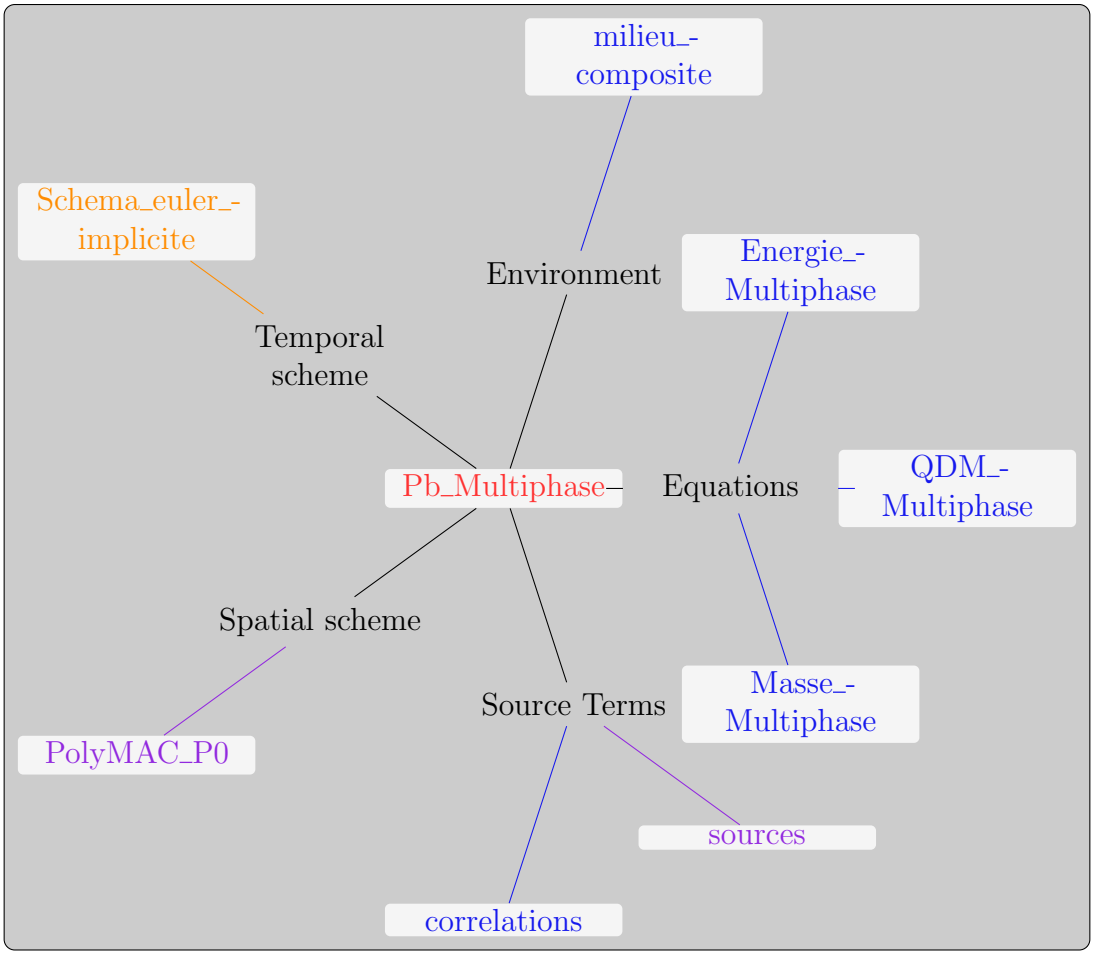


Figure 2.1: Overview of the dataset

Chapter 3

Architecture of Pb_Multiphase

This chapter aims to describe how the software is programmed and managed within the TrioCFD/TRUST and physical framework. We first present the numerical problems available (section 3.1), which then give rise to a numerical description of the system of equations presented in section 3.2. This system is stored in matrices, whose filling principle is described in the section 3.3. The various code storage locations are explained in the section 3.4. Finally, a description of the TrioCFD multiphase GUI prototype for beginners is proposed in section 3.5.

3.1 The problem

In order to launch the most efficient and suitable numerical simulation possible, it is important to specify what type of equation system one wishes to use. This is referred to as a Problem (Pb) in TRUST/TrioCFD. This Problem is equivalent to a module in other CFD codes. Thus, depending on the type of Problem chosen, different sets of equations are obtained, solved over the domain, so it is crucial to choose it correctly. The different Problems are documented in the Trust Reference Manual at https://cea-trust-platform.readthedocs.io/en/latest/.
TrioCFD multiphase corresponds to the multiphase averaged Euler-Euler equation set for motion. Two types of problems are currently available:

- `Pb_multiphase` : it allows solving N-phases with a set of 3*N equations (mass, energy, and momentum for each phase). This modeling corresponds to the Two-fluid described in section 1.2.
- `Pb_multiphase_HEM` : it allows solving 2 mechanically and thermally coupled phases at equilibrium via a single set of 3 equations (mass, energy, and momentum for the mixture). This modeling corresponds to the Drift flux with thermal equilibrium described in section 1.2.

`Pb_multiphase_HEM` is notably a reduction of the Pb_multiphase system via an operator called Evanescence, described in section 4. The use of this problem is equivalent to adding the following line to the data set in a `Pb_multiphase` :

```
evanescence { homogene { alpha_res 1 alpha_res_min 0.5 } }
```

3.2 The equations

Here is a description of the structure of `Pb_multiphase` set of equations in the code.
A structural hypothesis is that the sum of void fractions α_k is such that (see equation 1.3):

$$\sum_k \alpha_k = 1 \tag{3.1}$$

Then, the governing equations of the model used in Pb_multiphase can be written as (refer to equations 1.7, 1.8 and 1.9 respectively for mass, momentum and energy equations):

$$(\mathcal{M}_k) \quad \frac{\partial \alpha_k \rho_k}{\partial t} + \underbrace{\nabla \cdot (\alpha_k \rho_k \overrightarrow{u_k})}_{\text{convection}} = \underbrace{\Gamma_k}_{\text{sources}} \tag{3.2}$$

$$(\mathcal{Q}_k) \quad \begin{aligned} \frac{\partial \alpha_k \rho_k \overrightarrow{u_k}}{\partial t} + \underbrace{\underline{\nabla} \cdot (\alpha_k \rho_k \overrightarrow{u_k} \otimes \overrightarrow{u_k})}_{\text{convection}} = & \underbrace{- \alpha_k \underline{\nabla} P}_{\text{solveur\_pression}} + \underbrace{\underline{\nabla} \cdot (\alpha_k \mu_k \underline{\underline{\nabla}} \overrightarrow{u_k})}_{\text{diffusion}} \\ & \underbrace{- \underline{\nabla} \cdot (\alpha_k \rho_k \overline{u_i' u_j'})}_{\text{diffusion}} + \underbrace{\overrightarrow{F}_{ki} + \overrightarrow{F}_k}_{\text{sources}} \end{aligned} \tag{3.3}$$

$$(\mathcal{E}_k) \quad \begin{aligned} \frac{\partial \alpha_k \rho_k e_k}{\partial t} + \underbrace{\nabla \cdot (\alpha_k \rho_k e_k \overrightarrow{u_k})}_{\text{convection}} = & \underbrace{\nabla \cdot \left(\alpha_k \lambda_k \underline{\nabla} T - \alpha_k \rho_k \overline{u_i' e_k'} \right)}_{\text{diffusion}} \\ & \underbrace{- p \left(\frac{\partial \alpha_k}{\partial t} + \nabla \cdot (\alpha_k \overline{u_k}) \right)}_{\text{sources}} + \underbrace{q_{ki} + q_{kp}}_{\text{sources}} \end{aligned} \tag{3.4}$$

Equation	Mass	Momentum	Energy
Keyword	Masse_Multiphase	QDM_Multiphase	Energie_Multiphase
Abbreviations	$(\mathcal{M}_k)$	$(\mathcal{Q}_k)$	$(\mathcal{E}_k)$
Unknown	α_k	$p, \overrightarrow{u_k}$	T
Conservation	$\rho_k \alpha_k$	$\rho_k \alpha_k \overrightarrow{u_k}$	$\rho_k \alpha_k e_k$
Operators	convection	convection , diffusion , solveur_pression	convection , diffusion

Table 3.1: Solved equations in TrioCFD multiphase module.

A conservation equation can be seen as a time-change term for a conserved quantity varying according to the physical models we use (see equation 1.1). Among those models are convection, diffusion, and source terms. Thus, in the code, an equation corresponds to a time-dependent term for a conserved quantity. It is then associated with keywords allowing to fill in the equation. This is the case for the previous equations 3.2, 3.3 and 3.4, where the conserved quantities and the models are partially given in Table 3.1.

3.2.1 The correlation block and sources

The correlation bloc contains the list of all closure laws and models for the right-hand side of the equation that can be used in the equations as sources. Thus, all terms that are not related to convection or diffusion or solveur_pression of the conserved field must inherit from `Correlation_base`.
Once notified in the correlation block, the models can be used as a source for the equations. The source terms are semi-implicit to manage the method's tables, notably the availability of derivative calculation during the method. The only purely implicit term is interfacial friction. Regarding time resolution methods described in section 5, it should be noted that in ICE/SETS, all other terms are semi-implicit. In PAT, all terms are implicit except for turbulent dispersion due to the void fraction gradient.

3.2.2 Turbulent viscosity

It is a `Correlation_base`. The value turbulent viscosity field is filled with the `eddy viscosity` method. If we want to use a more complex form, we can use the Viscosite_turbulente_multiple which uses the reynolds_stress method. It is used by the WIT, WIF or sato model, described in section 8.3.3. It might be useful to generalise this to the Visco_turbu_base.

3.3 Principle of matrix filling

The implementation of new source terms is based on the principle of filling partial derivative matrices using the chain rule method. The chain rule is a fundamental concept in calculus that allows us to find the derivative of a composite function. It is particularly useful when dealing with functions of several variables. Consider a function $f(x_1, x_2, \ldots, x_n)$ which depends on n variables. If each of these variables x_i depends on another variable t, i.e. $x_i = g_i(t)$, then we can define a composite function $F(t) = f(g_1(t), g_2(t), \ldots, g_n(t))$. Then we can compute an infinitesimal change dF as:

$$dF = \sum_i \frac{\partial F}{\partial g_i} \delta g_i \tag{3.5}$$

The chain rule describes how variations in the independent variable t are transmitted through the dependent variables x_i, ultimately influencing changes in the composite function F. In our case $F(t) = f(\alpha(t), T(t), P(t), \overrightarrow{u}(t))$. We then have:

$$dF = \frac{\partial F}{\partial \alpha}\delta\alpha + \frac{\partial F}{\partial P}\delta P + \frac{\partial F}{\partial T}\delta T + \frac{\partial F}{\partial \overrightarrow{u}}\delta\overrightarrow{u} \tag{3.6}$$

This gives us how to fill respectively the void fraction, temperature, pressure and velocity dependance matrices M_α, M_T, M_P and M_u and the right-hand side of the equation (called `secmem` in the code, French for RHS) so that:

$$F^{n+1} = \underbrace{F^n}_{\text{secmem}} + \underbrace{\frac{\partial F}{\partial \alpha}}_{-M_\alpha}\delta\alpha + \underbrace{\frac{\partial F}{\partial P}}_{-M_P}\delta P + \underbrace{\frac{\partial F}{\partial T}}_{-M_T}\delta T + \underbrace{\frac{\partial F}{\partial \overrightarrow{v}}}_{-M_u}\delta\overrightarrow{u} \tag{3.7}$$

For example, we can consider the pressure term in the energy equation 3.4. Its expression is :

$$-P\left(\frac{\partial \alpha_k}{\partial t} + \nabla\cdot(\alpha_k u_k)\right) \tag{3.8}$$

Regarding the temporal change $P\frac{\partial \alpha}{\partial t}$ (similar for the divergence but easier to write here), if we perform the chain rule, one can get :

- $F^n = P^n \frac{\Delta\alpha}{\Delta t}$,
- $\frac{\partial F}{\partial \alpha} = P^n \frac{1}{\Delta t}$
- $\frac{\partial F}{\partial P} = \frac{\Delta\alpha}{\Delta t}$,
- $\frac{\partial F}{\partial T} = 0$,
- $\frac{\partial F}{\partial u} = 0$.

Then, we can fill the matrix. We add in the right-hand side of the energy equations :

$$\texttt{secmem} = -P^n \times \frac{\alpha_k^n - \alpha_k^{n-1}}{\Delta t} \tag{3.9}$$

In the void fraction matrix :

$$M_\alpha = \frac{P^n}{\Delta t} \tag{3.10}$$

In the pressure matrix :

$$M_P = \frac{\alpha_k^n - \alpha_k^{n-1}}{\Delta t} \tag{3.11}$$

In the energy equation, it means that this term is :

$$M_T = -P^n \times \frac{\alpha_k^n - \alpha_k^{n-1}}{\Delta t} - \frac{\alpha_k^n - \alpha_k^{n-1}}{\Delta t}\delta P - \frac{P^n}{\Delta t}\delta\alpha_k \tag{3.12}$$

Let's introduce $\alpha_k^n - \alpha_k^{n-1} = \Delta\alpha_k$ and remind that $\delta P = P^{n+1} - P^n$. We then have :

$$- P^n \times \frac{\Delta\alpha_k}{\Delta t} - \frac{\Delta\alpha_k}{\Delta t}(P^{n+1} - P^n) - \frac{P^n}{\Delta t}\delta\alpha_k = -P^{n+1}\frac{\Delta\alpha_k}{\Delta t} - P^n\frac{\delta\alpha_k}{\Delta t} \sim -(P\frac{\partial\alpha_k}{\partial t})^{n+1} \tag{3.13}$$

Warning pressure reduction

Not all dependencies are allowed. Indeed, in order to perform pressure reduction, it is mandatory to keep empty matrices for dependencies on α and T. Thus, if the source term F is in the momentum conservation equation, it can only depend on velocity and pressure $F(P, \overrightarrow{u})$ (see Tchen force). In summary :

- In mass equation (with the unknown α), $F(\alpha, T, P, \overrightarrow{u}) \Leftrightarrow (M_\alpha, M_T, M_P, M_u)$ filling
- In energy equation (with the unknown T), $F(\alpha, T, P, \overrightarrow{u}) \Leftrightarrow (M_\alpha, M_T, M_P, M_u)$ filling
- In momentum equation (with the unknown $\overrightarrow{v}$), $F(P, \overrightarrow{u}) \Leftrightarrow (M_P, M_u)$ filling otherwise no pressure reduction allowed.

Furthermore, only dependence matrices M_p and M_u (except in momentum equation for the velocity) can be filled with non diagonal terms. In other words, M_α and M_T are always diagonal for all transported variables (including turbulence and diameter) and M_u in the momentum equation is always diagonal.

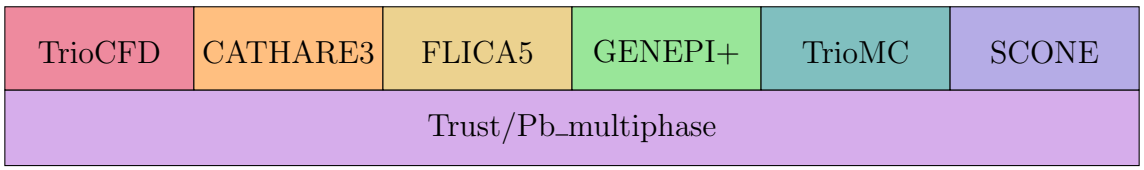
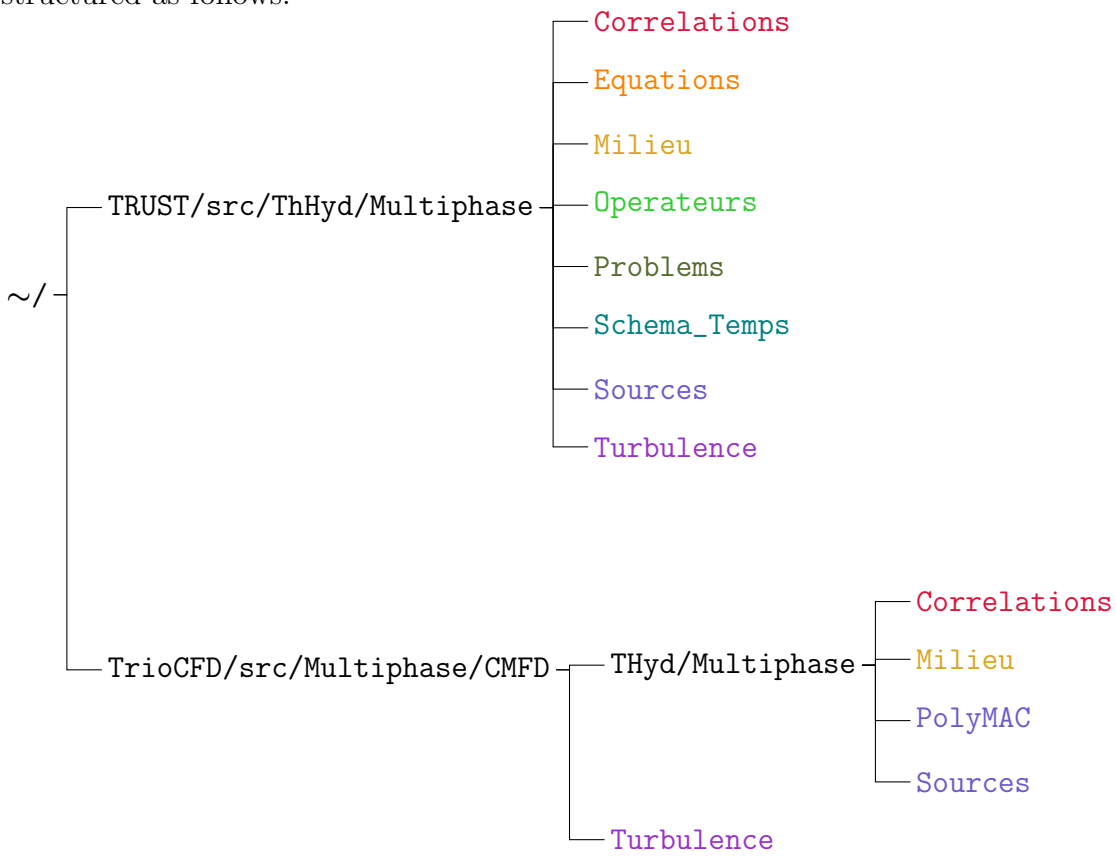


Figure 3.1: `Pb_multiphase` and its baltiks.

3.4 Folder structure

The software inherits the complex structure of TrioCFD, which shares part of its source code with TRUST. The aim of this section is to describe the distribution of lines of code linked to TrioCFD multiphase. TrioCFD multiphase is structured into 2 parts. The first part inherits the structuring of the CFD codes from CEA with the TRUST base. The `Pb_multiphase` from TRUST is common to several CEA codes as depicted in Figure 3.1. The specific parts of TrioCFD multiphase is located in the TrioCFD and Trust folders. The different parts are structured as follows:



3.4.1 TrioCFD multiphase in TRUST folders

Some lines of code are located in the TRUST folders. The available multiphase models in TRUST folders are :

```
Trust/src/ThHyd/Multiphase/Correlations
├── Changement_phase_base
│   └── Changement_phase_Silver_Simpson →
├── Coalescence_bulles_1groupe_base
├── Diametre_bulles_champ
│   └── Diametre_bulles_constant → Trio
└── Dispersion_bulles_base
```

```
    |__Dispersion_bulles_constante → Trio
|__Flux_interfacial_base
    |__Flux_interfacial_Chen_Mayinger →
    |__Flux_interfacial_Coef_Constant →
    |__Flux_interfacial_Kim_Park →
    |__Flux_interfacial_Ranz_Marshall →
    |__Flux_interfacial_Wolfert_composant →
    |__Flux_interfacial_Wolfert →
    |__Flux_interfacial_Zeitoun →
|__Flux_parietal_base
|__Frottement_interfacial_base
    |__Frottement_interfacial_bulles_composant →
    |__Frottement_interfacial_bulles_constant → Trio
    |__Frottement_interfacial_Garnier →
    |__Frottement_interfacial_Ishii_Zuber → Trio
    |__Frottement_interfacial_Rusche →
    |__Frottement_interfacial_Simonnet →
    |__Frottement_interfacial_Sonnenburg →
    |__Frottement_interfacial_Tomiyama → Trio
    |__Frottement_interfacial_Wallis →
    |__Frottement_interfacial_Weber →
    |__Frottement_interfacial_Zenit →
|__Gravite_Multiphase
|__Masse_ajoutee_base
    |__Masse_ajoutee_Coef_Constant → Trio
    |__Masse_ajoutee_Wijngaarden →
    |__Masse_ajoutee_Zuber → Trio
|__Multiplicateur_diphasique_base
    |__Multiplicateur_diphasique_Friedel →
    |__Multiplicateur_diphasique_homogene →
    |__Multiplicateur_diphasique_Lottes_Flinn →
    |__Multiplicateur_diphasique_Muhler_Steinhagen →
|__Portance_interfaciale_base
    |__Portance_interfaciale_Constante → Trio
    |__Portance_interfaciale_Sugrue → Trio
    |__Portance_interfaciale_Tomiyama → Trio
|__Rupture_bulles_1groupe_base
|__Vitesse_derive_base
    |__Vitesse_derive_constante → Trio
    |__Vitesse_derive_Forces →
    |__Vitesse_derive_Ishii →
|__Vitesse_relative_base
```

The available multiphase validation test cases in TRUST folders are located in **TRUST/Validation/Rapports_automatiques/Verification/Multiphase** :

Regarding table 3.2, the test cases with no cross are used to validate textttPb_multiphase general coding.

Folder name	TrioCFD	Cathare3	Flica5	GENEPI+	TrioMC	Scone
buoyant_driven_cavity						
buoyant_driven_cavity_two_phase						
canal_axi						
canal_axi_two_phase						
canal_bouillant						
canal_bouillant_void_drift						
comparaison_lois_eau_diphasique						
comparaison_lois_eau_monophasique		✓	✓			
conduction_two_phase		✓	✓			
fluide_reel						
mp_simple						
Ransom_StiffenedGas_MUSIG						✓
sedimentation						
tubes_a_choc						
Tube_solution_analytique	✓					

Table 3.2: Test cases in TRUST folder for specific baltik validation.

3.4.2 TrioCFD multiphase in TrioCFD folders

The available multiphase models in Trio folders are located in **TrioCFD/src/Multiphase/CMFD/Th-Hyd/Multiphase/Correlations:**

- Coalescence_bulles_1groupe_Yao_Morel
- Dispersion_bulles_turbulente_Bertodano
- Dispersion_bulles_turbulente_Burns
- Dispersion_bulles_turbulente_constante
- Flux_parietal_Kommajosyula
- Flux_parietal_Kurul_Podowski
- Frottement_interfacial_Ishii_Zuber_Deformable
- Portance_interfaciale_Sugrue_Modifiee
- Rupture_bulles_1groupe_Yao_Morel
- Vitesse_derive_Spelt_Biesheuvel

The available multiphase validation test cases in Trio folders are located in **TrioCFD/share/-Validation/Rapports_automatiques/Multiphase/CMFD:**

- canal_plan_pb_multi
- conduite_circulaire_vef_vdf_polymac
- CoolProp_Single_phase_Debora
- decroissance_ktau

- expansion_NO_MASS
- expansion_WITH_MASS
- Gabillet
- marche_descendante
- shock_dodecane
- Tube_solution_analytique_turbulent
- Verification1D_drift-flux_models
- Verification_k_tau_omega_transport_equation
- Verification_wall_laws

3.5 Graphical User Interface for beginners

Given the recent development of TrioCFD multiphase, there is currently no dedicated training available. Additionally, there are only two TrioCFD training sessions per year. To partially address these issues and to ensure that the initial learning curve is not too steep, a prototype of a TrioCFD multiphase Graphical User Interface (GUI) for beginners has been created (see Figure 3.2). This GUI does not cover all available options or application cases (only 2 fluids). However, it should help users quickly grasp the concept of data sets. For instance, a first exercise could be to replicate the data set presented in Chapter 2 using the GUI. This GUI should be available in TrioCFD multiphase folders soon. For more information or if the app is not yet accessible, one can send an e-mail to the TrioCFD multiphase team. The GUI is coded in Python and may require the installation of the following packages: customtkinter, tkinter, and PIL. To launch the GUI, please find yourself in the right folder and do the command : python3 GUI. Its straightforward approach is based on reading and writing files, allowing users to fill in the data file step by step without loosing every files. Generally, users just need to click as many blue buttons as possible. However, red buttons should be used with caution. To complete the file, simply interact with all the buttons on the left side of the main interface. Don't forget to save at each submenu and to push the End datafile button when finished. The data file visualization window on the right is especially useful for checking that the data file is being filled out correctly.

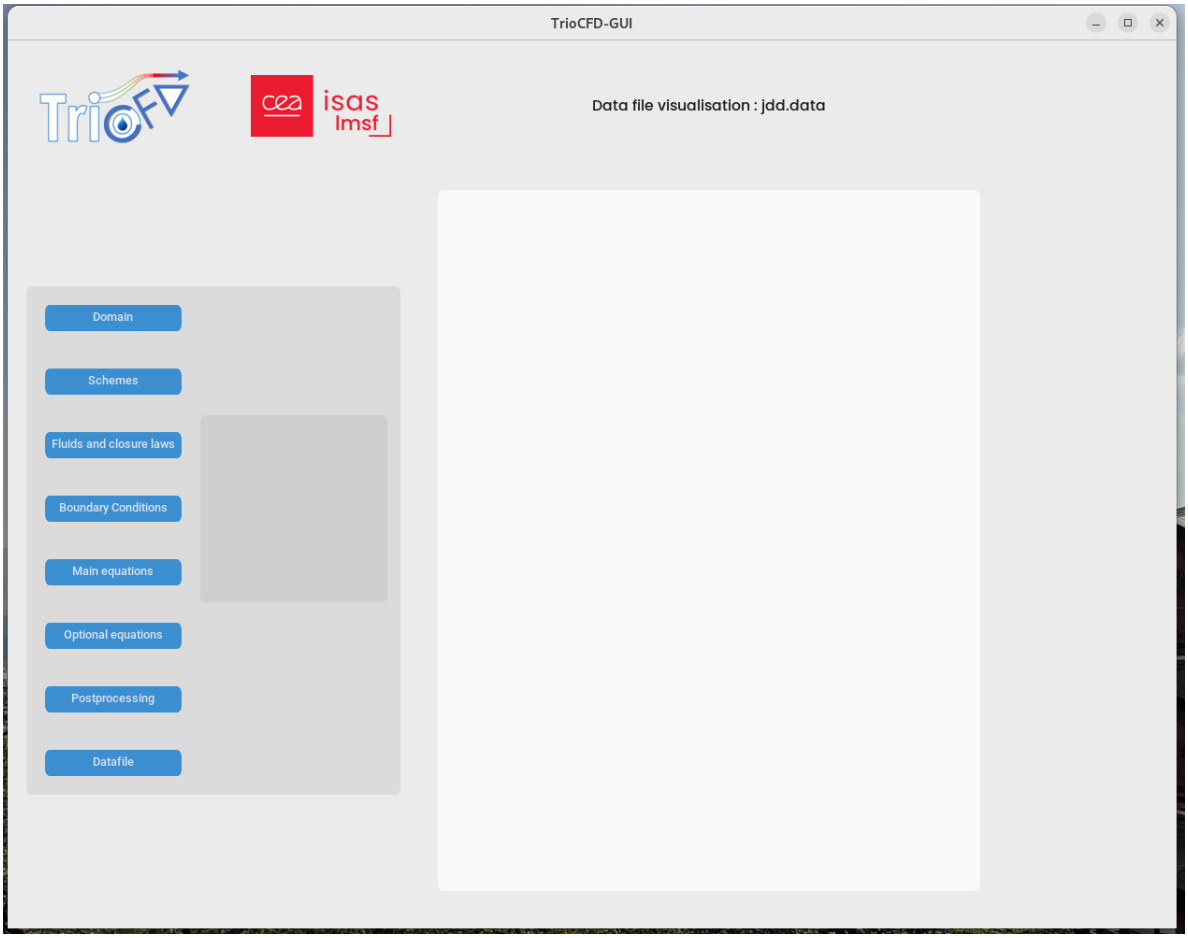
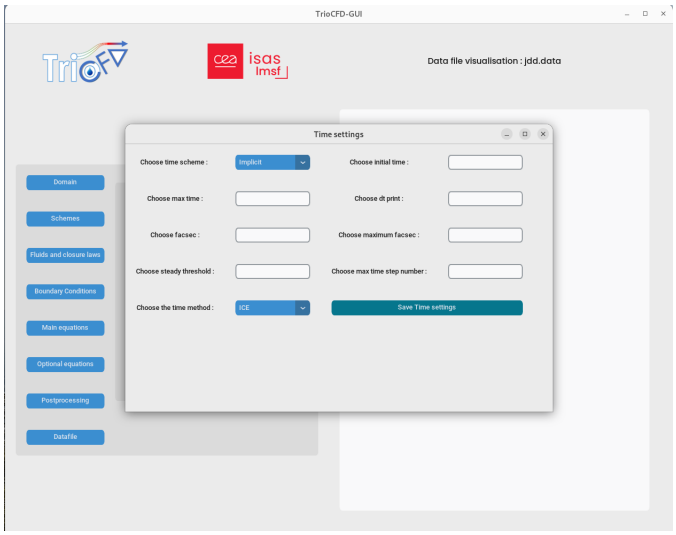
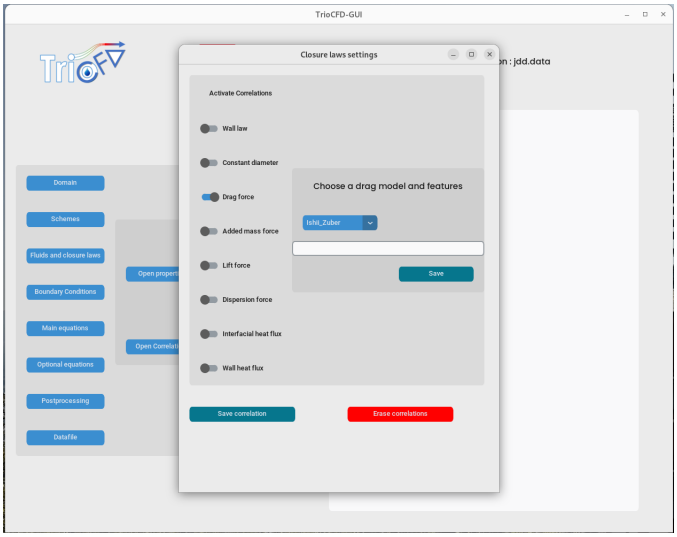


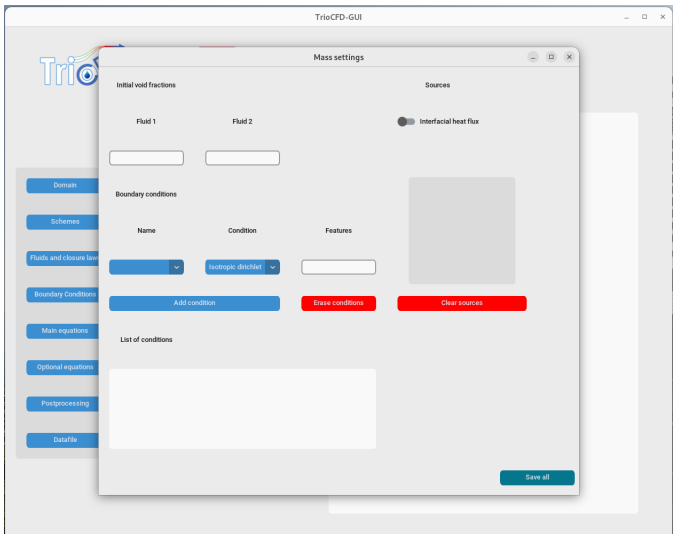
Figure 3.2: Visualization of TrioCFD multiphase GUI.



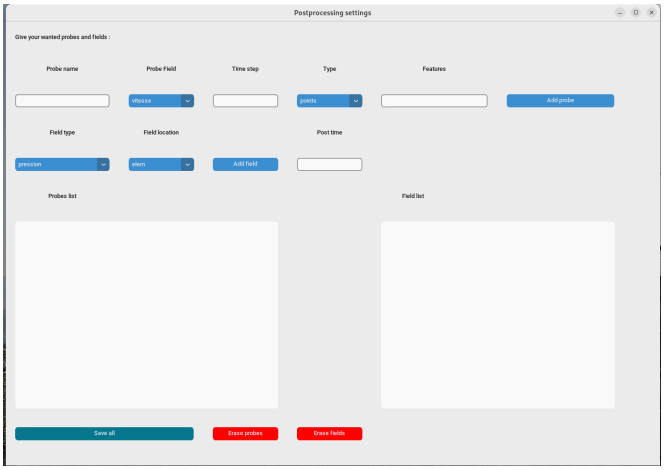
(a) Time settings of GUI.



(b) Closure laws of GUI.



(c) Mass equation settings of GUI.



(d) Post-processing settings of GUI.

Figure 3.3: Snapshots of different sub-menus of the GUI.

Chapter 4

Evanescence operator

This chapter aims to explain a class that is central to the management of phases and models in TrioCFD multiphase: the evanescence operator. The first section serves to establish the basic vocabulary and concepts related to the operator (section 4.1). The next three sections explain the operator's action on the momentum (section 4.2) and energy equations (section 4.3) as well as a specific explanation of flux management (section 4.4). The last two sections focus on how this function is called in datasets (4.5) and the basics of their implementation in the code (section 4.6).

4.1 Objectives

Evanescence is an operator in term of TRUST library. This operator can be apply to the momentum or energy equation, it consists in a relaxation of a minority phase property (*i.e.* velocity or temperature) toward the property of a so-called predominant phase. Evanescence operator is systematically attached in the data-set to the momentum equation to ensure numerical stability. It is optional in the energy equation. The following sections explain the purpose of this operator, the data-set syntax and some details about its implementation. Evanescence operator is defined whatever the number of phases so the following explanations are generic. Nevertheless, it is necessary to define the following phases:

- the minority phase identified by the subscript *mino*,
- the predominant phase identified by the subscript *pred*
- and i refers to a non-specific phase.

This chapter deals with all the aspects link to the evanescence operator. It is organized as follow. The relaxation of the momentum and energy equations are detailed in section 4.2 and 4.3 respectively. Section 4.4 explains the process allowing to limit the interface face flux to avoid e negative volume fraction. Finally, sections 4.5 and 4.6 give information on the setup and coding of the evanescence process.

4.2 Evanescence operator and momentum equation

The multiphase resolution of TrioCFD is based on a non-conservative form of the momentum transport equations as v_i is evaluated instead of $\alpha_i \rho_i v_i$. This specificity can leads to important divergence of the velocity if the void fraction α_i becomes locally low. Evanescence operation can be applied to the momentum equation to overcome this difficulty. It enforces the coherence between the minority and the predominant phases by imposing a relation between the velocities

of the minority phase and predominant phases. It also works if $\alpha_k = 0$ locally, even though in this case the momentum of the minority phase is zero. Thus, it is necessary to enforce a specific relation on the minority phase velocity $v_{\text{mino}} = v_{\text{pred}} + v_{\text{drift}}$. By default v_{drift} is null, but this value can be set through a correlation (see section 7). To parameterize the relaxation of v_{mino} with respect to v_{pred}, it is necessary to define a threshold value of alpha beyond which a minority phase can be identified as null. This value is noted $\alpha_{\text{res,min}}$. By essence, the condition $\alpha_i < \alpha_{\text{res,min}}$ can occur intermittently both spatially and temporally. To prevent numerical instabilities, a second threshold α_{res} is defined to relax v_{mino} toward $v_{\text{pred}} + v_{\text{drift}}$. By definition, this second threshold α_{res} must be higher than $\alpha_{\text{res,min}}$. This operation can be highlight in a mathematical framework considering $\mathcal{Q}_i$ the momentum of a the i^{th} phase

$$\begin{pmatrix} \mathcal{Q}_{\text{pred}} \\ \mathcal{Q}_{\text{mino}} \end{pmatrix} \Longrightarrow \begin{pmatrix} \mathcal{Q}_{\text{pred}} + c\mathcal{Q}_{\text{mino}} - c\left(v_{\text{mino}} = v_{\text{pred}} + v_{\text{drift}}\right) \\ (1-c)\,\mathcal{Q}_{\text{mino}} + c\left(v_{\text{mino}} = v_{\text{pred}} + v_{\text{drift}}\right) \end{pmatrix} \tag{4.1}$$

with c defined as

$$c = \min\left(1,\ \max\left(0,\ \frac{\alpha_{\text{res}} - \alpha_{\text{min}}}{\alpha_{\text{res,min}} - \alpha_{\text{res}}}\right)\right). \tag{4.2}$$

The evolution of the parameter c is illustrated Figure 4.1. This operator is coded within the class `Op_Evanescence_Homogene_Face_base` in TRUST.

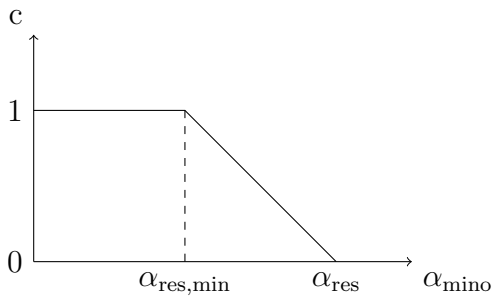


Figure 4.1: Evolution of the coefficient c as a function of α_{mino}.

Considering the evolution of the coefficient, the system 4.2 can be described as follow:

- $\alpha_{\text{res}} < \alpha_{\text{mino}}$: the system is unchanged.
- $\alpha_{\text{res,min}} < \alpha_{\text{mino}} < \alpha_{\text{res}}$: partial relaxation of the velocity of the minority phase toward the velocity of the majority phase. The momentum equation solved for the minority phase is a relaxation equation that brings the velocity of the minority phase to that of the majority phase.
- $\alpha_{\text{mino}} < \alpha_{\text{res,min}}$: the system is fully reduced by summing the momentum equation of α_{mino} with α_{pred}.

In other words, when α of the minority phase is less than `alpha_res_min` (i.e., $c = 1$), we are in pure HEM, whereas when α of the minority phase is greater than `alpha_res`, we are in the Two-fluid model. Thus, when we are between the two, we are in a linear combination of the two cases (relaxation) to avoid making the transition abruptly. It should also be noted that in the case of a two-phase system, the minority phase is defined by $\alpha < 0.5$. Therefore, if we take `alpha_res`=1 and `alpha_res_min`=0.5 with 2 phases, it means we are always in HEM for the minority phase because $\alpha < 0.5$ for the minority phase is always satisfied with 2 phases. However, if we only want to avoid velocity problems with the minority phase when it tends to disappear, we rather impose `alpha_res` $= 1.0 \times 10^{-6}$ (arbitrary pseudo-zero) and

`alpha_res_min` $= 5.0 \times 10^{-7}$ (slightly lower to smooth the transition to HEM), this is ultimately equivalent to having a single-phase system.

This operation is implemented through the object `correlation`. Equation 4.2 is an example of a correlation that can be used. Correlations related to the momentum equations are documented in 9.2.1 and correlation link to the energy equations are documented in 7.3.7.

4.3 Evanescence operator and energy equation

As reminder, the energy equations for the minority phase reads

$$\frac{\partial \alpha_{\text{mino}} \rho_{\text{mino}} h_{\text{mino}}}{\partial t} + \nabla \cdot (\alpha_{\text{mino}} \rho_{\text{mino}} h_{\text{mino}} \overrightarrow{v}_{\text{mino}}) = -p \left[\frac{\partial \alpha_{\text{mino}}}{\partial t} + \nabla \cdot (\alpha_{\text{mino}} \overrightarrow{v}_{\text{mino}}) \right] + \nabla \cdot (\alpha_{\text{mino}} \lambda_{\text{mino}} \nabla T_{\text{mino}}) + q_{\text{mino,p}} + q_{\text{mino,i}} \quad (4.3)$$

where $q_{\text{mino},i}$ the interfacial flux, $q_{\text{mino},p}$ the parietal flux (to evaluate the diffusive flux near wall boundary conditions). This parietal flux $q_{\text{mino},p}$ can be linked to the mass interface flux Γ_i thought a correlation. In this case, they are linked as follow

$$\Gamma_g = -\Gamma_l = \frac{q_l + q_g}{L_{\text{vap}}}, \quad (4.4)$$

with L_{vap} the vaporisation enthalpy. For the same reason as mentioned in the previous section, the problem can be ill-posed if the minority phase tends toward zero. For this reason, the evanescence operator can be applied to the energies equations. We obtain the same system than for the momentum one :

$$\begin{pmatrix} \mathcal{E}_{\text{pred}} \\ \mathcal{E}_{\text{mino}} \end{pmatrix} \Longrightarrow \begin{pmatrix} \mathcal{E}_{\text{pred}} + c\mathcal{E}_{\text{mino}} - c\left(T_{\text{mino}} = T_{\text{pred}} + T_{\text{corr}}\right) \\ (1-c)\,\mathcal{E}_{\text{mino}} + c\left(T_{\text{mino}} = T_{\text{pred}} + T_{\text{corr}}\right) \end{pmatrix} \quad (4.5)$$

Similarly to the momentum equation, it is necessary to define a relevant minority phase temperature in the case that α_{mino} is too low. In this case, it is important to define T_{mino}. Two scenarios may arise:

No phase change ($\Gamma_{\text{mino}} = 0$): in this case $q_{\text{mino},i} = h\left(T_{\text{mino}} - T_{\text{mino,sat}}\right)$ (with h the heat transfer coefficient) and there is no need of evanescence operator if h does not tend toward zero.

Considering phase change ($\Gamma_{\text{mino}} \neq 0$): in this case $q_{\text{mino}} = h\left(T_{\text{mino}} - T_{\text{mino,sat}}\right) + \Gamma_{\text{mino}} h_{\text{mino}}$. The term $\Gamma_{\text{mino}} h_{\text{mino}}$ is related to the default of energy due to mass transfer at the interface. It must be noted that if $\Gamma_{\text{mino}} > 0$ then $h_{\text{mino}} = h_{\text{mino,sat}}$. The underlying idea is that the phase is created from an interface at saturation temperature. This specificity is well documented in CATHARE documentation [**dummycitation**]. It aims at avoiding that vapour condensate considering too low latent heat and stay to high.

It has to be precised that T_{mino} can be imposed at T_{sat} if the composite medium is at saturation.

4.4 Flux limiter

The source term Γ_k has to respect two conditions:

1. flux consistency: $q_{ki} + q_{ki} = 0$
2. positive volume fractions for each phase: $\alpha_i(\boldsymbol{x}, t)\ \forall\ \boldsymbol{x}$ and t.

To achieve this goal, a first estimation of the interfacial heat flux is calculated from correlations for each phase:

$$\Gamma_k = f(q_i) \text{ such as } q_{\text{mino}} + q_{\text{pred}} = 0. \tag{4.6}$$

Then a flux limiter is calculated from the transport equation of the minority phase:

$$\Gamma_{\text{lim}} = \nabla \cdot (\alpha_{\text{mino}} \rho_{\text{mino}}) \overrightarrow{v}_{\text{mino}} - \frac{\partial \alpha_{\text{mino}} \rho_{\text{mino}}}{\partial t}. \tag{4.7}$$

If $\Gamma_k < \Gamma_{\text{lim}}$ both conditions are respected.

However, if $\Gamma_k \geq \Gamma_{\text{lim}}$, the resulting volume fraction of the minority phase would be negative. In this case, Γ_k is imposed at Γ_{lim} value and the flux consistence is imposed through the following relations

$$q_{\text{mino}} = h\left(T_{\text{mino}} - T_{\text{mino,sat}}\right) + \Gamma_{\text{lim}} h_{\text{mino}}, \tag{4.8}$$

$$q_{\text{pred}} = -q_{\text{mino}}. \tag{4.9}$$

These operations are coded in file `Source_Flux_interfacial_base`.

4.5 Dataset

An example of evanescence operator is

```
QDM_Multiphase
{
    evanescence { homogene { alpha_res 1.e-6 alpha_res_min 5.e-7 } }
    solveur_pression petsc cli_quiet { -pc_type hypre -pc_hypre_type
       boomeramg -ksp_type fgmres }
    convection { amont }
    diffusion { turbulente $diffusion { sigma 1 } }
    initial_conditions
    {
        ...
    }
    conditions_limites
    {
        ...
    }
}
```

Listing 4.1: data set definition

with `alpha_res` corresponding to α_{min} and `alpha_res_min` corresponding to $\alpha_{\text{res,min}}$.

4.6 Coding

4.6.1 Faces

- `Op_Evanescence_Homogene_Face_base`: management for faces (linked to momentum equation and so can impose the drift velocity)

The model is implemented in 2 files.

```
void Op_Evanescence_Homogene_Elem_base::readOn(Entree& is)
{
  Param param(que_suis_je());
  param.ajouter("alpha_res", &alpha_res_, Param::REQUIRED);
  param.ajouter("alpha_res_min", &alpha_res_min_);
  param.lire_avec_accolades_depuis(is);
  return is;
}
```

4.6.2 Elements

- Op_Evanescence_Homogene_Elem_base: management for elements (linked to mass and energy equations and can weakly imposed conditions for temperature and void fraction, based on fluxes)

```
void Op_Evanescence_Homogene_Face_base::readOn(Entree& is)
{
  Param param(que_suis_je());
  param.ajouter("alpha_res", &alpha_res_, Param::REQUIRED);
  param.ajouter("alpha_res_min", &alpha_res_min_);
  param.lire_avec_accolades_depuis(is);

  Pb_Multiphase& pbm = ref_cast(Pb_Multiphase, equation().probleme());
  if (pbm.has_correlation("Vitesse_relative") && !pbm.has_correlation("gravite
     ")) Process::exit(que_suis_je() + "␣:␣you␣must␣define␣a␣multiphase␣
     gravity␣field␣if␣you␣want␣a␣drift␣flux!!");
  if (pbm.has_correlation("Vitesse_relative") && ref_cast(
     Vitesse_relative_base, pbm.get_correlation("Vitesse_relative").valeur()).
     needs_vort()) pbm.creer_champ("vorticite");

  return is;
}
```

Default values : alpha_res_ = 0, alpha_res_min_ = 0.

4.6.3 Implementation

The model is called for each dimensioning and assembly of blocks or matrices (assembler_blocs, dimensionner_forblocs, dimensionner_matrice). It allows adding the equation of the residual phase to the majority phase, locally according to a global criterion alpha_res_. It is possible to impose an algebraic relationship for velocity and temperature (i.e., $v_l = v_g$ or $v_l = v_g + v_r$ for velocity, $T_l = T_{\text{sat}}$ or $T_l = T_g$ for temperature). Conditions on void fraction and temperature are imposed through flux corrections in the mass and energy equations. For example, for mass equation:

$$\Gamma_{k,\text{mino}} = \nabla \cdot (\alpha_k \rho_k) \overrightarrow{v_k} - \frac{\partial \alpha_k \rho_k}{\partial t} \tag{4.10}$$

In the case of the energy equation, if the flux is imposed by a correlation, then it is the pressure term that is corrected.

It is implemented as:

- For elements and faces `secmem(e, k)` adds corrected flux in right hand side of the equation of local main phase,
- For elements and faces `secmem(e, n)` corrects the flux in right hand side of the equation of local evanescent phase,
- For faces, if a drift velocity model is defined with `correlation_vd`, we impose the relative velocity between the local main phase and the local evanescent phase,

with `e` elements, `k` majority local phase, `n` evanescent local phase.

Chapter 5

Time schemes

This chapter aims to describe the different time schemes available for TrioCFD multiphase computations. The first section (5.1) is an introduction to the time stepping general methods before delving into the ICE/SETS method (section 5.2) and the future PAT method (section 5.3).

5.1 Preamble

Compressible codes fall into two categories based on their approach to solving the compressible Navier-Stokes equations. The first category comprises density-based methods, which are commonly employed for modeling flows dominated by acoustic phenomena or fluid compressibility effects. These methods aim to directly solve the compressible form of the Navier-Stokes equations, closing the system by computing thermodynamic pressure through an equation of state. However, density-based solvers face a drawback in low-Mach number flow modeling due to their explicit time integration approach. The main drawback of density-based solvers in the context of low-Mach number flow modeling is their explicit time integration of the governing equation system. When fluid flow is modeled with a fully explicit method, time-step sizes are restricted by the local Courant-Friedrichs-Lewy limit as:

$$\Delta_{t,acs} \leq \frac{k\Delta_x}{\mid u \mid + c} \, , \tag{5.1}$$

with k a constant depending of the numerical scheme, c the sound speed and u the velocity. Because of the significant difference between the entropy and the acoustic wave propagation speeds in low-Mach number flows, the explicit resolution of these phenomena is poorly optimized in low-Mach cases. Indeed, defining a convective acoustic time step as

$$\Delta_{t,conv} \leq \frac{k\Delta_x}{\mid u \mid} \, , \tag{5.2}$$

it leads to the relation

$$\Delta_{t,acs} = \Delta_{t,conv} \frac{\mathcal{M}_a}{1 + \mathcal{M}_a} \, , \tag{5.3}$$

with $\mathcal{M}_a$ the local Mach number. $\Delta_{t,acs} \ll \Delta_{t,conv}$ if $\mathcal{M}_a \ll 1$.

To overcome this difficulty pressure-based methods have been developed. In contrast to density-based methods, which solve the Navier-Stokes equation for density and compute the pressure thanks to an equation of state, the main idea of pressure-based methods is to use the pressure gradient as a constraint on the velocity in the momentum equations. The pressure gradient, calculated from a Helmholtz-type wave equation, has thus the great advantage of remaining finite in low-Mach number flows. Hence, such methods are commonly used to compute

compressible phenomena in low-Mach number flows. This section presents the two pressure-based methods available within CFMD called ICE (Implicit Continuity Equation) and initially developed by **harlow1968numerical**, **harlow1971numerical**. It is a pressure-based approach used in solving the Navier-Stokes equations for numerical simulation of fluid flows.

5.2 The ICE/SETS methods

All the temporal schemes available with the `Pb_multiphase` problem belong to the SETS class. ICE temporal scheme is a sub-class of SETS class.
The class ICE inherits from the SETS class. Both classes share the same parameters that are listed below.

```
int p_degen = -1
```

If `SETS.p_degen` is equal to -1, the numerical schem is corresponding to a semi-implicit scheme. If `SETS.p_degen` is equal to 1, the numerical scheme is corresponding to an incompressible solver. In this case, the pressure has a relative value: no BC can be impose on the pressure and no compressibility effects are considered.

5.2.1 Description of the algorithm

The keyword to activate the ICE method in a multiphase computation in TrioCFD is `solver ICE`. The algorithm is implemented in the `SETS.cpp` file as it is closely related to the SETS method. This algorithm carries two main functions that are used to solve the equations, namely `iterer_eqn` and `iterer_NS`. The first is use to iterate all equations in time and the second uses a Newton method to solve the momentum equation on velocity with pressure reduction. We briefly describe the algorithm of both function.

General algorithm for every equation

Algorithm 1: Algorithm iterer_eqn

Input: Unknowns: void fraction (always), temperature (ICE), standard convection-diffusion quantities (for example turbulence or IATE, ICE)
Output: Updated unknowns

Step 1: Positivity Enforcement;
foreach *unknown* **do**
  if *unknown is positive* **then**
    Update current unknown via increment;
  else
    Impose 0 (via unknown_positivation);
  end
end

Step 2: Solve Navier-Stokes Equations;
if *Navier-Stokes equations are in the system* **then**
  Call iterer_NS to solve velocity and pressure;
end

Step 3: Solve Thermal Equation;
Solve multiphase or conduction thermal equation explicitly if ICE is used;

Step 4: Check Positivity of Temperature;
Check if temperature is positive;

Step 5: Update Time Step and Unknowns;
Update time step and the unknowns;

Specific algorithm for the momentum and pressure equations

Algorithm 2: Algorithm for iterer_NS

Input: Initial values
Output: Updated values

Step 1: Initialization;
Save current values in the past;

Step 2: Solve Velocity;
if *SETS is activated* **then**
  Solve velocity with explicit pressure and store it;
else
  Save past value of velocity;
end

Step 3: PolyMAC Correction;
if *PolyMAC is active* **then**
  Correct velocity at elements using mass;
end

Step 4: Newton Method;
Step 4.1: Fill energy equation to obtain wall transfers;
Step 4.2: Fill other equations;
Step 4.3: Assemble pressure system;
if *pressure reduction* **then**
  Step 4.4.1: Eliminate unknowns for A_p and b_p for each phase by substitution;
  Step 4.4.2: Assemble system A_p and B_p;
  Step 4.4.3: Solve pressure;
  Step 4.4.4: Solve other increments;
else
  Step 4.5: Solve directly by `mat_semi_impl`;
end
Step 4.6: If PolyMAC, apply mass correction to correct velocity at elements;

Step 5: Convergence Check;
Convergence criteria (see Table 5.1 for default criteria);

Step 6: Update;
Update values;

Field	Notation	Criteria
Void fraction	α	0.01
Temperature	T	0.1
Velocity	v	0.01
Pressure	p	100
Turbulent kinetic energy	k	0.01
Wake induced turbulent energy	k_{WIT}	0.01
Rate of dissipation	ω	0.01
Dissipation time scale	τ	0.01
Interfacial area concentration	ai	100

Table 5.1: Default convergence criteria

5.2.2 Principle of the time discretization

The following steps are also presented in **GerschenfeldPolyMAC2022**.
We can write the time discretized mass, momentum and energy conservation equations along with the continuity axiom:

$$(\mathcal{C}) \qquad \sum_k \alpha_k^{n+1} = 1 \tag{5.4}$$

$$(\mathcal{M}_k) \qquad \frac{\alpha_k^{n+1}\rho_k^{n+1} - \alpha_k^n\rho_k^n}{\Delta t} + \nabla\cdot(\alpha_k^n\rho_k^n\overrightarrow{u_k}^{n+1}) = \Gamma_k^{n+1} \tag{5.5}$$

$$\begin{aligned}(\mathcal{Q}_k) \qquad & \alpha_k^n\rho_k^n\frac{\overrightarrow{u_k}^{n+1} - \overrightarrow{u_k}^n}{\Delta t} + \underline{\nabla}\cdot(\alpha_k^n\rho_k^n\overrightarrow{u_k}^n\otimes\overrightarrow{u_k}^n) - \overrightarrow{u_k}^n\underline{\nabla}\cdot(\alpha_k^n\rho_k^n\overrightarrow{u_k}^n) = \\ & \qquad -\alpha_k^n\underline{\nabla}P^{n+1} + \underline{\nabla}\cdot\left(\alpha_k^n\mu_k^n\underline{\underline{\nabla}}\overrightarrow{u_k}^n\right) \\ & \qquad -\underline{\nabla}\cdot(\alpha_k^n\rho_k^n\overline{u_i'u_j'}^n) + \overrightarrow{F}_{ki}^{n+1} + \overrightarrow{F}_k^{n+1}\end{aligned} \tag{5.6}$$

$$\begin{aligned}(\mathcal{E}_k) \qquad & \frac{\alpha_k^{n+1}\rho_k^{n+1}e_k^{n+1} - \alpha_k^n\rho_k^ne_k^n}{\Delta t} + \nabla\cdot(\alpha_k^n\rho_k^ne_k^n\overrightarrow{u_k}^{n+1}) = \\ & \qquad \nabla\cdot\left(\alpha_k^n\lambda_k\underline{\nabla}T_k^n - \alpha_k^n\rho_k^n\overline{u_i'e_k'}^n\right) + q_{ki}^n + q_{kp}^n \\ & \qquad -P^{n+1}\left(\frac{\alpha_k^{n+1}-\alpha_k^n}{\Delta t} + \nabla\cdot(\alpha_k^n\overrightarrow{u_k}^{n+1})\right)\end{aligned} \tag{5.7}$$

Once this system is written, one can express a matrix system in terms of increments on the unknowns. For example, for the temporal term in the mass equation:

$$\frac{1}{\Delta t}\left(\alpha_k^{n+1}\rho_k^{n+1} - \alpha_k^n\rho_k^n\right) = \frac{1}{\Delta t}\left(\rho_k^n(\alpha_k^{n+1}-\alpha_k^n) + \alpha_k^{n+1}\left(\rho_k^{n+1}-\rho_k^n\right) + \alpha_k^n\rho_k^n - \alpha_k^n\rho_k^n\right) \tag{5.8}$$

Let's remind that the unknowns are (α, T, P, $\overrightarrow{u}$). Then, the void fraction increment is $\delta\alpha_k = \alpha_k^{n+1} - \alpha_k^n$. For the density, one must add its dependencies to the unknowns. In this case, the density field $\rho(T,P)$ depends only on the temperature and the pressure fields. Also its differential is $d\rho = \frac{\partial\rho}{\partial T}\delta T + \frac{\partial\rho}{\partial P}\delta P$. Thus we obtain the following equation for the increments:

$$\underbrace{\frac{\rho_k^n}{\Delta t}}_{\frac{\partial\mathcal{M}_k}{\partial\alpha_k}^{(n)}}\delta\alpha_k + \underbrace{\frac{\alpha_k^{n+1}}{\Delta t}\left(\frac{d\rho_k}{dT_k}\right)^n}_{\frac{\partial\mathcal{M}_k}{\partial T_k}^{(n+1)}}\delta T_k + \underbrace{\frac{\alpha_k^{n+1}}{\Delta t}\left(\frac{d\rho_k}{dP}\right)^n}_{\frac{\partial\mathcal{M}_k}{\partial P}^{(n+1)}}\delta P \tag{5.9}$$

We can then do this for all terms and construct the following system for the increments of unknowns:

$$\begin{pmatrix} \boxed{\frac{\partial\mathcal{M}_k}{\partial\alpha_k}^{(n)}} & \boxed{\frac{\partial\mathcal{M}_k}{\partial T_k}^{(n+1)}} & \frac{\partial\mathcal{M}_k}{\partial\overrightarrow{u_k}}^{(n)} & \frac{\partial\mathcal{M}_k}{\partial P}^{(n+1)} \\ \boxed{\frac{\partial\mathcal{E}_k}{\partial\alpha_k}^{(n+1)}} & \boxed{\frac{\partial\mathcal{E}_k}{\partial T_k}^{(n+1)}} & \frac{\partial\mathcal{E}_k}{\partial\overrightarrow{u_k}}^{(n+1)} & \frac{\partial\mathcal{E}_k}{\partial P}^{(n)} \\ 0 & 0 & \boxed{\frac{\partial\mathcal{Q}_k}{\partial\overrightarrow{u_k}}^{(n)}} & \frac{\partial\mathcal{Q}_k}{\partial P}^{(n)} \end{pmatrix} \begin{pmatrix} \delta\alpha_k^{(n+1)} = \alpha_k^{n+1} - \alpha_k^n \\ \delta T_k^{(n+1)} \\ \delta\overrightarrow{u_k}^{(n+1)} \\ \delta P^{(n+1)} \end{pmatrix} = \begin{pmatrix} \delta\mathcal{M}_k^{(n)} \\ \delta\mathcal{E}_k^{(n)} \\ \delta\mathcal{Q}_k^{(n)} \end{pmatrix} \tag{5.10}$$

The terms framed in blue ( ) are block diagonal because they only depend on unknowns coming from the cell in which they are computed.
In a Newton algorithm, one can take the first line of the system, replace unknows then adds

for any phases and substracts the void fraction dependency with $\sum_k \delta\alpha_k^{n+1} = 0$ and then get $\delta P^{(n+1)}$. This step is called pressure reduction and is described after. The last line allows the direct prediction to get $\delta\overrightarrow{u_k}^{(n+1)}$, so that the system becomes:

$$\begin{pmatrix} \frac{\partial \mathcal{M}_k}{\partial \alpha_k}^{(n+1)} & \frac{\partial \mathcal{M}_k}{\partial T_k}^{(n)} \\ \frac{\partial \mathcal{E}_k}{\partial \alpha_k}^{(n+1)} & \frac{\partial \mathcal{E}_k}{\partial T_k}^{(n+1)} \end{pmatrix} \begin{pmatrix} \delta\alpha_k^{(n+1)} \\ \delta T_k^{(n+1)} \end{pmatrix} = \begin{pmatrix} \delta\mathcal{M}_k^{(n)} \\ \delta\mathcal{E}_k^{(n)} \end{pmatrix} + \begin{pmatrix} \frac{\partial \mathcal{M}_k}{\partial P_k}^{(n)} \\ \frac{\partial \mathcal{E}_k}{\partial P_k}^{(n)} \end{pmatrix} \delta P^{(n+1)} + \begin{pmatrix} \frac{\partial \mathcal{M}_k}{\partial \overrightarrow{u_k}}^{(n)} \\ \frac{\partial \mathcal{E}_k}{\partial \overrightarrow{u_k}}^{(n+1)} \end{pmatrix} \delta\overrightarrow{u_k}^{(n+1)} \tag{5.11}$$

Then the temperature can be solved. The convergence criteria must be then verified.

5.2.3 Principle of the pressure reduction

To avoid heavy notations, the subscript k which refers to each phase are dropped. The previous system can be written as:

$$M_\alpha \delta\alpha + M_T \delta T + M_u \delta u + M_p \delta P = f_M \tag{5.12}$$
$$E_\alpha \delta\alpha + E_T \delta T + E_u \delta u + E_p \delta P = f_E \tag{5.13}$$
$$Q_u \delta u + Q_p \delta P = f_Q \tag{5.14}$$

From equation (5.14), we can express the velocity increment as

$$\delta u = Q_u^{-1} f_Q - Q_u^{-1} Q_p \delta P. \tag{5.15}$$

Injecting it into the previous system, we obtain the following system

$$M_\alpha \delta\alpha + M_T \delta T + M_u \big(Q_u^{-1} f_Q - Q_u^{-1} Q_p \delta P\big) + M_p \delta P = f_M \tag{5.16}$$
$$E_\alpha \delta\alpha + E_T \delta T + E_u \big(Q_u^{-1} f_Q - Q_u^{-1} Q_p \delta P\big) + E_p \delta P = f_E \tag{5.17}$$

One can then replace the temperature from the energy equation (5.17) in the mass equation (5.16):

$$\begin{aligned} M_\alpha \delta\alpha + M_T E_T^{-1} \big(-E_\alpha \delta\alpha - E_u \big(Q_u^{-1} f_Q - Q_u^{-1} Q_p \delta P\big) - E_p \delta P + f_E\big) \\ - M_u \big(Q_u^{-1} f_Q - Q_u^{-1} Q_p \delta P\big) + M_p \delta P = f_M \end{aligned} \tag{5.18}$$

We then group together terms that multiply $\delta\alpha$ and δP. We obtain:

$$\begin{aligned} \underbrace{\big(M_\alpha - M_T E_T^{-1} E_\alpha\big)}_{K_\alpha} \delta\alpha + \underbrace{\big(M_T E_T^{-1} (E_u Q_u^{-1} Q_p - E_p) - M_u Q_u^{-1} Q_p + M_p\big)}_{K_p} \delta P = \\ \underbrace{f_M - M_u Q_u^{-1} f_Q - M_T E_T^{-1} (f_E - E_u Q_u^{-1} f_Q)}_{K_f} \end{aligned} \tag{5.19}$$

Remark: if we neglect the temperature, we have a simpler system:

$$\underbrace{M_\alpha}_{K_\alpha} \delta\alpha + \underbrace{\big(M_p - M_u Q_u^{-1} Q_p\big)}_{K_p} \delta P = \underbrace{f_M - M_u Q_u^{-1} f_Q}_{K_f}. \tag{5.20}$$

By inverting matrix K_α, equations (5.19) and (5.20) can be rewritten as

$$\delta\alpha + K_\alpha^{-1} K_p \delta P = K_\alpha^{-1} K_f. \tag{5.21}$$

With the axiom of continuity, i.e. by summing equation (5.21) on all phases, we obtain the final linear system of the form $Ax = B$:

$$\underbrace{\sum_k (K_\alpha^{-1} K_p)_k}_{A_p} \delta P = \underbrace{\sum_k (K_\alpha^{-1} K_f)_k}_{B_p} \tag{5.22}$$

5.2.4 SETS

The Stability-Enhancing Two-Step (SETS) [**MAHAFFY1982329**] time resolution method offers a mean to overcome the material Courant limit, thus mitigating certain stability issues associated with time-step size. Nonetheless, this approach does introduce a higher degree of numerical diffusion. Its interest lies notably in its suitability for simulating slow transients, allowing the use of way larger time steps compared to ICE. However, in multiphase simulations the time step is still limited by the convective time step because of α. At each time step, this method yields an intermediate time velocity, temperature and other convected quantities (turbulence, IATE) field. This is particularly advantageous because achieving a solution with mass and energy convection becomes smoother when the velocity is already pressure-balanced. This algorithm is activated through the data set with the following keyword.

```
solveur SETS
```

The main idea is to remind that we can use an intermediate time stepping, written $\widetilde{\cdot}$, to get closer to the wanted solution:

$$\delta P = P^{n+1} - P^n = \underbrace{P^{n+1} - \widetilde{P}^{n+1}}_{\delta P^{n+1}} + \underbrace{\widetilde{P}^{n+1} - P^n}_{\delta \widetilde{P}} \tag{5.23}$$

Thus we obtain:

$$\delta T = T^{n+1} - T^n = \underbrace{T^{n+1} - \widetilde{T}^{n+1}}_{\delta T^{n+1}} + \underbrace{\widetilde{T}^{n+1} - T^n}_{\delta \widetilde{T}} \tag{5.24}$$

$$\delta u = u^{n+1} - u^n = \underbrace{u^{n+1} - \widetilde{u}^{n+1}}_{\delta u^{n+1}} + \underbrace{\widetilde{u}^{n+1} - u^n}_{\delta \widetilde{u}} \tag{5.25}$$

It means that if one choose wisely the intermediate pressure, we can have a first prediction of velocity and temperature to ease the computation of convection in the Newton algorithm. In the case of SETS, using a first prediction with explicit pressure $\delta \widetilde{P} = 0$ allows to predict a pressure balanced intermediate time step for the velocity and temperature.

In the alogorithm, if `facsec_diffusion_for_sets` is bigger than 0, then the facsec is chosen to impose on the diffusion time step in SETS while the total time step stays smaller than the convection time step.

In order to overcome problems of implicit convection for α (due to the constraint $\sum \alpha_k = 1$), one solution that could be implemented is to predict the minority phases and to impose for the major phase $\alpha^{major} = 1 - \sum \alpha_k^{minor}$.

5.2.5 Specific case of ICE modelization within single phase framework

Initially, ICE method solves the equations for momentum and pressure simultaneously by using an implicit formulation of the continuity equation. ICE method was initially developed for single-phase flows, and the pressure reduction method relies on MAC numerical scheme [**harlow1965numerical**]. A easy way to be more familiar with ICE method is to visualize the equations in the simple case of a single-phase flow. In this specific condition, ICE method relies on the resolution of a Poisson equation based on the implicit continuity equation. The pressure field at the end of the time step, P^{n+1} is calculated by

$$\frac{\delta P}{\Delta_t^2 c^2} - \Delta P = \nabla \cdot \nabla \left(\rho \mathbf{u}\mathbf{u} - \tau \right) + \frac{\nabla \cdot \rho \mathbf{u}}{\Delta_t} \, . \tag{5.26}$$

From the sound speed definition the density can be updated

$$\delta \rho = \frac{\delta P}{c^2} \, . \tag{5.27}$$

The equation of state is involved in the calculation of the sound speed c. Once P^{n+1} and ρ^{n+1} are known, the velocity can be estimated solving the transport equation of the momentum. Finally, at the end of the time step, the temperature can be calculated from P^{n+1}, ρ^{n+1} and $\mathbf{u}^{n+1}$. The ICE solver is robust but must respect a strong assumption $CFL \leq 1$. Note that in this method, the diffusion of the temperature is always explicit.

5.3 PAT (incoming)

The so-called PAT for (P, α, T) resolution is a NeptuneCFD-looklike time scheme. It is available as an alternative for PolyVEF spatial scheme, which is not optimal with ICE/SETS method. This quasi-implicit method features a velocity prediction step followed by the calculation of pressure P, volume fraction α, and temperature T. It uses maximal implicitness while avoiding saddle points anomalies. This time scheme is not yet validated, we advise not to use it.

Chapter 6

Spatial Scheme

This chapter gives an overview of the available spatial schemes for multiphase flow in TRUST/TrioCFD. First, common notations are given at Section 6.1. The different schemes of the PolyMAC family are described in Section 6.2. Few elements are given on the PolyVEF scheme in Section 6.3 and on the VDF scheme in Section 6.4. Finally, details about the boundary conditions are given in Section 6.5.

6.1 Notation

First, we consider a certain polyhedral mesh M. In a three-dimensional framework, M is composed of different elements: cells c, faces f, edges e and vertexes v. A cell is a peculiar volume subset of the grid. The set of those subsets span the whole grid. A face is defined as the intersection of two cells or the intersection of a cell and a boundary of the domain. An edge is therefore an intersection of faces (and potentially a boundary) and a vertex is the intersection of edges (and potentially a boundary). The total number of each element el on the mesh is fixed and will be referred as $\#el$. The set of faces of a peculiar cell c will be denoted F_c. In the same fashion, the set of edges of a peculiar face f will be noted as E_f and eventually, the two vertexes of an edge e will be denoted V_e.

In the sequel, the measure of an unknown x at an element of the mesh el will be denoted:

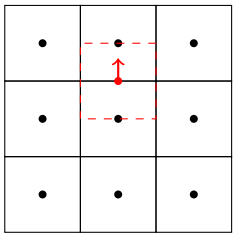
$$[x]_{el} = \frac{1}{|el|} \int_{el} x \, \mathrm{d}\,(el) \tag{6.1}$$

where $|\cdot|$ will be a global measure operator of a mesh element. For example, $|c|$ refers to the volume of the cell c, $|f|$ to the surface of the face f and $|e|$ to the length of the edge e.

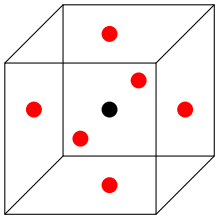
Available spatial discretisations in Pb_multiphase can be seen as extensions of the Marker And Cell (MAC) method [**harlow1965numerical**] to meshes more complex than Cartesian grids. All those schemes are also available in TRUST [**trustonline**] for a single phase flow. They are called respectively: VDF, PolyMAC, PolyMAC_P0, PolyMAC_P0-P1NC. On the other hand, the PolyVEF_P0 scheme can be seen as an extension of the Crouzet-Raviart Finite Element scheme.

Compatibility of the available spacial discretisation is summarized in Table 6.1. Please note that only PolyMAC_P0 and VDF discretisations are currently[1] validated on multiple test cases. We recommend to use VDF whenever it is possible as computation time is drastically lower than PolyMAC. Figure 6.1 succinctly presents the locations of the unknowns for the different discretisations available in Pb_multiphase.

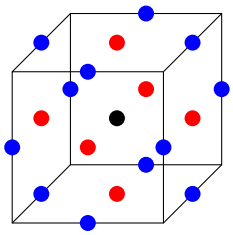
[1] May 2024



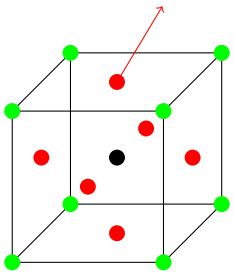
(a) VDF MAC.
Legend:
• (α, T, P) ;
• $(\overrightarrow{u} \cdot \overrightarrow{n})$, where $\overrightarrow{n}$ is the edge normal vector.



(b) PolyMAC_P0.
Legend:
• $(\alpha, T, P, \overrightarrow{u})$;
• $(\overrightarrow{u} \cdot \overrightarrow{n})$, where $\overrightarrow{n}$ is the face normal vector.



(c) PolyMAC_P0-P1NC.
Legend:
• $(\alpha, T, P, \overrightarrow{u})$;
• $(\overrightarrow{u} \cdot \overrightarrow{n}, T_{aux}, P_{aux})$, where $\overrightarrow{n}$ is the face normal vector ;
• $\overrightarrow{curl}(\overrightarrow{u}).\overrightarrow{t}$, where $\overrightarrow{t}$ is the edge tangent vector.



(d) PolyVEF_P0P1.
Legend:
• (α, T, P) ;
• $\overrightarrow{u}$;
• (α, T, P).

Figure 6.1: Scheme of different storing locations for each discretisation available in Pb_multiphase.

Scheme	PolyMAC	PolyMAC_P0	PolyMAC_P0-P1NC	PolyVEF_P0P1	VDF
TRUST	✓	✓	✓	✓	✓
Pb_Multiphase	✗	✓	✓	✓	✓
Source terms	✗	✓	✗	✓	✓
Validation	✗	✓	✗	✗	✓

Table 6.1: Availability of spatial numerical schemes in Trio_CMFD.

6.2 The PolyMAC family

As stated above, the PolyMAC discretisation's family aims to extend the MAC scheme to complex polyhedral and deformed grids. Indeed, staggered discretisation tends to decrease the appearance of potential spurious mode in low Mach number regime. PolyMAC are based on Finite Volumes methods. As such, unknown are discretised and stored on control volumes. Moreover, for each PolyMAC method, a dual mesh (more or less complex depending on the PolyMAC version) is constructed and operators are defined in order to interpolate unknowns from a control volume to another (face to edge, vertex to cell, ...). Those scheme are also available in TRUST code for solving the incompressible Navier-Stokes system. In practice, the discretisation methods are coded in TRUST in a modular fashion, so that they can be used for various models.

6.2.1 PolyMAC

The first developed PolyMAC version, simply called PolyMAC in TRUST/TrioCFD, is based on a *mimetic* method [**lipnikov2014mimetic**]. The core idea of those methods is to conserve exact property at the discrete level. To be more precise, PolyMAC lays in the *Compact Discrete Operator* framework, such as the one presented by **bonelle2014**, **milani2020**. A dual mesh has to be build and unknown and equations are discretised on this dual mesh. In this framework, local quantities are discretised according to their physical properties. This leads to exact discrete operator such as the gradient, curl and divergence. Some other approximated interpolation operators, called Hodge operators, must also be defined.

However, up to know, this version is not available in Pb_multiphase as the discretisation of the two-phase flow model leads to a non-triangular matrix (because of the use of the vorticity) and two saddle point problems that impair greatly the stability and performance of the computation in the multiphase flow framework.

6.2.2 PolyMAC_P0

Unlike the first PolyMAC discretisation, PolyMAC_P0 is not based on a CDO approach and the discretisation does not require explicitly a dual mesh. Moreover, in this case, equations are discretised at the control volume of their principal unknown: $\alpha_k\rho_k$ for $\mathcal{M}_k$, u_k for $\mathcal{Q}_k$ and T_k for $\mathcal{E}_k$.

Discretisation of the unknown

For $k \in \{1, 2\}$:

- the velocity u_k is discretised by the average of its outer-pointing normal component at the

face:

$$[u_k]_f = \frac{1}{|f|}\int_f \overrightarrow{u_k}\cdot\overrightarrow{n_f}\,dS, \tag{6.2}$$

with $\overrightarrow{n_f}$ a unit outer-pointing normal of the face f.

- The pressure P is discretised over the cell:

$$[P]_c = \frac{1}{|c|}\int_c p dV, \tag{6.3}$$

- the temperature T_k is discretised over the cell:

$$[T_k]_c = \frac{1}{|c|}\int_c T_k dV, \tag{6.4}$$

- the partial fraction α_k is discretised over the cell:

$$[\alpha_k]_c = \frac{1}{|c|}\int_c \alpha_k dV, \tag{6.5}$$

Discretisation of gradient terms

For PolyMAC_P0, the reconstruction of the gradient at the face is conducted through a Multi Point Flux Approximation (MPFA) method [**droniou2018gradient**]. Those methods gives approximations of the gradient at the face, called G^{MPFA} (see sub-section 6.2.3 for its definition). However, gradient of a scalar field $\mathcal{F}_s$ at the cell center will be needed in the following. Its value is thus interpolate by using the following operator taken from **bonelle2014**:

$$\left[\overrightarrow{\nabla F_s}\right]_c = \frac{1}{|c|}\sum_{f\in F_c}|f|[u]_f|\nabla\mathcal{F}_s|_f\,\overrightarrow{x}_{c\to f}, \tag{6.6}$$

where $\overrightarrow{x}_{c\to f}$ is the vector linking the gravity center of the cell c to the gravity center of the face f.

Discretisation of the scalar equations

First, we want to take care of the scalar equations. Those equations, *i.e.* for $k\in\{1,2\}$ $(\mathcal{M}_k)$ and $(\mathcal{E}_k)$, are discretised at the cell. Considering a scalar field $\mathcal{F}_s$, then, owing to the Stokes theorem, we have:

$$|c|[\nabla\cdot(F_s\overrightarrow{u})]_c = \sum_{f\in F_c}|f|[\mathcal{F}_s]_f[u]_f \tag{6.7}$$

The scalar fields at the face are reconstructed from their values at the cell by using an upwind reconstruction. Equation (6.7) is used to discretised every divergence terms in the scalar equations.

Discretisation of the momentum equation

The momentum equation $(\mathcal{Q}_k)$ is discretised at the faces. However, as it is challenging to construct a discretisation of convective terms $\nabla\cdot(\alpha_k\rho_k\overrightarrow{u}_k\otimes\overrightarrow{u}_k)$ and diffusives terms $[\nabla\cdot(\alpha_k\mu_k(\overrightarrow{\nabla}\overrightarrow{u}_k)+(\overrightarrow{\nabla}\overrightarrow{u}_k)^{\intercal})]$ with only the velocity projected at the face, the following method is used:

- For convective terms:

1. Approximate the value of the velocity at the cells in the same manner as the gradient:

$$[\overrightarrow{u}]_c = \frac{1}{|c|} \sum_{f \in F_c} |f| [u]_f \overrightarrow{x}_{c \to f}. \tag{6.8}$$

2. Discretise the convective terms at the cell centers:

$$\begin{aligned}
[\nabla \cdot (\alpha_k \rho_k \overrightarrow{u}_k \otimes \overrightarrow{u}_k)]_c &= [\alpha_k \rho_k]_c [\nabla \cdot (\overrightarrow{u}_k \otimes \overrightarrow{u}_k)]_c && (6.9)\\
&= \frac{[\alpha_k \rho_k]_c}{|c|} \sum_{f \in F_c} |f| [\overrightarrow{u}_k \otimes \overrightarrow{u}_k]_f && (6.10)\\
&\simeq \frac{[\alpha_k \rho_k]_c}{|c|} \sum_{f \in F_c} |f| [u]_f \left(\beta \left(\gamma \overrightarrow{u_c} + (1-\gamma)\, \overrightarrow{u_{c'}}\right)\right.\\
&\quad \left. + (1-\beta) \left(\frac{\overrightarrow{u_c} + \overrightarrow{u_{c'}}}{2}\right)\right), && (6.11)
\end{aligned}$$

with $\beta \in [0,1]$ and $\gamma \in \{0,1\}$ such that $\gamma = 1$ if $[u_f] \geq 0$ and 0 otherwise. When taking $\beta = 0$, the staggered scheme is retrieved, and when taking $\beta = 1$, we retrieved the upwind scheme (used by default in PolyMAC_P0).

3. Interpolate convective terms to the face:

$$[\nabla \cdot (\alpha_k \rho_k \overrightarrow{u}_k \otimes \overrightarrow{u}_k)]_f = \lambda_{c,f} [\nabla \cdot (\alpha_k \rho_k \overrightarrow{u}_k \otimes \overrightarrow{u}_k)]_c + \lambda_{c',f} [\nabla \cdot (\alpha_k \rho_k \overrightarrow{u}_k \otimes \overrightarrow{u}_k)]_{c'} \tag{6.12}$$

with the penalty coefficient $\lambda_{c,f} = \frac{|\overrightarrow{x}_{c' \to f}|}{|\overrightarrow{x}_{c' \to f}| + |\overrightarrow{x}_{c \to f}|}$, with c' the neighbouring cell of c sharing the face f.

- For diffusive terms:

1. Approximate the velocity u_k at the cell using a second order method:

$$[\overrightarrow{u}]_c = \frac{1}{|c|} \sum_{f \in F_c} (\overrightarrow{\alpha}_f + \overrightarrow{\beta}_f)[u]_f, \tag{6.13}$$

where $\overrightarrow{\alpha}_f = |f| \overrightarrow{x}_{c \to f}$ and $\overrightarrow{\beta}$ is a vector which component depends on $\overrightarrow{x}_{c' \to f'}$. Vector $\overrightarrow{x}_{c' \to f'}$ links the gravity center of the neighbour cells c' of c to the gravity center of the face f' of c' which have a common vertex with c. Vector β_f is computed through the minimisation of a function.

2. Compute:

$$\begin{aligned}
[\nabla \cdot (\alpha_k \mu_k (\overrightarrow{\nabla} \overrightarrow{u}_k) + (\overrightarrow{\nabla} \overrightarrow{u}_k)^{\intercal})]_c = \sum_{\tilde{f}} |\tilde{f}| [\alpha_k]_c [\rho_k]_c (G^{\mathrm{MPFA}}([u]_c)\\
+ \left(G^{\mathrm{MPFA}}([u]_c)\right))^{\intercal} \cdot \overrightarrow{n}_f.
\end{aligned} \tag{6.14}$$

3. Interpolate diffusive terms to the face:

$$\begin{aligned}
[\nabla \cdot (\alpha_k \mu_k (\overrightarrow{\nabla} \overrightarrow{u}_k) + (\overrightarrow{\nabla} \overrightarrow{u}_k)^{\intercal})]_f = &\lambda_{c,f} [\nabla \cdot (\alpha_k \mu_k (\overrightarrow{\nabla} \overrightarrow{u}_k) + (\overrightarrow{\nabla} \overrightarrow{u}_k)^{\intercal})]_c\\
&+ \lambda_{c',f} [\nabla \cdot (\alpha_k \mu_k (\overrightarrow{\nabla} \overrightarrow{u}_k) + (\overrightarrow{\nabla} \overrightarrow{u}_k)^{\intercal})]_{c'}
\end{aligned} \tag{6.15}$$

The other terms of $(\mathcal{Q}_k)$ are simply constructed by using the Stokes theorem, the velocity at the face and MPFA methods.

Remark: *Point 3 for reconstruction at the face of convective and diffusive terms can lead to a stability issue on very deformed grids.*

6.2.3 MPFA methods: gradient approximations

In TRUST code, three version of MPFA for gradient reconstruction exist: MPFA-O [**AgMa08**], MPFA-O(h) [**edwards1998finite**] and MPFA-symmetric [**Lepot05**]. In practice, the choice of the MPFA method follow this algorithm:

Algorithm 3: Selection of the MPFA method

Input: A cell c
Output: MPFA_scheme

Step 1: Test of the stability condition (6.18) **on** c **using a MPFA-O method**;
if (6.18)$==TRUE$ **then**
  MPFA_scheme=MPFA-O;
  break;
Step 2: Test of the stability condition (6.18) **on** c **using a MPFA-O(h) method**;
if (6.18)$==TRUE$ **then**
  MPFA_scheme=MPFA-O(h);
  break;
MPFA_scheme=MPFA-symmetric;
break;

The first one is more precise but less stable than the second one which is respectively more precise and less stable than the third one. The MPFA-symmetric method can always be computed but can lack in consistency on very deformed mesh. MPFA-O [**AgMa08**] and MPFA-O(h) methods have a stability condition which ensures that they stay coercive, see (6.18). A detailed introduction to MPFA methods can be found in [**droniou2014finite**]. Those methods are considered as gradient approximation methods, see [**droniou2018gradient**].

In the sequel, we will present the MPFA-O method.

In this case, in order to compute the gradient approximation, a dual mesh is constructed. It can be seen on Figure 6.2 in the case of a triangular mesh. The primal mesh vertexes are represented by • in Figure 6.2. The procedure to build the dual mesh is:

- Link each cell's (c) gravity center (• in Figure 6.2) to the gravity center of each cell's face $f \subset c$ (• in Figure 6.2). Doing so, the face of the mesh are cut into two sub-faces called $\tilde{f}_1$ and $\tilde{f}_2$. Each cell can then be subdivided into N_i quadrilaterals, called $(S_{c,i})_{i\in\{1,\dots,N_i\}}$ in Figure 6.2.
- Introduce inside each sub-face $\tilde{f} \subset f$, an auxiliary quantity (• in Figure 6.2). For MPFA-symmetric method, those auxiliary quantities are set at one third and two third of the face f.

On S_1 in Figure 6.2 for example, the gradient of a potential p, $G_{S_{c,i}}([p]_c)$ is computed as:

$$G_{S_{c,i}}([p]_c) = \frac{1}{|S_{c,i}|}((p_{S_{c,1},1} - p_c)\overrightarrow{n_1} + (p_{S_{c,1},2} - p_c)\overrightarrow{n_2}), \tag{6.16}$$

where $\overrightarrow{n_1}$ and $\overrightarrow{n_2}$ are the outward unit normal vectors of the respective sub-faces $\tilde{f} \subset f$ where the auxiliary elements $p_{S_{c,1}}$ and $p_{S_{c,2}}$ are located. Thus, G^{MPFA} writes:

$$G^{\text{MPFA}} : [p]_c \mapsto G^{\text{MPFA}}([p]_c)\ , \quad \forall c \in C\ , \quad i \in S_c\ : \quad G^{\text{MPFA}}_{|S_{c,i}} = G_{S_{c,i}}([p]_c). \tag{6.17}$$

A core assumption of the MPFA method is to suppose that $G^{\text{MPFA}}([p]_c)$ is constant on each $S_{c,i}$.

When enforcing the continuity across the sub-faces that are linked by a vertex of the primal mesh, auxiliary variables can be substitute by cells unknowns.

By construction, this approach guarantees local flux imbalance.

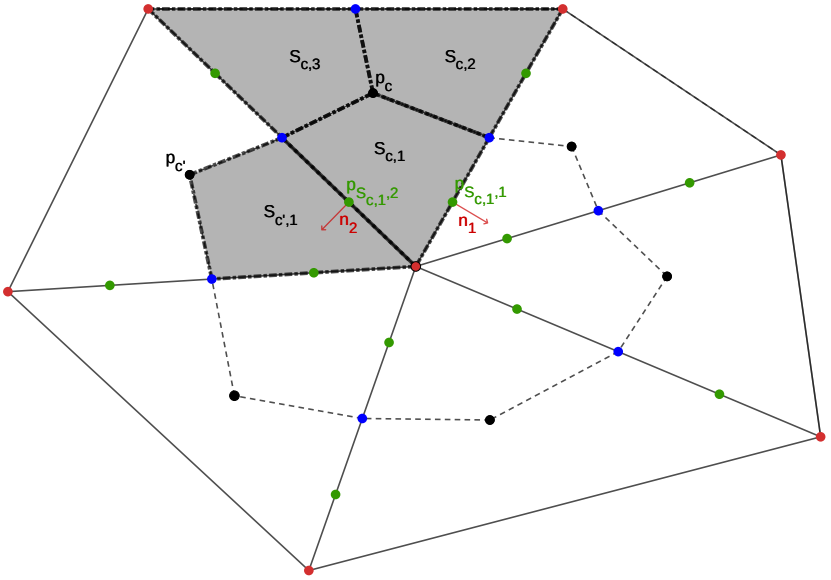


Figure 6.2: Scheme of the MPFA method for a triangular mesh [**Bacq**].

For a face f of vertex i and j, shared by cells c and c', the **stability condition** of the MPFA method reads:

$$\forall (k,l) \in \{i,j\},\ \forall (\tilde{c},\hat{c}) \in \{c,c'\},\ G_{S_{\tilde{c},k}} \cdot G_{S_{\hat{c},l}} \geq 0 \tag{6.18}$$

Some remarks:

- *Due to large stencil of the MPFA scheme induced by the elimination of the auxiliary unknows, PolyMAC_P0 tends to be more costly when using tetrahedron grids rather than hexahedron grids.*
- *On the boundary of the domain, only one auxiliary quantity is constructed by face when no Dirichlet or Neumann are imposed.*

6.2.4 PolyMAC_P0-P1NC

The PolyMAC_P0-P1NC discretisation aims at reducing some stability issues of PolyMAC_P0. It uses an Hybrid Finite Volume method described in [**eymard2007**] in the case of a diffusion problem. Moreover, as in the first PolyMAC discretisation, the vorticity $\omega = \nabla \wedge u$ is introduced. We remind that no source terms are implemented and no validation have been made for this scheme. We do not recommend to use this scheme for most cases.

6.3 PolyVEF_P0

The PolyVEF_P0 discretisation method is different from the PolyMAC family. Indeed, it is an extension to arbitrary polyhedron of the Crouzeix-Raviart Finite Element Method named VEF in `TRUST/TrioCFD`, hence the name PolyVEF. In this case, scalar quantities are stored at the cell and at the vertexes. However, the entire velocity vector is stored at the face as described in Figure 6.1d. As in PolyMAC_P0, the gradient is reconstructed by using MPFA methods. However, time schemes ICE and SETS are not sufficient for using PolyVEF_P0. We remind that no validation have been made for this scheme and that it is not available in the TRUST environment. We do not recommend to use this scheme for most cases.

6.4 VDF

This discretisation method is designed to be compatible with conforming meshes composed of hexahedral elements. It is a conservative Finite Volume scheme of MAC [**harlow1965numerical**] type. All scalars are stored at the center of each control volume when the velocity field which is defined on a staggered mesh. It is summarized on Figure 6.1a. It must be emphasised that a hexahedral mesh is not considered being a cartesian mesh. In practise, hexahedral grids are not always cartesian.

This discretisation allows 2D axi-symetrical configurations. It can be activated using the keyword `bidim_axi`.

6.5 Boundary conditions

An object called a boundary condition is used to define, for a given equation, the conditions to be applied on a domain's boundary. Each boundary condition object contains a reference to the `Domain_Cl_dis_base` object to which it belongs. Each object also contains a `Champ_front` object containing the values to be imposed on the boundary. The main `Champ_front` used are summarized in the following table 6.2.

Keyword	Steady	Uniform
`champ_front_uniforme`	✓	✓
`champ_front_fonc_xyz`	✓	✗
`champ_front_fonc_t`	✗	✓
`champ_front_fonc_txyz`	✗	✗

Table 6.2: Main Champ_front for boundary conditions.

In multiphase problems `champ_front_composite` can be used to associate different boundary conditions to each phase. For example:

```
champ_front_composite 2
{
    champ_front_uniforme 3 0 0 1
    champ_front_uniforme 3 0 0 1
}
```

A special `Champ_Front_MUSIG` associated with `Milieu_MUSIG` can be used to give boundary conditions to each phase. For example:

```
Champ_Front_MUSIG {
    nbPhases 1 Champ_Front_Uniforme 1 0.03 # Gas 0 #
    nbPhases 1 Champ_Front_Uniforme 1 0.04 # Gas 1 #
    nbPhases 1 Champ_Front_Uniforme 1 0.05 # Gas 2 #
    nbPhases 1 Champ_Front_Uniforme 1 0.07 # Gas 3 #
    nbPhases 1 Champ_Front_Uniforme 1 0.03 # Liquid 0 #
    nbPhases 1 Champ_Front_Uniforme 1 0.05 # Liquid 1 #
    nbPhases 1 Champ_Front_Uniforme 1 0.07 # Liquid 2 #
    nbPhases 1 Champ_Front_Uniforme 1 0.09 # Liquid 3 #
    nbPhases 1 Champ_Front_Uniforme 1 0.11 # Liquid 4 #
    nbPhases 1 Champ_Front_Uniforme 1 0.13 # Liquid 5 #
    nbPhases 1 Champ_Front_Uniforme 1 0.15 # Liquid 6 #
```

```
   nbPhases 1 Champ_Front_Uniforme 1 0.17 # Liquid 7 #
}
```

We present in the following tables the main boundary conditions available for the mass (Table 6.3), momentum (Table 6.4) and energy equations (Table 6.5).

Condition	Keyword
Dirichlet	`frontiere_ouverte`
Wall	`paroi`
Symmetry	`symetrie`
Mass flux	`Neumann_paroi`

Table 6.3: Boundary conditions for mass equation.

Condition	Keyword
Dirichlet	`frontiere_ouverte`
Dirichlet on velocity inlet	`frontiere_ouverte_vitesse_imposee`
Dirichlet on velocity of open boundary	`frontiere_ouverte_vitesse_imposee_sortie`
Dirichlet on pressure of open boundary	`frontiere_ouverte_pression_imposee`
Dirichlet on tangential velocity	`paroi_defilante`
Wall function	`Paroi_frottante_loi`
Symmetry	`symetrie`
Adhesion to the wall	`paroi_fixe`

Table 6.4: Boundary conditions for momentum equation.

Condition	Keyword
Dirichlet on temperature	`frontiere_ouverte`
Dirichlet on temperature for wall	`paroi_temperature_imposee`
Adiabatic wall	`paroi_adiabatique`
Wall flux	`paroi_flux_impose`
Symmetry	`symetrie`
Zero diffusion flux	`frontiere_ouverte_temperature_imposee`

Table 6.5: Boundary conditions for energy equation.

Remark : *No periodic boundary conditions or mixed conditions are implemented. We observe that the second one could be easier to implement.*

Chapter 7

Two-fluid physical modeling

This chapter details the physical models for the Two-fluid approach. First, the definition of fluid properties is defined using two methods, through the dataset (section 7.1 and through an external software (section 7.2). Then the interfacial forces are described in section 7.3. Some particular source terms for the momentum (section 7.4) and the mass equations (section 7.5). The management of the dispersed phase diameter is described in section 7.6.

7.1 Fluid proprieties

7.1.1 Basic proprieties of fluid

The domain used for the different phases must be defined by a medium characterized by one or more fluid subdomains of different properties. This composite environment is defined by `Milieu_composite`.
The models aims to give the basic proprieties of fluids such as :

- Kinematic viscosity μ
- Density ρ
- Thermal diffusivity α
- Conductivity λ
- Calorific capacity Cp
- Thermal dilatation coefficient β_{co}

The model is implemented in `Fluide_reel_base` as

```
void Fluide_reel_base::set_param(Param& param)
{
  param.ajouter("T_ref", &T_ref_);
  param.ajouter("P_ref", &P_ref_);
  set_additional_params(param);
}
```

The fluid proprieties operator must fill following tabs :

- `rho_` density
- `dP_rho_` density derivative regarding pressure

- `dT_rho_` density derivative regarding temperature
- `h_` enthalpy
- `dP_h_` enthalpy derivative regarding pressure
- `dT_h_` enthalpy derivative regarding temperature
- `cp_` heat capacity
- `beta_` thermal dilatation coefficient
- `mu_` kinematic viscosity
- `lambda_` conductivity

Non-compressible fluid

The model is implemented in `Fluide_Incompressible` (from TRUST incompressible) :

```
void Fluide_Incompressible::set_param(Param& param)
{
  Fluide_base::set_param(param);
  //La lecture de rho est rendue obligatoire ici
  param.supprimer("rho");
  param.ajouter("rho",&rho,Param::REQUIRED);
}
```

This class allows to give the proprieties previously introduced (or are set to 0) but with a density that is mandatory. Let's notice that here neither the density nor the specific heat can be a `Champ_Uniforme`.

Sodium gas

The model is implemented in `Fluide_sodium_gaz` :

```
void Fluide_sodium_gaz::set_param(Param& param)
```

The model implemented reads `Lois_sodium.h` and writes:

- "temperature", 371 - 273.15, 2503.7 - 273.15 , //Tri-critical point temperature "pression", 4.127e-6, 260e5 ; // Associated pressure

Sodium liquid

The model is implemented in `Fluide_sodium_liquide`:

```
void Fluide_sodium_liquide::set_param(Param& param)
```

The model implemented reads `Lois_sodium.h` and writes:

- "temperature", 371 - 273.15, 2503.7 - 273.15 ; //de la temperature de solidification au pt tricritique

Stiffened gas

The model is implemented in `Fluide_stiffened_gas`:

```
void Fluide_stiffened_gas::set_param(Param& param)
{
  Fluide_reel_base::readOn(is);
  if (Cv_ == -1.) Cv_ = R_ / (gamma_ - 1.0);
  return is;
}
void Fluide_stiffened_gas::set_param(Param& param)
{
  Fluide_reel_base::set_param(param);
  param.ajouter("gamma",&gamma_); // Heat ratio
  param.ajouter("pinf",&pinf_); // Reference pressure in law
  param.ajouter("mu",&mu__); // viscosity
  param.ajouter("lambda",&lambda__); // conductivity
  param.ajouter("Cv",&Cv_); // heat capacity
  param.ajouter("q",&q_);
  param.ajouter("q_prim",&q_prim_);
}
```

The model implemented is:

- `rho_` $= \frac{p+\texttt{pinf\_}}{(\texttt{gamma\_}-1.0)(T+273.15)\texttt{Cv\_}}$, the density
- `dP_rho_` $= \frac{1.0}{(\texttt{gamma\_}-1.0)(T+273.15)(\texttt{Cv\_})}$, the density derivative regarding pressure
- `dT_rho_`$= -\frac{p+\texttt{pinf\_}}{(\texttt{gamma\_}-1.0)(T+273.15)^2\texttt{Cv\_}}$, the density derivative regarding temperature
- `h_` $=$ `gamma_Cv_` $(T + 273.15) +$ `q_`, the enthalpy
- `dP_h_` $= 0.$, the enthalpy derivative regarding pressure
- `dT_h_`$=$ `cp_` (T, P), the enthalpy derivative regarding temperature
- `cp_` $=$ `gamma_Cv_`, the heat capacity
- `beta_`$= \frac{1.0}{T+273.15}$
- `mu_`$=$ `mu__`
- `lambda_`$=$ `lambda__`

R12_C1 and Eau_c3

The models are implemented in `Fluide_R12_c1_liquide`, `Fluide_R12_c1_gaz`, `Fluide_eau_c3_liquide` and `Fluide_eau_c3_gaz`. The low boiling of refrigerant R12 (dichlorodifluoromethane) mimics Pressurized water reactor dimensionless numbers. Those values are available in Cathare code. For example, in **KREPPER20113851**, one can find :

Fluid	Water	R12
Pressure [bar]	155	26
T_{sat} [°C]	344.9	86.5
ρ_{liquid} [kg/m3]	594.4	1019.3
Cp_{liquid} [kJ/kg]	8.950	1.413
λ_{liquid} [W/m/K]	0.472	0.0458
μ_{liquid} [W/m/K]	6.82×10^{-5}	9.23×10^{-5}
ρ_{steam} [kg/m3]	101.9	170.7
Cp_{steam} [kJ/kg]	14.0	1.281
λ_{steam} [W/m/K]	0.126	0.0175
μ_{steam} [W/m/K]	2.30×10^{-5}	1.57×10^{-5}
σ [J/m2]	4.65×10^{-3}	1.80×10^{-3}
L_{vap} [kJ/kg]	966.2	86.48

MUSIG fluid and medium

In homogeneous MUlti-SIze Group (MUSIG), a distribution of bubbles or droplets between r_{min} and r_{max} is characterized by a statistical law (linear, exponential or log) and its discretization in n sub groups with the same velocity, as depicted in Figures 7.1 from **cheung**. All bubbles

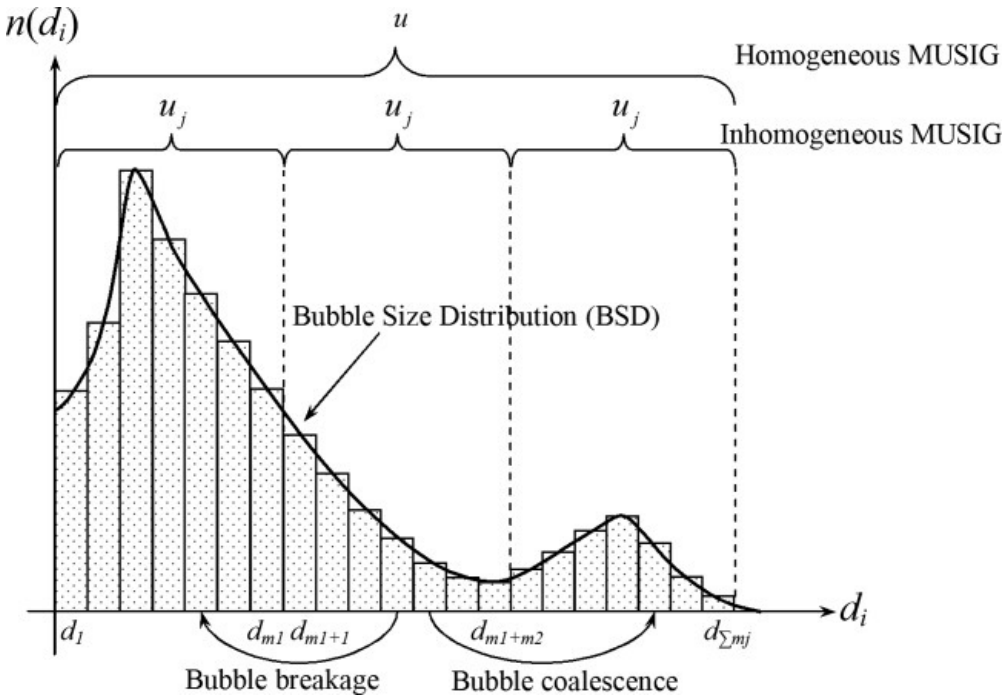


Figure 7.1: Schema of the standard MUSIG and i-MUSIG models. In MUSIG, all size groups move with the same velocity field, whereas i-MUSIG displays an arbitrary number of velocity groups.

from sub groups share the same velocity as the velocity of the mean Sauter diameter of the distribution D_{sm}. Thus, they share the same common total void fraction equation (mass), velocity (momentum equation) and temperature, allowing to compute only one common set of equations (mass, momentum and energy) with new source terms linked to the changes in the distribution (dilatability, coalescence, break-up).
The model is implemented in `Fluide_MUSIG` and `Milieu_MUSIG` :

```
void Fluide_MUSIG::set_param(Param& param)
```

```
{
if(Motcle(mot) == "NBPHASES")
if (Motcle(mot) == "RMIN")
if (Motcle(mot) == "RMAX")
if(Motcle(mot) == "DIAMETRES" || Motcle(mot) == "DIAMETERS")
if (Motcle(mot) == "LIN")
{
    repartionType=0;
}
if (Motcle(mot) == "EXP")
{
    repartionType=1;
}
if (Motcle(mot) == "LOG")
{
    repartionType=2;
}
}
```

Default value : `nbSubPhases_` $= -1$, `rMin` $= -1$, r`Max` $= -1$.
Example of data set to perform MUSIG computation :

```
Milieu_MUSIG
{
    gaz_helium FLUIDE_MUSIG
    {
        fluide StiffenedGas { gamma 1.4 pinf 0.0 }
        nbPhases 4
        diametres { rmin 0.01 rmax 0.1 lin }
    }
    liquide_sodium FLUIDE_MUSIG
    {
        fluide StiffenedGas { gamma 4.4 pinf 6e8 }
        nbPhases 8
    }
}
```

7.1.2 Interfacial proprieties

The liquid-gas interface can be characterized by the surface tension σ. The model is implemented in `Interface_base` as

```
void Interface_base::set_param(Param& param)
{
  Param param(que_suis_je());
  set_param(param);
  param.lire_avec_accolades_depuis(is);
  return is;
}
```

Default values : `sigma__` $= -1$.
The interfacial propriety operator must fill `sigma_`.

Constant surface tension

The model is implemented in `Interface_sigma_constant` :

```
void sigma_(const SpanD T, const SpanD P, SpanD res, int ncomp = 1, int ind =
   0) const override
 {
   for (int i =0; i < (int)P.size(); i++)
     res[i * ncomp + ind] = sigma__;
 }
```

7.1.3 Saturation proprieties

The model aims to give the thermal proprieties at saturation conditions. This class inherits from `Interface`. The model is implemented in `Saturation_base` as

```
void Saturation_base::set_param(Param& param)
{
  Interface_base::set_param(param);
  param.ajouter("P_ref", &P_ref_);
  param.ajouter("T_ref", &T_ref_);
}
```

The saturation proprieties operator must fill following tabs :

- `Tsat`: saturation temperature
- `dP_Tsat`: saturation temperature derivative regarding pressure
- `Psat`: saturation pressure
- `dT_ Psat`: saturation pressure derivative regarding temperature
- `Lvap`: phase change enthalpy
- `dP_Lvap`: phase change enthalpy regarding pressure
- `Hls`: enthalpy of liquid phase at saturation
- `dP_Hls`: derivative of enthalpy of liquid phase at saturation regarding pressure
- `Hvs`: enthalpy of steam phase at saturation
- `dP_Hvs`: derivative of enthalpy of steam phase at saturation regarding pressure

Warning: We suppose that we have a unique pressure field to compute it.

Constant saturation proprieties

The model is implemented in `Saturation_constant`:

```
void Saturation_constant::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("Tsat", &tsat_,Param::REQUIRED); // Temperature at saturation
  param.ajouter("Psat", &psat_,Param::REQUIRED); //Pressure at saturation
```

```
  param.ajouter("Lvap", &lvap_); // Phase change enthalpy
  param.ajouter("Hlsat", &hls_); // Enthalpy of liquid phase at saturation
  param.ajouter("Hvsat", &hvs_); // Enthalpy of steam phase at saturation
  param.ajouter("tension_superficielle", &sigma__); // Surface tension
  param.lire_avec_accolades_depuis(is);
  // verifications hlsat/hvsat/lvap
  const int i = (lvap_ > 0) + (hls_ > 0) + (hvs_ > 0);
  if (i != 2) Process::exit(que_suis_je() + " Please give 2 properties among {
     Lvap, Hlsat, Hvsat}");
  if (lvap_ > 0 && hls_ > 0) hvs_ = hls_ + lvap_;
  else if (lvap_ > 0 && hvs_ > 0) hls_ = hvs_ - lvap_;
  else if (hls_ > 0 && hvs_ > 0) lvap_ = hvs_ - hls_;
  else Process::exit(que_suis_je() + "bad parameters");
  return is;
}
```

The model implemented is:

- $T_{sat} =$ tsat_
- $\frac{DT_{sat}}{Dp} = 0$
- $P_{sat} =$ psat_
- $\frac{Dp_{sat}}{DT} = 0$
- $L_{vap} =$ lvap_
- $\frac{DL_{vap}}{Dp} = 0$
- $H_{ls} =$ hls__
- $\frac{DH_{ls}}{Dp} = 0$
- $H_{vs} =$ hvs__
- $\frac{DH_{vs}}{Dp} = 0$
- sigma_ = sigma__

Sodium saturation proprieties

The model is implemented in Saturation_sodium :

```
void Saturation_sodium::set_param(Param& param)
{
return Saturation_base::readOn(is);
}
```

R12_C1 and Eau_c3

The models are implemented in `Fluide_R12_c1_liquide`, `Fluide_R12_c1_gaz`, `Fluide_eau_c3_liquide` and `Fluide_eau_c3_gaz`. The low boiling of refrigerant R12 (dichlorodifluoromethane) mimics Pressurized water reactor dimensionless numbers. Those values are available in Cathare code.

7.2 Fluid proprieties from external software

In this section, the management of fluid properties through an external software is described. One can use EOS, the CEA-EDF software for state equation management (section xx), or CoolProp, an open-source alternative. Both EOS and CoolProp can call the NIST state equation software named Refprop. EOS can use a user-defined state equation through a plugin. CoolProp provides free state equations for a selected list of fluids[1].

7.2.1 Basic proprieties of fluid

The generic table of proprieties from external software is implemented in `Fluide_generique_TPPI_base` :

```
void Fluide_generique_TPPI_base::set_param(Param& param)
#
if (tmax_ < -100. )
  return Fluide_reel_base::unknown_range();
return { { "temperature", { tmin_ - 273.15, tmax_ - 273.15 } }, { "pression",
    { pmin_, pmax_ } } };
}
```

Default values : $tmin\_ = -123.$, $tmax\_ = -123.$, $pmin\_ = -123.$, $pmax\_ = -123.$

- `rho_` density
- `dP_rho_` density derivative regarding pressure
- `dT_rho_` density derivative regarding temperature
- `h_` enthalpy
- `dP_h_` enthalpy derivative regarding pressure
- `dT_h_` enthalpy derivative regarding temperature
- `cp_` heat capacity
- `beta_` thermal dilatation coefficient
- `mu_` kinematic viscosity
- `lambda_` conductivity

CoolProp

The model is implemented in `Fluide_generique_CoolProp`:

```
void Fluide_generique_CoolProp::set_param(Param& param)
{
param.ajouter("model|modele", &model_name_, Param::REQUIRED);
param.ajouter("fluid|fluide", &fluid_name_, Param::REQUIRED);
TPPI_ = std::make_shared<EOS_to_TRUST_generique>();
TPPI_->set_fluide_generique(model_name_, fluid_name_);
TPPI_->desactivate_handler(false); // throw on error
```

[1] Available here http://www.coolprop.org/fluid_properties/index.html

```
if (model_name_ == "CATHARE2" || model_name_ == "EOS_CATHARE2")
{
      tmin_ = TPPI_->tppi_get_T_min();
      tmax_ = TPPI_->tppi_get_T_max();
      pmin_ = TPPI_->tppi_get_p_min();
      pmax_ = TPPI_->tppi_get_p_max();
}
}
```

Validity domain for Cathare tables with `tmin_`, `tmax_`, `pmin_`, `pmax_`.
Accessible at http://www.coolprop.org/v4/index.html.

EOS

The model is implemented in `Fluide_generique_EOS` :

```
void Fluide_generique_EOS::set_param(Param& param)
{
param.ajouter("model|modele", &model_name_, Param::REQUIRED);
param.ajouter("fluid|fluide", &fluid_name_, Param::REQUIRED);
param.ajouter("phase", &phase_, Param::OPTIONAL); // optional : liquid or
    vapor. PI : specify the phase it is really useful (better perf for coolprop
    ) !
if (model_name_ == "REFPROP") TPPI_->set_path_refprop();
  tmin_ = TPPI_->tppi_get_T_min();
  tmax_ = TPPI_->tppi_get_T_max();
  pmin_ = TPPI_->tppi_get_p_min();
  pmax_ = TPPI_->tppi_get_p_max();
}
```

Validity domain for Refprop tables with `tmin_`, `tmax_`, `pmin_`, `pmax_`.
Availability of 147 pure fluids, 5 pseudo-pure fluids (such as air), and mixtures with up to 20 components. Accessible at https://www.nist.gov/srd/refprop.

For instance, a boiling flow using `refprop10` must be specified as

```
liquide_eau Fluide_generique_EOS { model refprop10 fluid waterliquid }
gaz_eau Fluide_generique_EOS { model refprop10 fluid watervapor }
```

7.2.2 Saturation proprieties

The generic table of proprieties from external software is implemented in `Saturation_generique_TPPI_base` :

```
void Saturation_generique_TPPI_base::set_param(Param& param)
```

The saturation proprieties operator must fill following tabs :

- `Tsat` saturation temperature
- `dP_Tsat` saturation temperature derivative regarding pressure
- `Psat` saturation pressure

- `dT_Psat` saturation pressure derivative regarding temperature
- `Lvap` phase change enthalpy
- `dP_Lvap` phase change enthalpy regarding pressure
- `Hls` enthalpy of liquid phase at saturation
- `dP_Hls` derivative of enthalpy of liquid phase at saturation regarding pressure
- `Hvs` enthalpy of steam phase at saturation
- `dP_Hvs` derivative of enthalpy of steam phase at saturation regarding pressure

CoolProp

The model is implemented in `Fluide_generique_CoolProp` :

```
void Fluide_generique_CoolProp::set_param(Param& param)
{
if (model_name_ == "REFPROP") TPPI_->set_path_refprop();
param.ajouter("model|modele", &model_name_, Param::REQUIRED);
param.ajouter("fluid|fluide", &fluid_name_, Param::REQUIRED);
param.ajouter("phase", &phase_, Param::OPTIONAL); // optional : liquid or
   vapor. PI : specify the phase it is really useful (better perf for coolprop
   ) !
param.ajouter("sigma_mano", &sigma_mano_, Param::OPTIONAL); // optional :
   because of issues when we call surface tension in TTSE in coolprop ! Try
   without and if calculation doesn't pass, input sigma
}
```

Default value : `sigma_mano_` $= -1$.

EOS

The model is implemented in `Fluide_generique_CoolProp` :

```
void Fluide_generique_CoolProp::set_param(Param& param)
{
TPPI_ = std::make_shared<EOS_to_TRUST_Sat_generique>();
param.ajouter("model|modele", &model_name_, Param::REQUIRED);
param.ajouter("fluid|fluide", &fluid_name_, Param::REQUIRED);
param.ajouter("sigma_mano", &sigma_mano_, Param::OPTIONAL); // optional :
   because of issues when we call surface tension in TTSE in coolprop ! Try
   without and if calculation doesn't pass, input sigma
}
```

Default value : `sigma_mano_` $= -1$.
For example :

```
saturation_eau saturation_generique_EOS { model refprop10 fluid waterliquid }
```

7.3 Interfacial forces

The interfacial forces defined below are source terms that can be added to the momentum equation and represent the mechanical interactions between the phases. Then the equations of motion for each phase velocities are coupled by the interfacial forces.

7.3.1 The Drag force

The general expression of the drag force is:

$$\overrightarrow{F^D_{l\to g}} = -\frac{3}{4} C_D \frac{\alpha_g \rho_l}{d_b} ||\vec{u_g} - \vec{u_l}|| \, (\vec{u_g} - \vec{u_l}) = -f^D ||\vec{u_g} - \vec{u_l}|| \, (\vec{u_g} - \vec{u_l}) \tag{7.1}$$

The force is implemented in :

```
void Source_Frottement_interfacial_base::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("a_res", &a_res_);
  param.ajouter("dv_min", &dv_min);
  param.ajouter("exp_res", &exp_res);
  param.ajouter("beta", &beta_);
  param.lire_avec_accolades_depuis(is);

  const Pb_Multiphase& pbm = ref_cast(Pb_Multiphase, equation().probleme());
  if (pbm.has_correlation("frottement_interfacial")) correlation_ = pbm.
     get_correlation("frottement_interfacial"); //correlation fournie par le
     bloc correlation
  else correlation_.typer_lire(pbm, "frottement_interfacial", is); //sinon ->
     on la lit
  return is;
}
```

Default values : `a_res_` $= -1.$, `dv_min` $= 0.01$, `beta_` $= 1.$, `exp_res` $= 2.$
The interfacial drag operator must fill coeff tab so that :

- coeff$(k1, k2, 0) = f^D ||\overrightarrow{u_{k1}} - \overrightarrow{u_{k2}}||$;
- coeff$(k1, k2, 1) = f^D$; (it is the derivative of coeff$(k1, k2, 0)$ with respect to the relative velocity)
- coeff$(k2, k1, 0) =$ coeff$(k1, k2, 0)$;
- coeff$(k2, k1, 1) =$ coeff$(k1, k2, 1)$;

Constant drag coefficient

The model is implemented in :

```
void Frottement_interfacial_bulles_constant::set_param(Param& param)
{
param.ajouter("coeff_derive", &C_d_, Param::REQUIRED);
param.ajouter("rayon_bulle", &r_bulle_);
}
```

Model	Used	Validated	Test case
Constant	✓	✓(100%)	TrioCFD/Tube analytique, Trust/Tube analytique
Composant	✓	✗(0%)	
Ishii Zuber Deformable	✓	✗(0%)	
Ishii Zuber	✓	✗(0%)	
Tomiyama	✓	✓(100%)	TrioCFD/CoolProp, TrioCFD/Gabillet
Weber	✓	✗(0%)	
Wallis	✓	✗(0%)	
Sonnenburg	✓	✗(0%)	
Garnier	✓	✗(0%)	
Rusche	✓	✗(0%)	
Simmonet	✓	✗(0%)	
Zenit	✓	✗(0%)	

Table 7.1: Availability of drag force models in Trio_CMFD.

Default values : `r_bulles_` $= -1$, `C_d_` $= -123.$.
If no rayon_bulle takes d_bulles.
The model implemented is :

$$f^D = \frac{3}{4}\frac{C_d \alpha_g \rho_l}{d_b}. \tag{7.2}$$

Composant drag coefficient

The model is implemented in :

```
void Frottement_interfacial_bulles_composant::set_param(Param& param)
{
param.ajouter("coeff_derive", &C_d_, Param::REQUIRED);
param.ajouter("rayon_bulle", &r_bulle_);
}
```

Default values `r_bulle_` $= -100$, `C_d` $= -100.$.
The model implemented is :

$$f_{ij}^D = \frac{3}{4}\frac{C_d \alpha_i \alpha_j \rho_m}{d_b}, \tag{7.3}$$

with $\rho_m = \sum_k \alpha_k \rho_k$ and $i \neq j$.

Ishii-Zuber : viscous regime

The model is described in **IshiiZuber**.
The model is implemented in:

```
void Frottement_interfacial_Ishii_Zuber_Deformable::set_param(Param& param)
{
  param.ajouter("beta", &beta_);
  param.ajouter("constante_gravitation", &g_);
}
```

Default values : `g_` $= 9.81$, `beta_` $= 1.$.
The model implemented is :

$$f^D = \frac{1}{2}\alpha_g \rho_l \sqrt{\frac{(\rho_l - \rho_g) \times g}{\sigma}} \frac{1}{\sqrt{max\left(1 - \alpha_g,\ 0.001\right)}}. \tag{7.4}$$

If $\alpha_l < 1.0 \times 10^{-6}$, then $f^D \times \alpha_l \times 1 \times 10^6$.

Ishii-Zuber : viscous regime and particle regime

The model is also described in **IshiiZuber**.
The model is implemented as :

```
void Frottement_interfacial_Ishii_Zuber::set_param(Param& param)
{
  param.ajouter("beta", &beta_);
  param.ajouter("constante_gravitation", &g_);
}
```

Default values `g_` $= 9.81$, `beta_` $= 1.$. The model implemented is :

$$f^D = \frac{3}{4} \frac{\max\left(\frac{24}{Re_b}\left(1 + 0.1 Re_b^{0.75}\right),\ \frac{2}{3}\sqrt{\frac{(\rho_l - \rho_g) g d_b^2}{\sigma}}\right) \beta \alpha_g \rho_l}{d_b}, \tag{7.5}$$

with $Re_b = \frac{\rho_l d (u_g - u_l)}{\mu_l}$.

Tomiyama : contaminated drag coefficient

The model is described in **Tomiyama1998**.
The model is implemented in :

```
void Frottement_interfacial_Tomiyama::set_param(Param& param)
{
  param.ajouter("beta", &beta_);
  param.ajouter("constante_gravitation", &g_);
  param.ajouter("contamination", &contamination_);
}
```

Default values : `g_` $= 9.81$, `beta_` $= 1.$, $contamination = 0$.
The model implemented is :

$$f_D = \frac{3}{4}\frac{\alpha_g \rho_l}{d} \begin{cases} \max(\min(16/Re_b(1 + .15Re^{.687}), 48/Re_b), 8Eo/(3 + 12)), \text{ No contamination } (0) \\ \max(\min(24/Re_b(1 + .15Re_b^{.687}), 72/Re_b), 8Eo/(3Eo + 12)), \text{ Slight contamination } (1) \\ \max(24/Re_b(1 + .15Re_b^{.687}), 8Eo/(3Eo + 12)), \text{ High contamination } (2) \end{cases} \tag{7.6}$$

with

- $Eo = \frac{g(\rho_l - \rho_v) d^2}{\sigma}$,
- $Re_b = \frac{\rho_l d_b (u_g - u_l)}{\mu_l}$.

If $\alpha_l < 1.0 \times 10^{-6}$, then $f^D \times \alpha_l \times 1 \times 10^6$.
This formulation was chosen as shown in **Sugrue2017**, it yields similar results as other closures and one can adjust the level of contamination.

Bubble critical diameter (incoming)

The model is described partially in **KUO1988547**.
The model is implemented in :

```
void Frottement_interfacial_Weber::set_param(Param& param)
{
    param.ajouter("Weber_critique", &We_c);
}
```

Default values $We\_c = 8.$.
The model implemented is :

$$f^D = \frac{6\alpha_g}{\pi d_b^{*3}} \frac{24}{Re_b}(1 + 0.1Re_b^{0.75}), \tag{7.7}$$

with $d_b^* = \frac{\sigma}{\rho_l(u_g-u_l)^2} We_c$, $Re_b = \frac{\rho_l d_b^*(u_g-u_l)}{\mu_l}$.
Warning : not homogeneous

Wallis : annular flow

The model is described in **wallis**.
The model is implemented in :

```
void Frottement_interfacial_Wallis::set_param(Param& param)
```

The model implemented is :

$$f^D = 5 \times 10^{-3} \times \rho_g \frac{4\sqrt{\alpha_g}}{D_h} \left(1 + 300 \frac{1 - \sqrt{1-\alpha_g}}{2}\right) \tag{7.8}$$

Sonnenburg : drift flux ?

The model is described in .
The model is implemented in :

```
void Frottement_interfacial_Sonnenburg::set_param(Param& param)
```

The model implemented is :

$$f^D = \rho_l \frac{\alpha_l \alpha_g}{D_h} \left(\frac{16}{9}(1 - \alpha_g^*(1 - \frac{9}{16}\sqrt{\frac{\rho_g}{\rho_l}})) \frac{1 - \alpha_g^{*40}}{tanh(32\alpha_g^*)}\right)^2, \tag{7.9}$$

with $\alpha_g^* = min(max(\alpha_g, 0.001), 0.999)$

Garnier : bubble swarm correction

The model is described in **GARNIER2002811**.
The model is implemented in :

```
void Frottement_interfacial_Garnier::set_param(Param& param)
```

The model implemented is :

$$f^D_{new} = f^D \begin{cases} \alpha_l \times 114.2, & \text{if } \alpha_l < 0.5 \\ \left(1 - \alpha_g^{1/3}\right)^{-2}, & \text{if not.} \end{cases} \tag{7.10}$$

Warning : Validated for $\alpha_g < 0.35$, $D_{sm} < 5.5mm$.

Rusche : swarm correction

The model is described in **Rusche**.
The model is implemented in :

```
void Frottement_interfacial_Rusche::set_param(Param& param)
```

The model implemented is :

$$f^D_{new} = f^D \left(\exp\left(3.64\alpha_g\right) + \alpha_g^{0.864} \right) \tag{7.11}$$

Warning : Validated for $\alpha_g < 0.5$.

Simonnet : bubble swarm correction

The model is described in **SIMONNET2007858**.
The model is implemented in :

```
void Frottement_interfacial_Simonnet::set_param(Param& param)
```

The model implemented is :

$$f^D_{new} = f^D \alpha_l \left(\alpha_l^{25} + \left(4.8 \frac{\alpha_g}{\alpha_l} \right)^{25} \right)^{-2/25} \tag{7.12}$$

Warning : Validated for $\alpha_g < 0.3$, $D_{sm} < 10mm$.

Zenit : bubble swarm correction

The model is described in **zenit**.
The model is implemented in :

```
void Frottement_interfacial_Zenit::set_param(Param& param)
```

The model implemented is :

$$f^D_{new} = f^D \frac{\left(1 + 3\alpha_g\right)^2}{\alpha_l^2} \tag{7.13}$$

Warning : Validated for $\alpha_g < 0.18$.

7.3.2 The Lift force

The general expression of the lift force is :

$$\overrightarrow{F^L_{l \to v}} = -C_L \rho_l \alpha_g \left(\overrightarrow{u_g} - \overrightarrow{u_l} \right) \wedge \overrightarrow{u_l} = -f^L \left(\overrightarrow{u_g} - \overrightarrow{u_l} \right) \wedge \overrightarrow{u_l} \tag{7.14}$$

The force is implemented in :

```
void Source_Portance_interfaciale_base::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("beta", &beta_);
  param.ajouter("g", &g_);
  param.lire_avec_accolades_depuis(is);
```

```
  Pb_Multiphase *pbm = sub_type(Pb_Multiphase, equation().probleme()) ? &
     ref_cast(Pb_Multiphase, equation().probleme()) : NULL;

  if (!pbm || pbm->nb_phases() == 1) Process::exit(que_suis_je() + "␣:␣not␣
     needed␣for␣single-phase␣flow!");

  for (int n = 0; n < pbm->nb_phases(); n++) //recherche de n_l, n_g : phase {
     liquide,gaz}_continu en priorite
    if (pbm->nom_phase(n).debute_par("liquide") && (n_l < 0 || pbm->nom_phase(
       n).finit_par("continu"))) n_l = n;

  if (n_l < 0) Process::exit(que_suis_je() + "␣:␣liquid␣phase␣not␣found!");

  if (pbm->has_correlation("Portance_interfaciale")) correlation_ = pbm->
     get_correlation("Portance_interfaciale"); //correlation fournie par le
     bloc correlation
  else correlation_.typer_lire((*pbm), "Portance_interfaciale", is); //sinon
     -> on la lit

  pbm->creer_champ("vorticite"); // Besoin de vorticite

  return is;
}
```

Default values : `beta_` $= 1.$, `g_` $= 9.81$.
The interfacial lift operator must fill out.Cl tab so that :

- $Cl(k1, k2) = f^L$;
- $Cl(k2, k1) = Cl(k1, k2)$;

Model	Used	Validated	Test case
Constant	✓	✓(100%)	TrioCFD/Tube analytique, TrioCFD/CoolProp, Trust/Tube analytique
Sugrue	✓	✓(100%)	TrioCFD/CoolProp TrioCFD/Gabillet
Tomiyama	✓	✗(0%)	

Table 7.2: Availability of lift force models in Trio_CMFD.

Constant lift coefficient

The model is implemented in :

```
void Portance_interfaciale_Constante::set_param(Param& param)
{
  param.ajouter("Cl", &Cl_, Param::REQUIRED);
}
```

Default values `Cl_` $= -123$.
The model implemented is :

$$f^L = C_L \rho_l \alpha_g \max\left(\min\left(\frac{\alpha_l - 0.05}{0.25},\ 1\right),\ 0\right) \tag{7.15}$$

The void fraction correction is used to damp the lift at too high void fractions.

Sugrue

The model is described in .
The model is implemented in :

```
void Portance_interfaciale_Sugrue::set_param(Param& param)
{
  param.ajouter("constante_gravitation", &g_);
}
```

Default values `g_` $= 9.81$.
The model implemented is :

$$f^L = \rho_l \alpha_g \max\left(1.0155 - 0.0154 \exp\left(8.0506 \alpha_g\right),\ 0\right) \times \min\left(5.0404 - 5.0781 (Wo)^{0.0108},\ 0.03\right) \tag{7.16}$$

with

- the wobbling number $Wo = \min\left(\frac{k_l Eo}{\max\left((u_g - u_l)^2,\ 1 \times 10^{-8}\right)},\ 6.\right)$
- the Eotvos number $Eo = \frac{g(\rho_l - \rho_g) d_b^2}{\sigma}$

.

Tomiyama : lift sign reversal

The model is described in **Tomiyama2002**.
The model is implemented in :

```
void Portance_interfaciale_Tomiyama::set_param(Param& param)
{
  param.ajouter("constante_gravitation", &g_);
}
```

Default values `g_` $= 9.81$.
The model implemented is :

$$f^L = \rho_l \alpha_g \begin{cases} \min(0.288 \tanh(.121 Re_b), f(Eo)), \text{ if } Eo < 4 \\ f(Eo), \text{ if } 4 \leq Eo \leq 10.7 \end{cases} \tag{7.17}$$

with

- the Eotvos number $Eo = \frac{g(\rho_l - \rho_g) d_b^2}{\sigma}$
- $f(Eo) = 0.00105 \times Eo^3 - 0.0159 \times Eo^2 - 0.0204 \times Eo + 0.474.$

7.3.3 The Added mass force

The general expression of the added mass force is :

$$\overrightarrow{F^{AM}_{l\to v}} = C_{AM}\alpha_g\rho_l\frac{D(u_g-u_l)}{Dt} = f^{AM}\frac{D(u_g-u_l)}{Dt} \tag{7.18}$$

The model is implemented in:

```
void Masse_ajoutee_base::set_param(Param& param)
{
* IN :
 * alpha[n] -> void fraction
 * rho[n] -> density
 *
 * IN/OUT :
 * a_r(k, l) -> update in momentum equation
 *
 * NB: no need of derivative because it is past void fraction
}
```

Default values : `limiter_liquid_` = 0.5.
The added mass operator must add to a_r tab so that:

- $a_r(k1,k1)$ += f^{AM}; coefficient in front of $\frac{D(u_{k1})}{Dt}$ in $k1$ equation
- $a_r(k1,k2)$ −= f^{AM}; coefficient in front of $\frac{D(u_{k2})}{Dt}$ in $k1$ equation
- $a_r(k2,k2)$ += f^{AM}; coefficient in front of $\frac{D(u_{k2})}{Dt}$ in $k2$ equation
- $a_r(k2,k1)$ −= f^{AM}; coefficient in front of $\frac{D(u_{k1})}{Dt}$ in $k2$ equation

Model	Used	Validated	Test case
Constant	✓	✓(100%)	TrioCFD/Tube analytique, Trust/Tube analytique
Wijngaarden	✓	✗(0%)	
Zuber	✓	✗(0%)	

Table 7.3: Availability of added mass force models in Trio_CMFD.

Constant added mass coefficient

The model is implemented in :

```
void Masse_ajoutee_Coef_Constant::set_param(Param& param)
{
  param.ajouter("beta", &beta);
  param.ajouter("inj_ajoutee_liquide", &inj_ajoutee_liquide_);
  param.ajouter("inj_ajoutee_gaz", &inj_ajoutee_gaz_);
  param.ajouter("limiter_liquid", &limiter_liquid_);
}
```

Default values : `beta` = 0.5, `inj_ajoutee_liquid_` = 1., `inj_ajoutee_gaz_` = 1., `limiter_liquid_` = 0.5.
The model implemented is :

$$f^{AM} = \min\left(\beta\rho_l\alpha_g,\ \rho_l\alpha_l \times \text{limiter\_liquid\_}\right) \tag{7.19}$$

Warning : direct void fraction influence limited at $\alpha_{gmax} = \frac{\texttt{limiter\_liquid\_}}{\texttt{limiter\_liquid\_}+\texttt{beta}}$. Default value $\alpha_{gmax} = 0.5$.
For the injected mass flux $\dot{m}_{inj}$,

$$\dot{m}_{inj} = min(\texttt{beta}\rho_l, \texttt{limiter\_liquid\_}\times\frac{\alpha_l}{\alpha_g})\times\dot{m}\times\begin{cases}\texttt{inj\_ajoutee\_gaz\_}, for\ gas\ phase,\\ \texttt{inj\_ajoutee\_liquid\_}, for\ liquid\ phase.\end{cases} \tag{7.20}$$

If $\alpha_g < 0.0001$, no limiter part.

Wijngaarden : two bubbles interaction

The model is described in **Biesheuvel1984**.
The model is implemented in :

```
void Masse_ajoutee_Wijngaarden::set_param(Param& param)
{
  param.ajouter("beta", &beta);
  param.ajouter("inj_ajoutee_liquide", &inj_ajoutee_liquide_);
  param.ajouter("inj_ajoutee_gaz", &inj_ajoutee_gaz_);
  param.ajouter("limiter_liquid", &limiter_liquid_);
}
```

Default values : `beta` = 0.5, `inj_ajoutee_liquid_` = 1., `inj_ajoutee_gaz_` = 1., `limiter_liquid_` = 0.5.
The model implemented is :

$$f^{AM} = \min\left(\beta(1+2.78\alpha_g)\rho_l\alpha_g, \rho_l\alpha_l \times \texttt{limiter\_liquid\_}\right) \tag{7.21}$$

Warning : direct void fraction influence limited at :

$$\alpha_{gmax} = \frac{5}{139\beta}(\sqrt{25\texttt{beta} + 328\texttt{beta}\times\texttt{limiter\_liquid\_} + 25\texttt{limiter\_liquid\_}^2} \\ -5(\texttt{beta}+\texttt{limiter\_liquid\_})). \tag{7.22}$$

Default value : $\alpha_{gmax} \approx 0.34$
For the injected mass flux $\dot{m}_{inj}$,

$$\dot{m}_{inj} = min(\texttt{beta}(1+2.78\alpha_g), \texttt{limiter\_liquid\_}\frac{\alpha_l}{\alpha_g})\rho_l\dot{m}\times\begin{cases}\texttt{inj\_ajoutee\_gaz\_}, \ for\ gas\ phase,\\ \texttt{inj\_ajoutee\_liquid\_}, \ for\ liquid\ phase.\end{cases} \tag{7.23}$$

If $\alpha_g < 0.0001$, no limiter part.
Warning : Corrected value in **Biesheuvel1984** is 3.32 instead of 2.78.

Zuber : swarm of compliant bubbles

The model is described in **ZUBER1964897**.
The model is implemented in :

```
void Masse_ajoutee_Zuber::set_param(Param& param)
{
  param.ajouter("beta", &beta);
  param.ajouter("inj_ajoutee_liquide", &inj_ajoutee_liquide_);
  param.ajouter("inj_ajoutee_gaz", &inj_ajoutee_gaz_);
  param.ajouter("limiter_liquid", &limiter_liquid_);
}
```

Default values : `beta` $= 0.5$, `inj_ajoutee_liquid_` $= 1.$, `inj_ajoutee_gaz_` $= 1.$, `limiter_liquid_` $= 0.5$.
The model implemented is :

$$f^{AM} = min(\beta \frac{1+2\alpha_g}{max(1-\alpha_g, 0.001)} \rho_l \alpha_g, \rho_l \alpha_l limiter\_liquid\_) \tag{7.24}$$

Warning : direct void fraction influence limited at :

$$\alpha_{gmax} = \begin{cases} \frac{\sqrt{\texttt{beta}^2+12\texttt{beta}\times\texttt{limiter\_liquid\_}}-\texttt{beta}-2\texttt{limiter\_liquid\_}}{2(2\texttt{beta}-\texttt{limiter\_liquid\_})}, & if\ 2\texttt{beta}-\texttt{limiter\_liquid\_} \neq 0 \\ \frac{2}{5}, & otherwise. \end{cases} . \tag{7.25}$$

Default value : $\alpha_{gmax} \approx 0.303$
For the injected mass flux $\dot{m}_{inj}$,

$$\dot{m}_{inj} = \min(\beta \frac{1+2\alpha_g}{\alpha_l}, \texttt{limiter\_liquid\_} \frac{\alpha_l}{\alpha_g}) \rho_l \dot{m} \times \begin{cases} \texttt{inj\_ajoutee\_gaz\_}, & for\ gasphase, \\ \texttt{inj\_ajoutee\_liquid\_}, & for\ liquid\ phase. \end{cases} \tag{7.26}$$

If $\alpha_g < 0.0001$, no limiter part.

7.3.4 The Dispersion force

The general expression of the turbulent dispersion force is :

$$\overrightarrow{F^T_{l\to v}} = -f^T \nabla \alpha_g \tag{7.27}$$

The force is implemented in :

```
void Source_Dispersion_bulles_base::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("beta", &beta_);
  param.lire_avec_accolades_depuis(is);

  Pb_Multiphase *pbm = sub_type(Pb_Multiphase, equation().probleme()) ? &
     ref_cast(Pb_Multiphase, equation().probleme()) : NULL;

  if (!pbm || pbm->nb_phases() == 1) Process::exit(que_suis_je() + "␣:␣not␣
     needed␣for␣single-phase␣flow!");

  if (pbm->has_correlation("Dispersion_bulles")) correlation_ = pbm->
     get_correlation("Dispersion_bulles"); //correlation fournie par le bloc
     correlation
```

```
  else Process::exit(que_suis_je() + " : the turbulent dispersion correlation
     must be defined in the correlation bloc.");

  return is;
}
```

Default values : `beta_` = 1..
The added mass operator must add to out.Ctd tab so that :

- $Ctd(k1, k2) = f^T$; coefficient in front of $\nabla\alpha_{k1}$
- $Ctd(k2, k1) = f^T$; coefficient in front of $\nabla\alpha_{k2}$

Model	Used	Validated	Test case
Constant bubble	✓	✓(100%)	TrioCFD/Tube analytique Trust/Tube analytique
Constant turbulent	✓	✓(100%)	TrioCFD/Tube analytique,
Lopez	✓	✗(0%)	
Burns	✓	✓(100%)	TrioCFD/CoolProp TrioCFD/Gabillet

Table 7.4: Availability of dispersion force models in Trio_CMFD.

Constant bubble dispersion coefficient

The model is described in **MARFAING2016579**.
The model is implemented in :

```
void Dispersion_bulles_turbulente_constante::set_param(Param& param)
{
  param.ajouter("D_td_star", &D_td_star_, Param::REQUIRED);
}
```

The model implemented is :

$$f^T = D_{td}\rho_l(u_g - u_l)^2 \tag{7.28}$$

Constant turbulent dispersion coefficient

The model is described in **LOPEZDEBERTODANO1994805**.
The model is implemented in :

```
void Dispersion_bulles_turbulente_constante::set_param(Param& param)
{
  param.ajouter("C_td", &C_td_);
}
```

Default values : `C_td_` = 0.1.
The model implemented is :

$$f^T = C_{td}\rho_l k_l \tag{7.29}$$

Lopez de Bertodano : Stokes regime

The model is described in **LOPEZDEBERTODANO199865**.
The model is implemented in :

```
void Dispersion_bulles_turbulente_Bertodano::set_param(Param& param)
```

Default values : `Prt_` $= 0.9$.
The model implemented is :

$$f^T = 2\rho_l k_l \frac{1}{(1+St)St}, \tag{7.30}$$

with $St = \frac{\tau^F}{\tau^t}$, $\tau^t = \frac{\nu_t}{k_l}$ and $\tau^F = \frac{\frac{4}{3}\rho_g d}{C_d \rho_l (u_g - u_l)}$.
Warning, in literature :

$$f^T = \rho_l k_l \frac{C_\mu^{1/4}}{(1+St)St}, \tag{7.31}$$

with $\tau^t = C_\mu^{3/4}\frac{k_l}{\varepsilon_l}$.

Burns : Favre averaged drag

The model is described in **burns2004favre**.
The model is implemented in :

```
void Dispersion_bulles_turbulente_constante::set_param(Param& param)
{
  param.ajouter("minimum", &minimum_);
  param.ajouter("a_res", &a_res_);
  param.ajouter("g_", &g_);
  param.ajouter("coefBIA_", &coefBIA_);
}
```

Default values : `Prt_` $= .9$, `minimum_` $= -1.$, `a_res_` $= -1.$, `g_` $= 9.81$, `C_lambda_` $= 2.7$, `gamma_` $= 1.$, `coefBIA_` $= 0.$.
The model implemented is :

$$F^T = \frac{f^D |\overrightarrow{u_g} - \overrightarrow{u_l}|(\nu_t + \nu_{BIA})}{Pr_t}(\frac{1}{\alpha_g}\nabla\alpha_g - \frac{1}{\alpha_l}\nabla\alpha_l) \tag{7.32}$$

with

- $\nu_t = C_\mu \frac{k^2}{\varepsilon}$,
- $\nu_{BIA} = coefBIA\_\frac{k_{WIT}}{\omega_{WIT}}$,
- $\omega_{WIT} = 2\mu_l \frac{Re_b}{C_\lambda^2 d^2} max(min(\frac{16}{Re_b}(1 + 0.15Re_b^{0.687}), \frac{48}{Re_b}), 8\frac{Eo}{3(Eo+4)})$.

Warning in literature :

$$f^T = \frac{f^D |\overrightarrow{u_g} - \overrightarrow{u_l}|\nu_t}{Pr_t}(\frac{1}{\alpha_g} + \frac{1}{1-\alpha_g}) \tag{7.33}$$

7.3.5 The Wall force

Antal : wall lubrication

The model is described in **ANTAL1991635**.
The model is implemented in :

```
void Correction_Antal_PolyMAC_P0::set_param(Param& param)
{
  param.ajouter("Cw1", &Cw1_);
  param.ajouter("Cw2", &Cw2_);
}
```

Default values : `Cw1_` $= -0.1$, `Cw2_` $= 0.147$.
The model implemented is :

$$F^{WL} = C_{WL}\alpha_g\rho_l\frac{(\overrightarrow{u_g} - \overrightarrow{u_l})^2}{d_b}\overrightarrow{n}, \tag{7.34}$$

with

$$C_{WL} = max(-C_{W1} + C_{W2}\frac{d_b}{2y}, 0) \tag{7.35}$$

Warning This force was developped for fully developed laminar bubbly two-phase flows.

Lubchenko : wall force dumping

Tht model is partially described in **LUBCHENKO201836**.
The dumping is implemented in :

```
void Correction_Lubchenko_PolyMAC_P0::set_param(Param& param)
{
  param.ajouter("beta_lift", &beta_lift_);
  param.ajouter("beta_disp", &beta_disp_);
  param.ajouter("portee_disp", &portee_disp_);
  param.ajouter("portee_lift", &portee_lift_);
  param.ajouter("use_bif", &use_bif_);
}
```

Default values : `beta_lift_` $= 1.$, `beta_disp_` $= 1.$, `portee_disp_` $= 1.$, `portee_lift_` $= 1.$.
The wall force dumping implemented is a mix between the one proposed by **Lubchenko2018** and BIF near-wall dumping. The first part is a lift dumping close to the wall as :

$$C_L \rightarrow \begin{cases} 0,\ y/d_b < 1/2 * portee\_lift\_ \\ C_L\left(3\left(\frac{2y}{d_b} - 1\right)^2 - 2\left(\frac{2y}{d_b} - 1\right)^3\right),\ 1/2 * portee\_lift\_ \leq y/d_b < 1 * portee\_lift\_ \\ C_L,\ y/d_b \geq 1 * portee\_lift\_ \end{cases} \tag{7.36}$$

The second part is a dispersion balanced force F^{Tcorr} in the range y/d < 1*portee_disp_ obtained from the dispersion force by replacing $\underline{\nabla}\alpha_g$ by :

$$\underline{\nabla}\alpha_g = \alpha_g\frac{1}{y}\frac{d_b * portee\_disp\_ - 2y}{d_b * portee\_disp\_ - y}\overrightarrow{n} \tag{7.37}$$

The last part aims to cancel the BIF contribution $0.5d$ away from the wall. Test cases are available in TrioCFD/CoolProp and TrioCFD/Gabillet.

7.3.6 The Tchen force

Warning : This force is a good example of implementation but we discourage its use.
The general expression of the Tchen force [**Tchen1947**] is :

$$\overrightarrow{F_{l\to g}} = \alpha_g \rho_l \frac{\partial \overrightarrow{u_l}}{\partial t} \tag{7.38}$$

The force is implemented in :

```
void Source_Force_Tchen_base::set_param(Param& param)
```

It is discritized in right side of the equation `secmem` as :

$$\alpha_g^n \rho_l^n \frac{u_l^n - u^{n-1}}{\Delta t} \tag{7.39}$$

The local void fraction

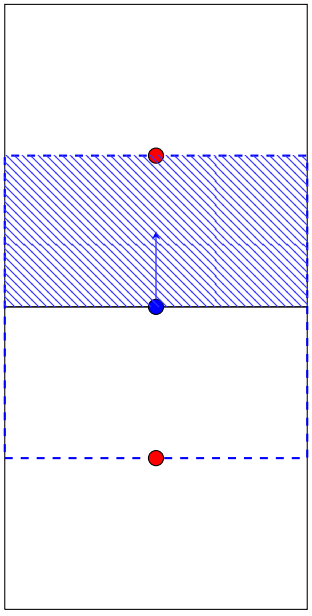


Figure 7.2: Volume management due to velocity located in faces. Black squares represents a mesh element. The blue dashed rectangle is `domaine.volumes_entrelaces()` and the blue hached part is `domaine.volumes_entrelaces_dir()`.

There are 3 types of thermal fluxes available in TrioCFD multiphase:

- The interfacial heat flux $h_k(\alpha, T, P)$
- The wall heat flux $q_{pk}(\alpha, T, P)$
- The phase change flux $G(\alpha, T, P)$

The computation of condensation and evaporation is done in Source_Flux_interfacial_base :

```
void Source_Flux_interfacial_base::set_param(Param& param)
{
  const Pb_Multiphase& pbm = ref_cast(Pb_Multiphase, equation().probleme());
  if (!pbm.has_correlation("flux_interfacial"))
    Process::exit(que_suis_je() + "␣:␣a␣flux_interfacial␣correlation␣must␣be␣
       defined␣in␣the␣global␣correlations␣{␣}␣block!");
```

```
  correlation_ = pbm.get_correlation("flux_interfacial");

  dv_min = ref_cast(Flux_interfacial_base, correlation_->valeur()).dv_min();

  return is;
}
```

With `n_lim` $= \begin{cases} -1, \text{ if G won't make the phase evanescent} \\ \text{number of the evanescent phase, otherwise.} \end{cases}$

If saturation activated then :

$$\Phi = h_g(T_g - T_{sat}) + h_l(T_l - T_{\text{sat}}) + q_p \tag{7.40}$$

$$L = \begin{cases} h_l - H_{ls}, \ if \ \Phi < 0 \\ h_g - H_{gs}, \ \text{otherwise.} \end{cases} \tag{7.41}$$

If no correlation for G or if $|\frac{\Phi}{L}| < |G|$ (limitation by energy conservation) then :

$$G = \frac{\Phi}{L} \tag{7.42}$$

The phase change G is limited by evanescence at G_lim(see evanescence part).

$$h_g = \begin{cases} h_g, \ if \ 0 < G, \\ H_{ls}, \ \text{otherwise.} \end{cases} \tag{7.43}$$

$$h_l = \begin{cases} h_l, \ if \ G < 0, \\ H_{vs}, \ \text{otherwise.} \end{cases} \tag{7.44}$$

$$\frac{dh_g}{dT_g} = \begin{cases} \frac{dh_g}{dT_g}, \ if \ 0 < G, \\ 0, \ \text{otherwise.} \end{cases} \tag{7.45}$$

$$\frac{dh_l}{dT_l} = \begin{cases} \frac{dh_l}{dT_l}, \ if \ G < 0, \\ 0, \ \text{otherwise.} \end{cases} \tag{7.46}$$

$$\frac{dh_g}{dP} = \begin{cases} \frac{dh_g}{dP}, \ if \ 0 < G, \\ \frac{dh_{ls}}{dP}, \ \text{otherwise.} \end{cases} \tag{7.47}$$

$$\frac{dh_l}{dP} = \begin{cases} \frac{dh_l}{dP}, \ if \ G < 0, \\ \frac{dh_{vs}}{dP}, \ \text{otherwise.} \end{cases} \tag{7.48}$$

If there is no evanescence and no phase change model then :

$$\frac{d\Phi}{dP} = \frac{dh_g}{dP}(T_g - T_{sat}) + \frac{dh_l}{dP}(T_l - T_{sat}) - \frac{dT_sat}{dP}(h_g - h_l) + \frac{dq_p}{dP}\frac{1}{vol} \tag{7.49}$$

$$\frac{dG}{dP} = \frac{\frac{d\Phi}{dP} - G\frac{dL}{dP}}{L} \tag{7.50}$$

$$\frac{dG}{dT} = \frac{1}{L}\left(\frac{dh_g}{dT}(T_g - T_{sat}) + \frac{dh_l}{dT}(T_l - T_{sat}) + \frac{dq_p}{dT} + \begin{cases} h_g - G\frac{dL}{dT_g}, \ if \ in \ gas \ phase \\ h_l - G\frac{dL}{dT_l}, \ \text{otherwise} \end{cases}\right) \tag{7.51}$$

$$\frac{dG}{d\alpha} = \frac{1}{L}\left(\frac{dh_g}{d\alpha}(T_g - T_{sat} + \frac{dh_l}{d\alpha}(T_l - T_{sat}) + \frac{dq_p}{d\alpha})\right) \tag{7.52}$$

These fluxes are then distributed to the following equations :

- Mass equation as source/sink
- Energy equation as heat transfer
- Interfacial area concentration as condensation/nucleation (cf equivalent diameter section)

The model is implemented as follows :

- In the energy equation :

$$h_m = \frac{1}{\frac{1}{h_g} + \frac{1}{h_l}} \tag{7.53}$$

$$secmem \mathrel{-}= h_m(T_g - T_l) \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ \text{otherwise.} \end{cases} . \tag{7.54}$$

$$M_T \mathrel{-}= h_m \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} \times \begin{cases} 1, \ regarding\ the\ same\ phase, \\ -1, \ otherwise. \end{cases} . \tag{7.55}$$

- If saturation is activated, then in the mass equation we get :

$$\texttt{secmem} \mathrel{-}= G \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} . \tag{7.56}$$

if there is no evanescence then :

$$M_\alpha \mathrel{+}= \frac{dG}{d\alpha} \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} . \tag{7.57}$$

$$M_T \mathrel{+}= \frac{dG}{dT} \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} . \tag{7.58}$$

$$M_P \mathrel{+}= \frac{dG}{dP} \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} . \tag{7.59}$$

- If saturation is activated, then in the energy equation we get :

$$\texttt{secmem} \mathrel{-}= \left(\texttt{signflux} \times h_c(T_c - T_{sat}) + Gh_c \right) \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} + \begin{cases} q_p, \ if\ not\ c, \\ 0, \ otherwise. \end{cases} . \tag{7.60}$$

$$M_\alpha \mathrel{+}= \left(\texttt{signflux} \times \frac{dh_c}{d\alpha}(T_c - T_{sat}) + h_c \frac{dG}{d\alpha} \begin{cases} 0, if\ evanescent, \\ 1, \ otherwise. \end{cases} \right) \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} - \begin{cases} \frac{dq_p}{d\alpha}, \ if\ not\ c, \\ 0, \ otherwise. \end{cases} . \tag{7.61}$$

$$M_T \mathrel{+}= \left(\texttt{signflux} \times \left(\frac{dh_c}{dT}(T_c - T_{sat}) + h_c \begin{cases} 1, if\ n_c, \\ 0, \ otherwise. \end{cases} \right) + h_c \frac{dG}{dT} \begin{cases} 0, if\ evanescent, \\ 1, \ otherwise. \end{cases} + G \frac{dh_c}{dT} \begin{cases} \frac{dq_p}{d\alpha}, \ if\ n_c, \\ 0, \ otherwise. \end{cases} \right) \times \begin{cases} -1, \ for\ the\ liquid, \\ 1, \ otherwise. \end{cases} - \begin{cases} \frac{dq_p}{dT}, \ if\ not\ c, \\ 0, \ otherwise. \end{cases} . \tag{7.62}$$

$$M_P \mathrel{+}= \left(\texttt{signflux} \times \left(\frac{dh_c}{dP}(T_c - T_{sat}) + h_c \frac{dT_{sat}}{dP} \right) + h_c \frac{dG}{dP} \begin{cases} 0, if\ evanescent, \\ 1,\ otherwise. \end{cases} + G \frac{dh_c}{dP} \right) \times \begin{cases} -1,\ for\ the\ liquid, \\ 1,\ otherwise. \end{cases} - \begin{cases} \frac{dq_p}{d\alpha},\ if\ not\ c, \\ 0,\ otherwise. \end{cases} . \tag{7.63}$$

With c the minority phase side to respect the energy conservation in case of evanescence.

7.3.7 Interfacial heat flux

The general expression of the interfacial heat flux is:

$$\phi_{kl} = h_{kl}(T_k - T_l) \tag{7.64}$$

The model is implemented in:

```
void Flux_interfacial_base::set_param(Param& param)
```

The available input parameters are :

```
  double dh; // Hydraulic diameter
  const double *alpha; // Void fraction
  const double *T; // Temperature
  const double *T_passe;// Previous time temperature
  double p; // Pressure
  const double *nv; // Norme of relative velocity
  const double *lambda; // Thermal conductivity
  const double *mu; // Viscosity
  const double *rho; // Density
  const double *Cp; // Calorific capacity
  const double *Lvap; // Latent heat
  const double *dP_Lvap;// Phase change latent heat
  const double *h; // Enthalpy
  const double *dP_h; // Enthalpy derivative regarding pressure
  const double *dT_h; // Enthalpy derivative regarding temperature
  const double *d_bulles;//Bubble diameter
  const double *k_turb; // Turbulent kinetic energy
  const double *nut; // Turbulent viscosity
  const double *sigma; // Superficial tension
  DoubleTab v; // Velocity
  int e; // Element index
```

The interfacial heat flux operator must fill hi tab so that :

- `hi`$(k1, k2)$ and `hi`$(k2, k1)$ exchange coefficients
- `dT_hi`$(k1, k2, n)$ and `dT_hi`$(k2, k1, n)$ Exchange coefficient derivative regarding the temperature
- `da_hi`$(k1, k2, n)$ and `da_hi`$(k2, k1, n)$ Exchange coefficient derivative regarding void fraction of phase n
- `dp_hi`$(k1, k2)$ and `dp_hi`$(k2, k1)$ Exchange coefficient derivative regarding pressure

Model	Used	Validated	Test case
Constant	✓	✓(100%)	TrioCFD/CoolProp, TrioCFD/Gabillet, TrioCFD/Canal axi two-phase, Trust/Canal bouillant two-phase, Trust/Canal bouillant drift, Trust/Comparaison lois eau
Chen-Mayinger	✓	✗(0%)	
Kim-Park	✓	✗(0%)	
Ranz-Marshall	✓	✗(0%)	
Wolfert	✓	✗(0%)	
Wolfert compsant	✓	✗(0%)	
Zeitoun	✓	✗(0%)	

Table 7.5: Availability of drift models in Trio_CMFD.

Constant

The model is implemented in :

```
void Flux_interfacial_Coef_Constant::set_param(Param& param)
{
param.ajouter(pbm.nom_phase(n), &h_phase(n), Param::REQUIRED);
}
```

Default values. The model implemented is :

$$\texttt{hi}(k,l) = \texttt{h\_phase}(k); \tag{7.65}$$

Chen and Mayinger

The model is implemented in :

```
void Flux_interfacial_Chen_Mayinger::set_param(Param& param)
```

Default values. The model implemented is :

$$Nu = 0.185 Re_b^{0.7} Pr^{0.5}, \tag{7.66}$$

$$\texttt{hi(n\_l, k)} = Nu \times \frac{\lambda_l}{d_b} \frac{6 \times \max(\alpha_g, 1 \times 10^{-4})}{d_b}, \tag{7.67}$$

$$\texttt{da\_hi(n\_l, k, k)} = \begin{cases} Nu \frac{6\lambda_l}{d_b^2}, \ if \ \alpha_g > 1e-4, \\ 0, \ otherwise \end{cases} \tag{7.68}$$

$$\texttt{hi(k, n\_l)} = 1e8, \tag{7.69}$$

with

- $Re_b = \frac{\rho_l d_b (u_g - u_l)}{\mu_l}$
- $Pr \frac{\mu_l C p_l}{\lambda_l}$

Kim and park

The model is also described in.
The model is implemented in :

```
void Flux_interfacial_Kim_Park::set_param(Param& param)
```

Default values. The model implemented is :

$$Nu = 0.2575 Re_b^{0.7} Pr - 0.4564 Ja^{-0.2043} \tag{7.70}$$

$$\texttt{hi(n\_l, k)} = Nu \times \frac{\lambda_l}{d_b} \frac{6 max(\alpha_g, 1e-4)}{d_b}, \tag{7.71}$$

$$\texttt{da\_hi(n\_l, k, k)} = \begin{cases} Nu \frac{6\lambda_l}{d_b^2}, \ if \ \alpha_g > 1e-4, \\ 0, \ otherwise \end{cases} \tag{7.72}$$

$$\texttt{hi(k, n\_l)} = 1e8, \tag{7.73}$$

with

- $Re_b = \frac{\rho_l d_b (u_g - u_l)}{\mu_l}$
- $Pr \frac{\mu_l Cp_l}{\lambda_l}$
- $Ja = \frac{\rho_l Cp_l (T_g - T_l)}{\rho_g L_{vap}}$

Ranz Marshall

The model is also described in.
The model is implemented in :

```
void Flux_interfacial_Ranz_Marshall::set_param(Param& param)
{
param.ajouter("dv_min", &dv_min_);
}
```

Default values $\texttt{a\_min} = 0.01$. The model implemented is :

$$Nu = 2.0 + 0.6 Re_b^{0.5} Pr^{0.3} \tag{7.74}$$

$$\texttt{hi(n\_l, k)} = Nu \times \frac{\lambda_l}{d_b} \frac{6 \max(\alpha_g, \texttt{a\_ min})}{d_b}, \tag{7.75}$$

$$\texttt{da\_hi(n\_l, k, k)} = \begin{cases} Nu \frac{6\lambda_l}{d_b^2}, \ \text{if } \alpha_g > \texttt{a\_ min}, \\ 0, \ \text{otherwise} \end{cases} \tag{7.76}$$

$$\texttt{hi(k, n\_l)} = 1e8, \tag{7.77}$$

with

- $Re_b = \frac{\rho_l d (u_g - u_l)}{\mu_l}$
- $Pr = \frac{\mu_l Cp_l}{\lambda_l}$

Wolfert

The model is also described in.
The model is implemented in :

```
void Flux_interfacial_Wolfert::set_param(Param& param)
{
param.ajouter("Pr_t", &Pr_t_);
}
```

Default values `Pr_t_` = 0.85. The model implemented is :

$$Nu = \frac{12}{\texttt{M\_PI}} Ja + \frac{2}{\sqrt{\pi}}(1 + \nu_t \rho_l \frac{Cp_l}{\lambda_l}) Pe^{0.5}; \tag{7.78}$$

$$\texttt{hi(n\_l, k)} = Nu \times \frac{\lambda_l}{d_b} \frac{6max(\alpha_g, 1e-4)}{d_b}, \tag{7.79}$$

$$\texttt{da\_hi(n\_l, k, k)} = \begin{cases} Nu \frac{6\lambda_l}{d_b^2}, \ if \ \alpha_g > 1e-4, \\ 0, \ otherwise \end{cases} \tag{7.80}$$

with

- $Ja = \frac{\rho_l Cp_l (T_g - T_l)}{\rho_l Lvap}$
- $Pe \frac{\rho_l Cp_l (u_g - u_l) d}{\lambda_l}$
- $\texttt{M\_PI} = \pi$

Wolfert compsant (To be erased)

The model is also described in.
The model is implemented in :

```
void Flux_interfacial_Wolfert_composant::set_param(Param& param)
{
  param.ajouter("Pr_t", &Pr_t_);
  param.ajouter("dv_min", &dv_min_);
}
```

Default values `Pr_t_ = 0.85`. The model implemented is :

$$Nu = \frac{12}{\texttt{M\_PI}} Ja + \frac{2}{\sqrt{\texttt{M\_PI}}}(1 + \lambda_t \rho_l \frac{Cp_l}{\lambda_l}) Pe^{0.5}; \tag{7.81}$$

$$\texttt{hi(n\_l, k)} = Nu \times \frac{\lambda_l}{d} \frac{6max(\alpha_g, 1e-4)}{d}, \tag{7.82}$$

$$\texttt{da\_hi(n\_l, k, k)} = \begin{cases} Nu \frac{6\lambda_l}{d^2}, \ if \ \alpha_g > 1e-4, \\ 0, \ otherwise \end{cases} \tag{7.83}$$

with

- $Ja = \frac{\rho_l Cp_l (T_g - T_l)}{\rho_l Lvap}$
- $Pe = \frac{\rho_l Cp_l (u_g - u_l) d}{\lambda_l}$
- $U_\tau = 0.1987 (u_g - u_l) (\frac{D_h \rho_l (u_g - u_l)}{\mu_l})^{-1/8}$
- $\lambda_t = 0.06 Pr_t U_\tau D_h$
- $\texttt{M\_PI} = \pi$

Zeitoun

The model is also described in.
The model is implemented in :

```
void Flux_interfacial_Zeitoun::set_param(Param& param)
{
param.ajouter("dv_min", &dv_min_);
  if (a_res_ < 1.e-12)
    {
      a_res_ = ref_cast(QDM_Multiphase, pb_->equation(0)).alpha_res();
      a_res_ = std::max(1.e-4, a_res_*10.);
    }
}
```

Default values $\texttt{a\_min\_coeff} = 0.1$, $\texttt{a\_min} = 0.01$, $\texttt{a\_res\_} = -1$.
The model implemented is :

$$Nu = 2.04 Re_b^{0.61} max(\alpha_g, \texttt{a\_min\_coeff})^{0.328} Ja^{-0.308} \tag{7.84}$$

If $(T_g > T_l)$ then :

$$\texttt{hi(n\_l, k)} = Nu * \frac{\lambda_l}{d_b} \frac{6 max(\alpha_g, \texttt{a\_min})}{d_b}, \tag{7.85}$$

$$\texttt{da\_ hi(n\_l, k, k)} = da_{Nu} \frac{6\lambda_l}{d_b^2} max(\alpha_g, \texttt{a\_min}) \begin{cases} Nu \frac{6\lambda_l}{d_b^2}, \ if \ \alpha_g > \texttt{a\_ min}, \\ 0, \ otherwise \end{cases} \tag{7.86}$$

$$\texttt{dp\_hi(n\_l, k)} = dP_{Nu} \frac{6\lambda_l}{d_b^2} max(\alpha_g, \texttt{a\_min}) \tag{7.87}$$

$$\texttt{dT\_hi(n\_l, k,n\_l)} = dT_{lNu} \frac{6\lambda_l}{d_b^2} max(\alpha_g, \texttt{a\_min}) \tag{7.88}$$

$$\texttt{dT\_hi(n\_l, k, k)} = dT_{gNu} \frac{6\lambda_l}{d_b^2} max(\alpha_g, \texttt{a\_min}) \tag{7.89}$$

$$\texttt{hi(k, n\_l)} = 1e8; \tag{7.90}$$

And if $(\alpha_g < \texttt{a\_res\_})$ then :

$$\texttt{hi(n\_l, k)} = 1e8 \times (1 - \frac{alpha_g}{\texttt{a\_res\_}}) + \texttt{hi(n\_l, k)} \frac{alpha_g}{\texttt{a\_res\_}}; \tag{7.91}$$

$$\texttt{da\_hi(n\_l, k, k)} = \frac{-1e8}{\texttt{a\_res\_}} + \texttt{da\_hi(n\_l, k, k)} \frac{\alpha_g}{\texttt{a\_res\_}} + \frac{\texttt{hi(n\_l, k)}}{\texttt{a\_res\_}}; \tag{7.92}$$

$$\texttt{dp\_hi(n\_l, k)} = \texttt{dp\_hi(n\_l, k)} \frac{\alpha_g}{\texttt{a\_res\_}}; \tag{7.93}$$

End. Else (Temperature condition)

$$\texttt{hi(n\_l, k)} = 1e8, \tag{7.94}$$

$$\texttt{da\_hi(n\_l, k, k)} = 0, \tag{7.95}$$

$$\texttt{dp\_hi(n\_l, k)} = 0, \tag{7.96}$$

$$\texttt{dT\_hi(n\_l, k,n\_l)} = 0, \tag{7.97}$$

$$\texttt{dT\_hi(n\_l, k, k)} = 0, \tag{7.98}$$

$$\texttt{hi(k, n\_l)} = 1e8; \tag{7.99}$$

with

- $Re_b = \frac{\rho_l (u_g - u_l) d_b}{\mu_l}$
- $Ja = \frac{max(T_g - T_l, 2.) \rho_l Cp_l}{\rho_l Lvap}$
- $dP_{Ja} = \frac{max(T_g - T_l, 2.) \rho_l Cp_l}{\rho_l} \frac{-dP_{vap}}{Lvap^2}$
- $dT_{gJa} = \begin{cases} \frac{\rho_l Cp_l}{\rho_g Lvap}, \ if \ T_g - T_l > 2. \\ 0, \ otherwise \end{cases}$
- $dT_{lJa} = \begin{cases} \frac{-\rho_l Cp_l}{\rho_g Lvap}, \ if \ T_g - T_l > 2. \\ 0, \ otherwise \end{cases}$
- $da_{Nu} = 2.04 Re_b^{0.61} Ja^{-1.308} \begin{cases} 0.328(\alpha_g^{0.328-1}, \ if \ \alpha_g > \texttt{a\_min\_coeff} \\ 0, \ otherwise \end{cases}$
- $dP_{Nu} = 2.04 Re_b^{0.61} max(\alpha_g, \texttt{a\_min\_coeff})^{0.328} - 0.308 dP_{Ja} Ja^{-1.308}$
- $dT_{gNu} = 2.04 Re_b^{0.61} max(\alpha_g, \texttt{a\_min\_coeff})^{0.328} - 0.308 dT_{Ja} Ja^{-1.308}$
- $dT_{lNu} = 2.04 Re_b^{0.61} max(\alpha_g, \texttt{a\_min\_coeff})^{0.328} - 0.308 dT_{Ja} Ja^{-1.308}$

7.3.8 Wall heat flux

The general expression of the wall heat flux is:

$$q_{pk} \tag{7.100}$$

The model is implemented in :

```
void Flux_parietal_base::set_param(Param& param)
```

The available input parameters are :

```
    int N; // Number of phases
    int f; // face number
    double y; // distance between the face and the center of gravity of the
       cell
    double D_h; // Hydraulic diameter
    double D_ch; // Heated hydraulic diameter
    double p; // Pressure
    double Tp; // Wall temperature
    const double *alpha; // Void fraction
    const double *T; // Temperature
    const double *v; // Velocity norm
    const double *lambda; // Thermal conductivity
    const double *mu; // Viscosity
    const double *rho; // Density
    const double *Cp; // Calorific capacity
    const double *Lvap; // Latent heat
    const double *Sigma; // Surface tension
    const double *Tsat; //Phase change saturated temperature
```

The interfacial heat flux operator must fill qpk and qpi tabs and there derivative so that :

- `qpk` heat flux
- `da_qpk` heat flux derivative regarding void fraction
- `dp_qpk` heat flux derivative regarding pressure
- `dv_qpk` heat flux derivative regarding velocity
- `dTf_qpk` heat flux derivative regarding temperature
- `dTp_qpk` heat flux derivative regarding wall temperature
- `qpi` phase change heat flux
- `da_qpi` phase change heat flux derivative regarding void fraction
- `dp_qpi` phase change heat flux derivative regarding pressure
- `dv_qpi` phase change heat flux derivative regarding velocity
- `dTf_qpi` phase change heat flux derivative regarding temperature
- `dTp_qpi` phase change heat flux derivative regarding wall temperature
- `d_nuc` nucleation diameter

Model	Used	Validated	Test case
Kommajosyula	✓	✗(0%)	
Kurul-Podowski	✓	✓(100%)	TrioCFD/CoolProp

Table 7.6: Availability of drift models in Trio_CMFD.

Kommajosyula (to be erased)

The model is described in **Ravik2020**.
The model is implemented in :

```
void Flux_parietal_Kommajosyula::set_param(Param& param)
{
  param.ajouter("contact_angle_deg",&theta_,Param::REQUIRED);
  param.ajouter("molar_mass",&molar_mass_,Param::REQUIRED);
}
```

Warning : the model was implemented but dropped because we could not fit the original data and so is not validated.

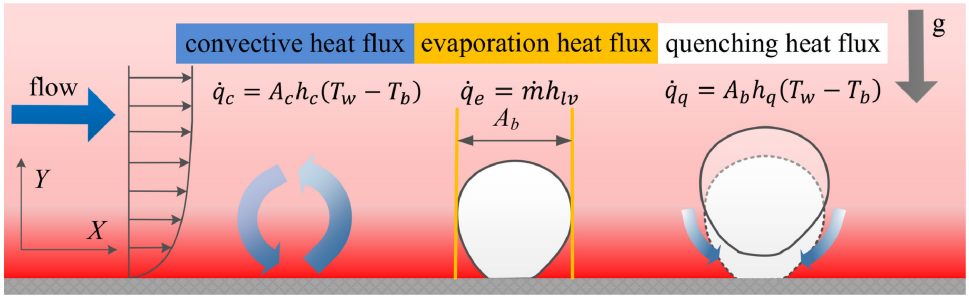


Figure 7.3: Depiction of the wall heat flux partitioning model for subcooled flow boiling from **ZHOU2021121295**.

Kurul Podowski

The model is described in **kurul1991modeling** and depicted in Figure 7.3.

The model is implemented in :

```
void Flux_parietal_Kurul_Podowski::set_param(Param& param)
```

The model implemented is :
1) Correction of single phase heat flux

- qpk $= \text{qpk}_{single\ phase}(1 - A_{bub})$,
- da_qpk $= \text{da\_qpk}_{single\ phase}(1 - A_{bub})$,
- dp_qpk $= \text{dp\_qpk}_{single\ phase}(1 - A_{bub})$,
- dv_qpk=dv_qpk$_{single\ phase}(1 - A_{bub})$,
- dTf_qpk $= \text{dT\_f\_qpk}_{single\ phase}(1 - A_{bub})$,
- dTp_qpk $= \text{dTp\_qpk}_{single\ phase}(1 - A_{bub})$.

2) Add partitioned heat flux

- dTp_qpk += $-\text{qpk}_{single\ phase}(1 - A_{bub})\frac{dA_{bub}}{dT_p} + \frac{dq_{quench}}{dT_p}$,
- qpk += q_{quench},
- dTf_qpk += $\frac{dq_{quench}}{dT_l}$,
- qpi += q_{evap},
- dTp_qpi += $\frac{dq_{evap}}{dT_p}$,

with

- Evaporation flux $q_{evap} = f_{dep}\frac{\pi d_b^3}{6}\rho_g L_{vap} N_{sites}$
- Evaporation flux derivative regarding wall departure $\frac{dq_{evap}}{dT_p} = (\frac{df_{dep}}{dT_p}\frac{\pi d_b^3}{6}N_{sites} + f_{dep}\frac{3\pi d^2}{6}\frac{dd_b}{dT_p}N_{sites} + f_{dep}\frac{\pi d_b^3}{6}\frac{dN_{sites}}{dT_p})\rho_g L_{vap}$.
- Quenching flux $q_{quench} = A_{bub}\sqrt{f_{dep}}\frac{2\lambda_l(T_p - T_l)}{\sqrt{\frac{\pi\lambda_l}{\rho_l C p_l}}}$.

- Quenching flux derivative regarding liquid temperature $\frac{dq_{quench}}{dT_l} = A_{bub}\sqrt{f_{dep}}\frac{-2\lambda_l}{\sqrt{\frac{\pi\lambda_l}{\rho_l C p_l}}}$.
- Quenching flux derivative regarding wall temperature $\frac{dq_{quench}}{dT_p} = \frac{dA_{bub}}{dT_p}\sqrt{f_{dep}}\frac{-2\lambda_l}{\sqrt{\frac{\pi\lambda_l}{\rho_l C p_l}}} + A_{bub}\sqrt{f_{dep}}\frac{2\lambda_l}{\sqrt{\frac{\pi\lambda_l}{\rho_l C p_l}}} - A_{bub}\frac{1}{2}\frac{df_{dep}}{dT_p}\frac{1}{\sqrt{f_{dep}}}\frac{2\lambda_l(T_p-T_l)}{\sqrt{\frac{\pi\lambda_l}{\rho_l C p_l}}}$.
- Number of evaporation sites $N_{sites} = (210 \times (T_p - T_{sat}))^{1.8}$.
- Number of evaporation sites $N_{sites} = (210 \times (T_p - T_{sat}))^{1.8}$.
- Number of evaporation sites derivative regarding wall temperature $\frac{dN_{sites}}{dT_p} = 210 \times 1.8(210.(T_p - T_{sat}))^{0.8}$.
- Wall bubble diameter $d_b = 0.0001(T_p - T_{sat}) + 0.0014$.
- Wall bubble diameter derivative regrading wall temperature $\frac{dd_b}{dT_p} = 0.0001$.
- Wall bubble total area $A_{bub} = min(1., \frac{\pi N_{sites} d_b^2}{4})$.
- Wall bubble total area derivative regarding wall temperature $\frac{dA_{bub}}{dT_p} = \frac{\pi d_b^2}{4}\frac{dN_{sites}}{dT_p} + \frac{\pi d_b N_{sites}}{2}\frac{dd_b}{dT_p}$, if $A_{bubbles} \neq 1., 0$, otherwise.
- Departure frequency $f_dep = \sqrt{\frac{4}{3}\frac{9.81(\rho_l-\rho_g)}{\rho_l}}d_b^{-0.5}$.
- Departure frequency derivative regarding wall temperature $\frac{df_dep}{dT_p} = -0.5\frac{dd_b}{dT_p}d^{-1.5}\sqrt{\frac{4}{3}\frac{9.81(\rho_l-rho_g)}{\rho_l}}$

7.3.9 Phase change

The general expression of the phase change mass flux is :

$$G(\alpha, p, T) \tag{7.101}$$

It needs to be considered when there is a kinetic limit of gas for the phase change, for liquid metals, for example, but it does not apply to water.
The model is implemented in :

```
void Changement_phase_base::set_param(Param& param)
```

The available input parameters are :

```
double D_h; // Hydraulic diameter
double p; // Pressure
const double *alpha; // Void fraction
const double *T; // Temperature
const double *lambda; // Thermal conductivity
const double *mu; // Viscosity
const double *rho; // Density
const double *Cp; // Calorific capacity
const double *Lvap; // Latent heat
const double *Tsat; //Phase change saturated temperature
```

The phase change mass flux operator must fill dT_G, da_G, dp_G tabs and there derivative so that :

- `G` mass flux
- `dT_G` mass flux derivative regarding temperature
- `dp_G` mass flux derivative regarding pressure
- `da_G` mass flux derivative regarding void fraction

Silver Simpson

The model is also described in Silver and Simpson (1949, not found).

The model is implemented as:

```
void Changement_phase_Silver_Simpson::set_param(Param& param)
{
  param.ajouter("lambda_e", &lambda_ec[0]); multiplicative factor for
     evaporation
  param.ajouter("lambda_c", &lambda_ec[1]); // multiplicative factor for
     condensation
  param.ajouter("alpha_min", &alpha_min); // minimal void fraction to activate
      phase change
  param.ajouter("M", &M, Param::REQUIRED); // molar mass of steam
}
```

Default values : `lambda_ec[2]` $= 1, 1$, `M` $= -100.$, `alpha_min` $= 0.1$.
The model implemented is :

$$\texttt{dT\_G}_g = \texttt{fac} \times \texttt{var\_a} \times \frac{dT_{Psat}(T_g) - 0.5 \times \frac{Psat(T_g)}{T_g+T_0}}{\sqrt{T_g + T_0}} \tag{7.102}$$

$$\texttt{dT\_G}_l = \texttt{fac} \times \texttt{var\_a} \times 0.5 \times P \times (T_l + T_0)^{-1.5} \tag{7.103}$$

$$\texttt{da\_G}_g = \begin{cases} \texttt{fac} \times \texttt{var\_T} \times \texttt{var\_al}, if\ \alpha_g > \text{alpha\_min}, \\ 0,\ otherwise. \end{cases} \tag{7.104}$$

$$\texttt{da\_G}_l = \begin{cases} \texttt{fac} \times \texttt{var\_T} \times \texttt{var\_ak} \times 1.5 \times \sqrt{\alpha_l}, if\ \alpha_l > \texttt{alpha\_min}, \\ 0,\ \text{otherwise}. \end{cases} \tag{7.105}$$

$$\texttt{dp\_G} = -\texttt{fac} \times \frac{\texttt{var\_a}}{\sqrt{T_l + T_0}} \tag{7.106}$$

$$\texttt{G} = \texttt{fac} \times \ \texttt{var\_ak} \times \texttt{var\_al} \times \texttt{var\_T}; \tag{7.107}$$

with

- $T_0 = 273.15$,
- $\texttt{var\_ak} = \max(\alpha_g, \texttt{alpha\_min})$,
- $\texttt{var\_al} = (\max(\alpha_l, \texttt{alpha\_min}))^{1.5}$,
- $\texttt{var\_a} = \texttt{var\_ak} \times \texttt{var\_al}$,
- $\texttt{var\_T} = \frac{Psat(T_g)}{\sqrt{T_g+T_0}} - \frac{P}{\sqrt{T_l+T_0}}$,
- $\texttt{fac} = \lambda_{ec}[var_T < 0] \frac{4}{D_h} \sqrt{\frac{\texttt{M}}{2\texttt{M\_PI}8.314}}$

7.4 Analytical terms as source terms

This section describes some analytical terms that are implemented as source terms in matrix.

7.4.1 Pressure term in energy equation

This term arises from the averaging of the pressure work term due to the transport of internal energy instead of enthalpy. It represents the pressure work associated with changes in the distribution of void fraction. Its expression is :

$$- P \left(\frac{\partial \alpha_k}{\partial t} + \nabla \cdot (\alpha_k v_k) \right) \tag{7.108}$$

It is implemented in:

```
void Source_Travail_pression_Elem_base::set_param(Param& param)
```

7.4.2 Gravity

Gravity is treated as a source term in `Pb_multiphase` and can't be as other problems from TrioCFD and TRUST.
The common way to add gravity is to add in the momentum equation a source. For example, with 2 phases :

```
source_qdm Champ_Fonc_xyz dom 6 0 0 0 0 -9.81 -9.81
```

One must use `Gravite_Multiphase` to get gravity in momentum equation when using drift correlation. The model is implemented as :

```
void Gravite_Multiphase::set_param(Param& param)
{
  Noms noms(D), unites(D);
  noms[0] = "gravite";
  unites[0] = "m/s^2";
  Motcle typeChamp = "champ_elem" ;
  const Domaine_dis& z = ref_cast(Domaine_dis, pb.domaine_dis());
  dis.discretiser_champ(typeChamp, z.valeur(), scalaire, noms , unites, D, 0,
     gravite_);
}
```

7.5 Injected sources of mass

It is necessary in order to simulate injectors of non-condensable bubbles, otherwise it boils.

7.5.1 Injection of mass

The model is implemented in:

```
void Source_injection_masse_base::set_param(Param& param)
{
  Cerr << "Lecture␣du␣Champ␣de␣masse␣injectee" << finl;
```

```
  Champ_Don flux_masse_lu_;
  Motcle type;
  is >> type;

  flux_masse_lu_.typer(type);
  Champ_Don_base& ch_flux_masse_lu_ = ref_cast(Champ_Don_base,flux_masse_lu_.
     valeur());
  is >> ch_flux_masse_lu_;
  const int nb_comp = ch_flux_masse_lu_.nb_comp();

  equation().probleme().discretisation().discretiser_champ("champ_elem",
     equation().domaine_dis(), "pp", "1",nb_comp,0., flux_masse_);
  flux_masse_lu_->fixer_nb_comp(nb_comp);
  if (ch_flux_masse_lu_.le_nom()=="anonyme") ch_flux_masse_lu_.nommer("
     Flux_masse_injectee");

  for (int n = 0; n < nb_comp; n++) flux_masse_lu_->fixer_nom_compo(n,
     ch_flux_masse_lu_.le_nom() + (nb_comp > 1 ? Nom(n) :""));
  for (int n = 0; n < nb_comp; n++) flux_masse_->fixer_nom_compo(n,
     ch_flux_masse_lu_.le_nom() + (nb_comp > 1 ? Nom(n) :""));
  equation().discretisation().nommer_completer_champ_physique(equation().
     domaine_dis(),ch_flux_masse_lu_.le_nom(),"1/s",flux_masse_lu_,equation().
     probleme());
  equation().discretisation().nommer_completer_champ_physique(equation().
     domaine_dis(),ch_flux_masse_lu_.le_nom(),"1/s",flux_masse_,equation().
     probleme());
  flux_masse_.valeur().valeurs() = 0;
  flux_masse_.valeur().affecter(flux_masse_lu_);

  const Pb_Multiphase& pb = ref_cast(Pb_Multiphase, equation().probleme());
  int N = pb.nb_phases();
  if (N != flux_masse_.valeurs().line_size()) Process::exit(que_suis_je() + "␣:
     ␣you␣must␣input␣as␣many␣fluxes␣as␣there␣are␣phases␣!!");

  return is;
}
```

In the mass equation:

$$\texttt{secmem} \mathrel{+}= \rho_k f_{inj} \tag{7.109}$$

7.5.2 Momentum correction

When a fluid flow is injected through a wall with zero momentum via a Neumann boundary condition on the mass equation, it is necessary to correct the momentum equation.

The model is implemented in:

```
void Injection_QDM_nulle_PolyMAC_P0::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("beta", &beta_);
  param.lire_avec_accolades_depuis(is);
```

```
}
```

Default values : `beta_` = 1.
Case 1: when bubbles are injected at the wall if not face from boundary

$$\texttt{f\_a\_masse} = \rho_g U_{inj} \tag{7.110}$$
$$\texttt{f\_a} = \rho_g U_{inj} \tag{7.111}$$

corr-ajouter_inj

$$\texttt{secmem} \mathrel{-}= \texttt{surface} \times \texttt{f\_a\_masse} \times U^n \times \texttt{beta\_} \tag{7.112}$$
$$M_v \mathrel{+}= \texttt{surface} \times \texttt{f\_a\_masse} \times \texttt{beta\_} \tag{7.113}$$

Case 2: wall boiling if not face from boundary

$$\texttt{G} = \frac{q_p}{L_{vap}} \tag{7.114}$$
$$\texttt{f\_a\_masse} \mathrel{-}= \frac{\texttt{G}}{\texttt{surface}} \times \begin{cases} \frac{1}{\rho_g}, \text{ if in gas,} \\ -\frac{1}{\rho_l}, \text{ if in liquid.} \end{cases} \tag{7.115}$$
$$\texttt{f\_a} \mathrel{-}= \frac{\texttt{G}}{\texttt{surface}} \times \begin{cases} 1, \text{ if in gas,} \\ -1, \text{ if in liquid.} \end{cases} \tag{7.116}$$

corr-ajouter_inj

$$\texttt{secmem} \mathrel{-}= \texttt{surface} \times \texttt{f\_a\_masse} \times U^n \times \texttt{beta\_} \tag{7.117}$$
$$M_v \mathrel{+}= \texttt{surface} \times \texttt{f\_a\_masse} \times \texttt{beta\_} \tag{7.118}$$

7.6 Equivalent diameter for dispersed phase

A fundamental aspect of modeling the interaction between liquid and gas phases lies in precisely predicting the concentration of interfaces. Some models rely on either the interfacial area concentration (ai) or an equivalent diameter (Dsm). In this context, the dispersed fluid is depicted as a collection of bubbles with diverse diameters, which essentially acts as a topological description for the fluid stage. This dispersed fluid is distinguished by two interlinked attributes: a distribution of bubble diameters and the concentration of interfaces. We can define the Sauter-Mean Diameter (Dsm) of the distribution (f_d) of sizes D as :

$$Dsm = \frac{\int f_d D^3 dD}{\int f_d D^2 dD} \tag{7.119}$$

The relationship between the void fraction (α)$and the Sauter-Mean Diameter(Dsm) hinges on calculating the$
i=πD^2). This relationship can be expressed as follows :

$$Dsm = \frac{\int f_d D^3 dD}{\int f_d D^2 dD} = 6\frac{\int f_d V dv}{\int f_d A_i dV} = 6\frac{\alpha}{ai} \tag{7.120}$$

Finally, to have a correct representation of the dispersed phase, it is sufficient to impose an equivalent diameter, but it is also possible to add a transport equation for the interfacial area concentration.

7.6.1 User defined diameter

Uniform diameter

It is possible to simply impose a constant diameter at every point in space. The model is implemented in `Diametre_bulles_constant` :

```
void Diametre_bulles_constant::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("diametre", &d_bulle_, Param::REQUIRED);
  param.lire_avec_accolades_depuis(is);

  Pb_Multiphase& pb = ref_cast(Pb_Multiphase, pb_.valeur());
  int N = pb.nb_phases();
  const Discret_Thyd& dis=ref_cast(Discret_Thyd,pb.discretisation());
  Noms noms(N), unites(N);
  noms[0] = "diametre_bulles";
  unites[0] = "m";
  Motcle typeChamp = "champ_elem" ;
  const Domaine_dis& z = ref_cast(Domaine_dis, pb.domaine_dis());
  dis.discretiser_champ(typeChamp, z.valeur(), scalaire, noms , unites, N, 0,
     diametres_);

  champs_compris_.ajoute_champ(diametres_);

  for (int n = 0; n < pb.nb_phases(); n++) //recherche de n_l, n_g : phase {
     liquide,gaz}_continu en priorite
    if (pb.nom_phase(n).debute_par("liquide") && (n_l < 0 || pb.nom_phase(n).
       finit_par("continu"))) n_l = n;
  if (n_l < 0) Process::exit(que_suis_je() + "␣:␣liquid␣phase␣not␣found!");

  DoubleTab& tab_diametres = diametres_->valeurs();
  for (int i = 0 ; i < tab_diametres.dimension_tot(0) ; i++)
    for (int n = 0 ; n <N ; n++)
      if (n!=n_l) tab_diametres(i, n) = d_bulle_;

  return is;
}
```

Default value : `d_ bulle_` $= -100$.

Non-uniform diameter

It is possible to impose a user-defined diameter that varies spatially through the TRUST fields. The model is implemented in `Diametre_bulles_champ` :

```
void Diametre_bulles_champ::set_param(Param& param)
{
  Champ_Don diametres_don_;
  is >> diametres_don_;

  Pb_Multiphase& pb = ref_cast(Pb_Multiphase, pb_.valeur());
```

```
  int N = pb.nb_phases();
  const Discret_Thyd& dis=ref_cast(Discret_Thyd,pb.discretisation());
  Noms noms(N), unites(N);
  noms[0] = "diametre_bulles";
  unites[0] = "m";
  Motcle typeChamp = "champ_elem" ;
  const Domaine_dis& z = ref_cast(Domaine_dis, pb.domaine_dis());
  dis.discretiser_champ(typeChamp, z.valeur(), scalaire, noms , unites, N, 0,
     diametres_);
  champs_compris_.ajoute_champ(diametres_);
  diametres_->affecter(diametres_don_.valeur());
  diametres_.valeurs().echange_espace_virtuel();
  return is;
}
```

Default value : `d_ bulle_` $= -100$.

7.6.2 Interfacial area concentration with 1 group (incoming)

The general equation

Two separated size-group methods are popular for the prediction of interfacial area concentration. One based on having an arbitrary number of groups to reproduce a distribution, referred as MUSIG or i-MUSIG [**Wang2005a**, **Das2010a**, **Liao2011**] and the other reproducing the distribution thanks to the Mean Sauter diameter referred as IATE. The generalized Interfacial Area Transport Equation (IATE) developed by Kocamustafaogullari and Ishii [**Kocamustafaogullari**, **Kocamustafaogullari1994b**] The general expression for adiabatic flows with $\psi_j^{internal}$ a source term and $\psi_j^{intergroup}$ an intergroup term is then :

$$\frac{\partial a_i}{\partial t} + \underbrace{\nabla.(\mathbf{U}a_i)}_{\text{convection}} = \underbrace{\frac{2}{3}\frac{a_i}{\alpha}\frac{D\alpha}{Dt}}_{\text{source of volume change}} + \underbrace{\sum_j \psi_j^{intergroup}}_{\text{intergroup sources}} + \underbrace{\sum_j \psi_j^{internal}}_{\text{intragroup sources}} . \tag{7.121}$$

The terms in green need new modeling linked to coalescence and break-up. Let's remind that :

$$\frac{D\alpha\rho_g}{Dt} = \alpha\frac{d\rho_g}{dt} + \rho_g\frac{D\alpha}{Dt} = \Gamma_{transfer} + \Gamma_{nucleation} \tag{7.122}$$

Then we can substitute :

$$\frac{D\alpha}{Dt} = \frac{1}{\rho_g}(\Gamma_{transfer} + \Gamma_{nucleation} - \alpha\frac{d\rho_g}{dt}) \tag{7.123}$$

It gives the following equation :

$$\begin{aligned}\frac{\partial a_i}{\partial t} + \underbrace{\nabla.(\mathbf{U}a_i)}_{\text{convection}} = &\underbrace{-\frac{2}{3}\frac{a_i}{\rho_g}\frac{d\rho_g}{dt}}_{\text{Density change}} + \underbrace{\frac{2}{3}\frac{a_i}{\alpha}\frac{\Gamma_{transfer}}{\rho_g}}_{\text{Condensation}} + \underbrace{\frac{2}{3}\frac{a_i}{\alpha}\frac{\Gamma_{nucleation}}{\rho_g}}_{\text{Nucleation}} + \\ &\underbrace{\sum_j \psi_j^{intergroup}}_{\text{intergroup sources}} + \underbrace{\sum_j \psi_j^{internal}}_{\text{intragroup sources}} .\end{aligned} \tag{7.124}$$

The Density change model is :

$$-\frac{2}{3}\frac{a_i}{\rho_g}\frac{d\rho_g}{dt} \tag{7.125}$$

The Density change is implemented in `Variation_rho_Elem_PolyMAC_P0` :

```
Void Variation_rho::set_param(Param& param)
```

It fills the following matrices :

$$M_{ai} \mathrel{-}= \frac{2}{3}\frac{1-\frac{\rho_g^{n-1}}{\rho_g^n}}{\Delta t} \tag{7.126}$$

$$M_T \mathrel{-}= \frac{2}{3}\frac{a_i^n}{\Delta t}\frac{\rho_g^{n-1}}{(\rho_g^n)^2}(-\texttt{dT\_rho}) \tag{7.127}$$

$$M_P \mathrel{-}= \frac{2}{3}\frac{a_i^n}{\Delta t}\frac{\rho_g^{n-1}}{(\rho_g^n)^2}(-\texttt{dP\_rho}) \tag{7.128}$$

$$\texttt{secmem} \mathrel{+}= \frac{2}{3}a_i^n\frac{1-\frac{\rho_g^{n-1}}{\rho_g^n}}{\Delta t} \tag{7.129}$$

For example, the chain rule for the temperature gives :

$$\frac{d(\frac{1}{\rho_g}\frac{d\rho_g}{dt})}{dT} = \frac{d\rho_g}{dT}\left(\frac{d}{d\rho_g}(\frac{1}{\rho_g})\frac{d\rho_g}{dt} + \frac{1}{\rho_g}\frac{d}{d\rho_g}(\frac{d\rho_g}{dt})\right) = \frac{d\rho_g}{dT}\left(-(\frac{1}{\rho_g^2})\frac{d\rho_g}{dt} + \frac{1}{\rho_g}\frac{d}{d\rho_g}(\frac{d\rho_g}{dt})\right) \tag{7.130}$$

Regarding the discreet form, it gives :

$$\frac{d\rho_g}{dT}\left(-(\frac{1}{(\rho_g^n)^2})\frac{\rho_g^n-\rho_g^{n-1}}{\Delta t} + \frac{1}{\rho_g^n\Delta t}\right) = \frac{1}{\Delta t}\frac{d\rho_g}{dT}\left(\frac{\rho_g^{n-1}}{(\rho_g^n)^2}\right) \tag{7.131}$$

The Condensation model is :

$$\frac{2}{3}\frac{a_i}{\alpha\rho_g}G, \tag{7.132}$$

with G given by a correlation.
The Condensation term is implemented in `Source_Flux_interfacial_base` as :

$$\texttt{secmem} \mathrel{+}= \frac{2}{3}\frac{a_i^n}{\alpha^n\rho_g^n}G^n \tag{7.133}$$

If the condensation is not making the phase evanescent then :

$$M_\alpha \mathrel{-}= \frac{-2}{3}\frac{a_i^n}{(\alpha^n)^2}\rho_g^n G^n \tag{7.134}$$

$$M_T \mathrel{-}= \frac{-2}{3}\frac{a_i^n}{\alpha^n(\rho_g^n)^2}G^n\frac{d\rho_l}{dT} + \frac{2}{3}\frac{a_i^n}{\alpha^n\rho_g}\frac{dG}{dT} \tag{7.135}$$

$$M_P \mathrel{-}= \frac{-2}{3}\frac{a_i^n}{\alpha^n(\rho_g^n)^2}G^n\frac{d\rho_g}{dP} + \frac{2}{3}\frac{a_i^n}{\alpha^n\rho_g}\frac{dG}{dP} \tag{7.136}$$

$$M_{a_i} \mathrel{-}= \frac{2}{3}\frac{1}{\alpha^n\rho_g^n}G^n \tag{7.137}$$

For the other source terms, refer to the models.

The Yao Morel model

The model is described in **YAO2004307**.

The equation is:

$$\frac{\partial a_i}{\partial t} + \underbrace{\nabla.(\mathbf{U}a_i)}_{\text{convection}} = \underbrace{\frac{2}{3}\frac{a_i}{\alpha}\left(\frac{\Gamma_{condensation}}{\rho_g} - \frac{\alpha_g}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{source of volume change}} + \underbrace{\pi d_{dep}^2 \Phi_N}_{\text{Nucleation}} + \underbrace{\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2 \Phi_{coal}}_{\text{Coalescence}} + \underbrace{\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2 \Phi_{breakup}}_{\text{Break-up}}. \tag{7.138}$$

The Coalescence model is :

$$\begin{aligned}\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2\Phi_{Coal} &= -\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2(\varepsilon d_b)^{1/3}\cdot\frac{\alpha^2}{d_b^4}\cdot K_{c1}\cdot\frac{1}{g(\alpha)+K_{c2}\alpha\sqrt{We/We_{cr}}}\cdot\exp\left(-K_{c3}\sqrt{\frac{We}{We_{cr}}}\right)\\ &= \frac{\pi}{3\times 6^{5/3}}\alpha^{1/3}a_i^{5/3}\varepsilon^{1/3}\cdot K_{c1}\frac{-1}{g(\alpha)+K_{c2}\alpha\sqrt{We/We_{cr}}}\cdot\exp\left(-K_{c3}\sqrt{\frac{We}{We_{cr}}}\right),\end{aligned} \tag{7.139}$$

with $K_{c1}=2.86$, $K_{c2}=1.922$, $K_{c3}=1.017$, $We_{cr}=1.24$, $g(\alpha)=\frac{\alpha_{\max}^{1/3}-\alpha^{1/3}}{\alpha_{\max}^{1/3}}$, $\alpha_{\max}=\frac{\pi}{6}$.

The Coalescence model is implemented as $\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2\Phi_{Coal} = f_1(\alpha,ai,k,\varepsilon)f_2(\alpha,ai,k,\varepsilon)$ so that for the sake of simplicity $d(f_1f_2)=f_2df_1$ in Coalescence_bulles_1groupe_PolyMAC_P0 :

```
void Coalescence_bulles_1groupe_PolyMAC_P0::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("beta_k", &beta_k_);
}
```

and Coalescence_bulles_1groupe_Yao_Morel:

```
void Coalescence_bulles_1groupe_Yao_Morel::set_param(Param& param)
```

Default values : `beta_k_` = 0.09, `Kc1` = 2.86, `Kc2` = 1.922, `Kc3` = 1.017, `alpha_max_1_3` = $(\frac{\pi}{6})^{1/3}$, `We_cr` = 1.24.

Warning : The following part describes the matrix filling for the $k-\varepsilon$ model, as an example, but it is currently not implemented since only the $k-\tau$ and $k-\omega$ models are available and are the only ones implemented for the source terms.

For the $k-\varepsilon$ model :

$$M_\alpha \mathrel{-}= \frac{\pi}{3\times 6^{5/3}}\frac{1}{3}(\alpha^n)^{-2/3}(a_i^n)^{5/3}(\varepsilon^n)^{1/3}f_2^n \tag{7.140}$$

$$M_{a_i} \mathrel{-}= \frac{\pi}{3\times 6^{5/3}}\frac{5}{3}(\alpha^n)^{1/3}(a_i^n)^{2/3}(\varepsilon^n)^{1/3}f_2^n \tag{7.141}$$

$$M_k = 0 \tag{7.142}$$

$$M_\varepsilon \mathrel{-}= \frac{\pi}{3\times 6^{5/3}}\frac{1}{3}(\alpha^n)^{-2/3}(a_i^n)^{5/3}(\varepsilon^n)^{-2/3}f_2^n \tag{7.143}$$

$$\text{secmem} \mathrel{+}= \frac{\pi}{3\times 6^{5/3}}(\alpha^n)^{1/3}(a_i^n)^{5/3}(\varepsilon^n)^{1/3}f_2^n \tag{7.144}$$

If the $k-\tau$ turbulence model is used,

$$\varepsilon = \frac{k^2}{\max(k\tau, \texttt{visc\_turb.limiteur()}\nu_l)}\times \texttt{beta\_k\_}. \tag{7.145}$$

If the $k-\omega$ turbulence model is used,

$$\varepsilon = k \times \omega \times \texttt{beta\_k\_}. \tag{7.146}$$

The Break-up model is :

$$\begin{aligned}\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2 \Phi_{\text{breakup}} &= -\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2 (\varepsilon d_b)^{1/3} \cdot \frac{\alpha(1-\alpha)}{d_b^4} \cdot K_{b1} \cdot \frac{1}{1+K_{b2}\alpha_l\sqrt{We/We_{cr}}} \cdot \exp\left(-\frac{We}{We_{cr}}\right) \\ &= \frac{\pi}{3\times 6^{5/3}}\alpha^{-2/3}(1-\alpha)a_i^{5/3}\varepsilon^{1/3} \cdot K_{b1} \cdot \frac{1}{1+K_{b2}(1-\alpha)\sqrt{We/We_{cr}}} \cdot \exp\left(-\frac{We}{We_{cr}}\right),\end{aligned} \tag{7.147}$$

with $K_{b1} = 1.6$, $K_{b2} = 0.42$, $We_{cr} = 1.24$.

The Break-up model is implemented as $\frac{36\pi}{3}\left(\frac{\alpha}{a_i}\right)^2 \Phi_{\text{breakup}} = f_1(\alpha, ai, k, \varepsilon) f_2(\alpha, ai, k, \varepsilon)$ so that for the sake of simplicity $d(f_1 f_2) = f_2 df_1$ in Rupture_bulles_1groupe_PolyMAC_P0 :

```
void Rupture_bulles_1groupe_PolyMAC_P0::set_param(Param& param)
{
  Param param(que_suis_je());
  param.ajouter("beta_k", &beta_k_);
}
```

and Rupture_bulles_1groupe_Yao_Morel:

```
void Rupture_bulles_1groupe_Yao_Morel::set_param(Param& param)
```

Default values: `beta_k_` = 0.09, `Kb1` = 1.6, `Kb2` = 0.42, `We_cr` = 1.24.
Warning : The following part describes the matrix filling for the $k-\varepsilon$ model, as an example, but it is currently not implemented since only the $k-\tau$ and $k-\omega$ models are available and are the only ones implemented for the source terms.
For the $k-\varepsilon$ model :

$$M_\alpha \mathrel{-}= \frac{\pi}{3\times 6^{5/3}}\frac{-2}{3}(\alpha^n)^{-5/3}\alpha_l^n(a_i^n)^{5/3}(\varepsilon^n)^{1/3} f_2^n \tag{7.148}$$

$$M_{\alpha_l} \mathrel{-}= \frac{\pi}{3\times 6^{5/3}}(\alpha^n)^{-2/3}(a_i^n)^{5/3}(\varepsilon^n)^{1/3} f_2^n \tag{7.149}$$

$$M_{a_i} \mathrel{-}= \frac{\pi}{3\times 6^{5/3}}\frac{5}{3}(\alpha^n)^{1/3}\alpha_l^n(a_i^n)^{2/3}(\varepsilon^n)^{1/3} f_2^n \tag{7.150}$$

$$M_k = 0 \tag{7.151}$$

$$M_\varepsilon \mathrel{-}= \frac{\pi}{3\times 6^{5/3}}\frac{1}{3}(\alpha^n)^{-2/3}\alpha_l^n(a_i^n)^{5/3}(\varepsilon^n)^{-2/3} f_2^n \tag{7.152}$$

$$\text{secmem} \mathrel{+}= \frac{\pi}{3\times 6^{5/3}}\frac{1}{3}(\alpha^n)^{-2/3}\alpha_l^n(a_i^n)^{5/3}(\varepsilon^n)^{-2/3} f_2^n \tag{7.153}$$

$$\tag{7.154}$$

If the $k-\tau$ turbulence model is used,

$$\varepsilon = \frac{k^2}{\max(k\tau,\ \texttt{visc\_turb.limiteur()}\nu_l)} \times beta\_k\_. \tag{7.155}$$

If the $k-\omega$ turbulence model is used

$$\varepsilon = k \times \omega \times beta\_k\_. \tag{7.156}$$

The Nucleation model is :

$$\pi d_{dep}^2 \Phi_N = \pi d_{dep}^2 \frac{\Phi_e}{L_{\text{vap}} \rho_g \frac{\pi}{6} d_{\text{dep}}^3} = 6 \frac{\Phi_{\text{nucleation}}}{\max(d_{\text{nuc}}, 10^{-8}) \rho_g L_{\text{vap}}} \tag{7.157}$$

with Φ_e wall heat transfer.

The Nucleation model is implemented in Nucleation_paroi_PolyMAC_P0:

```
void Nucleation_paroi_PolyMAC_P0::set_param(Param& param)
```

This terms injected only on boundary elements and is fully explicit:

$$\texttt{secmem} \mathrel{+}= 6 \frac{\Phi^n_{\text{nucleation}}}{\max(d^n_{\text{nuc}}, 10^{-8}) \rho^n_g L^n_{\text{vap}}} \tag{7.158}$$

7.6.3 Interfacial area concentration with 2 groups (incoming)

The equations

A particular case of the solution can be obtained if we consider two groups of bubbles. For example, experimentally a limit can be observed between quasi-spherical and distorted bubbles. Then we can separate the distribution of those groups into 2 distinct distributions on either side of the critical diameter $Dsmc = 4\sqrt{\frac{\sigma}{g(\rho_l - \rho_g)}}$, with σ the surface tension, g gravity and ρ_g and ρ_l respectively the densities of the gas and the liquid. For the first group we get :

$$\begin{aligned} \frac{\partial a_{i1}}{\partial t} + \underbrace{\nabla\left(a_{i1}\mathbf{U}_{g1}\right)}_{\text{convection}} = & \underbrace{\frac{2}{3}\frac{a_{i1}}{\alpha_{g1}}\left(-\frac{\alpha_{g1}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density change}} \underbrace{-\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^2 \frac{a_{i1}}{\alpha_{g1}}\left(-\frac{\alpha_{g1}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density sliding}} \\ & + \underbrace{\frac{2}{3}\frac{a_{i1}}{\alpha_{g1}}\left(\frac{\Gamma_{g1}}{\rho_g}\right)}_{\text{Mass transfer}} \underbrace{-\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^2 \frac{a_{i1}}{\alpha_{g1}}\left(\frac{\Gamma_{g1}}{\rho_g}\right)}_{\text{Mass transfer sliding}} \\ & + \underbrace{\sum_j \psi^{\text{intergroup}}_{1j}}_{\text{intergroup sources}} + \underbrace{\sum_j \psi^{\text{internal}}_{1j}}_{\text{intragroup sources}} . \end{aligned} \tag{7.159}$$

For the second group, we get:

$$\begin{aligned} \frac{\partial a_{i2}}{\partial t} + \underbrace{\nabla\left(a_{i2}\mathbf{U}_{g2}\right)}_{\text{convection}} = & \underbrace{\frac{2}{3}\frac{a_{i2}}{\alpha_{g2}}\left(-\frac{\alpha_{g2}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density change}} + \underbrace{\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^2 \frac{a_{i1}}{\alpha_{g1}}\left(-\frac{\alpha_{g1}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density sliding}} + \\ & \underbrace{\frac{2}{3}\frac{a_{i2}}{\alpha_{g2}}\left(\frac{\Gamma_{g2}}{\rho_g}\right)}_{\text{Mass transfer}} + \underbrace{\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^2 \frac{a_{i1}}{\alpha_{g1}}\left(\frac{\Gamma_{g1}}{\rho_g}\right)}_{\text{Mass transfer sliding}} + \\ & \underbrace{\sum_j \psi^{\text{intergroup}}_{2j}}_{\text{intergroup sources}} + \underbrace{\sum_j \psi^{\text{internal}}_{2j}}_{\text{intragroup sources}} . \end{aligned} \tag{7.160}$$

The different mass transfer are:

$$\Gamma_{g1} = \underbrace{-\frac{\rho_g}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\sum_j \eta_j^{inter}}_{\text{Intergroup}} + \underbrace{\frac{\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^3}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\alpha_{g1}\left(\frac{d\rho_g}{dt}\right)}_{\text{Density group shift}} + \underbrace{\frac{1}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\Gamma_{\text{condensation g1}}}_{\text{Condensation}} + \underbrace{\frac{\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^3}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\Gamma_{\text{Nucleation g1}}}_{\text{Nucleation}}. \tag{7.161}$$

$$\Gamma_{g2} = \underbrace{\frac{\rho_g}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\sum_j \eta_j^{inter}}_{\text{Intergroup}} - \underbrace{\frac{\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^3}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\alpha_{g1}\left(\frac{d\rho_g}{dt}\right)}_{\text{Density group shift}} + \underbrace{\Gamma_{\text{condensation g2}} + \frac{\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\Gamma_{\text{condensation g1}}}_{\text{Condensation}} - \underbrace{\frac{\chi_d\left(\frac{D_{smc}}{D_{sm1}}\right)^3}{1+\chi_d\left(\frac{Dsmc}{Dsm1}\right)^3}\Gamma_{\text{Nucleation g1}}}_{\text{Nucleation}}. \tag{7.162}$$

χ_d is equal to 1 for a uniform distribution profile. Indeed, because there is no prior determination of the form of the solution of the distribution, the easiest from to consider is a uniform distribution. During the averaging process proposed in [**Kataoka2012**] , two new terms emerged from the instantaneous equation: a diffusion term and a lift term. For example, the diffusion term can be implemented as:

$$K\sqrt{u'^2}D_{sm}\nabla a_i = K\sqrt{\frac{2k}{3}}D_{sm}\nabla a_i, \tag{7.163}$$

with K a constant equal to $1/3$. However, it is essential to note that these terms have not yet been fully validated in various configurations [**Rassame2023**]. For the implementation in the code, we must rewrite it to get rid of D_{sm}. For the first group we have:

$$\frac{\partial a_{i1}}{\partial t} + \underbrace{\nabla\left(a_{i1}\mathbf{U}_{g1}\right)}_{\text{convection}} = \underbrace{\frac{2}{3}\frac{a_{i1}}{\alpha_{g1}}\left(-\frac{\alpha_{g1}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density change}} - \underbrace{\frac{\chi_d}{36}D_{smc}^2\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3\left(-\frac{\alpha_{g1}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density sliding}} + \underbrace{\frac{2}{3}\frac{a_{i1}}{\alpha_{g1}}\left(\frac{\Gamma_{g1}}{\rho_g}\right)}_{\text{Mass transfer}} - \underbrace{\frac{\chi_d}{36}D_{smc}^2\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3\left(\frac{\Gamma_{g1}}{\rho_g}\right)}_{\text{Mass transfer sliding}} + \underbrace{\sum_j \psi_{1j}^{intergroup}}_{\text{intergroup sources}} + \underbrace{\sum_j \psi_{1j}^{internal}}_{\text{intragroup sources}}. \tag{7.164}$$

For the second group, we obtain:

$$\frac{\partial a_{i2}}{\partial t} + \underbrace{\nabla\left(a_{i2}\mathbf{U}_{g2}\right)}_{\text{convection}} = \underbrace{\frac{2}{3}\frac{a_{i2}}{\alpha_{g2}}\left(-\frac{\alpha_{g2}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density change}} + \underbrace{\frac{\chi_d}{36}D_{smc}^2\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3\left(-\frac{\alpha_{g1}}{\rho_g}\frac{d\rho_g}{dt}\right)}_{\text{Density sliding}} + \underbrace{\frac{2}{3}\frac{a_{i2}}{\alpha_{g2}}\left(\frac{\Gamma_{g2}}{\rho_g}\right)}_{\text{Mass transfer}} + \underbrace{\frac{\chi_d}{36}D_{smc}^2\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3\left(\frac{\Gamma_{g1}}{\rho_g}\right)}_{\text{Mass transfer sliding}} + \underbrace{\sum_j \psi_{2j}^{\text{intergroup}}}_{\text{intergroup sources}} + \underbrace{\sum_j \psi_{2j}^{\text{internal}}}_{\text{intragroup sources}} . \tag{7.165}$$

The different mass transfer are:

$$\Gamma_{g1} = \underbrace{-\frac{\rho_g}{1+\frac{\chi_d}{216}Dsmc^3\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3}\sum_j \eta_j^{inter}}_{\text{Intergroup}} + \underbrace{\frac{\frac{\chi_d}{216}Dsmc^3\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3}{1+\frac{\chi_d}{216}Dsmc^3\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3}\alpha_{g1}\left(\frac{d\rho_g}{dt}\right)}_{\text{Density group shift}} + \underbrace{\frac{1}{1+\frac{\chi_d}{216}Dsmc^3\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3}\Gamma_{\text{condensation g1}}}_{\text{Condensation}} + \underbrace{\frac{\frac{\chi_d}{216}Dsmc^3\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3}{1+\frac{\chi_d}{216}Dsmc^3\left(\frac{a_{i1}}{\alpha_{g1}}\right)^3}\Gamma_{\text{Nucleation g1}}}_{\text{Nucleation}} . \tag{7.166}$$

$$\Gamma_{g2} = \underbrace{\frac{\rho_g}{1+\frac{\chi_d}{216}Dsmc^3(\frac{a_{i1}}{\alpha_{g1}})^3}\sum_j \eta_j^{inter}}_{\text{Intergroup}} - \underbrace{\frac{\frac{\chi_d}{216}Dsmc^3(\frac{a_{i1}}{\alpha_{g1}})^3}{1+\frac{\chi_d}{216}Dsmc^3(\frac{a_{i1}}{\alpha_{g1}})^3}\alpha_{g1}\left(\frac{d\rho_g}{dt}\right)}_{\text{Density group shift}} + \underbrace{\Gamma_{\text{condensation g2}} + \frac{\frac{\chi_d}{216}Dsmc^3(\frac{a_{i1}}{\alpha_{g1}})^3}{1+\frac{\chi_d}{216}Dsmc^3(\frac{a_{i1}}{\alpha_{g1}})^3}\Gamma_{\text{condensation g1}}}_{\text{Condensation}} - \underbrace{\frac{\frac{\chi_d}{216}Dsmc^3(\frac{a_{i1}}{\alpha_{g1}})^3}{1+\frac{\chi_d}{216}Dsmc^3(\frac{a_{i1}}{\alpha_{g1}})^3}\Gamma_{\text{Nucleation g1}}}_{\text{Nucleation}} . \tag{7.167}$$

The source terms

All source term models are based on five categories of mechanism: the Random Collisions (RC), the Wake Entrainment (WE), the Turbulent Impacts (TI), the Shearing-off (SO) and the Surface Instability (SI) (see Figure 7.4). The RC is a bubble coalescence phenomenon where 2 bubbles collide and merge because of a turbulent eddy of comparable size. The WE happens when one smaller bubble is in the wake of a bigger one, accelerates and collides it. The TI is due to turbulent eddies that break-up bubbles. The SO is a break-up phenomenon that source from the shearing-off of cap bubbles. The SI is due to the break-up of large bubbles due to their surface instability.
The number of processes and the dimensionless coefficient can strongly differ from one model to another :

- The **SUN2** model was developed for a 2 group configuration with a $200 \times 10\ mm^2$ confined rectangular channel data. The effect of the wall is then very significant. It was performed for liquid superficial velocity between 0.32 and 2.84 m/s and gas velocity between 0.39 and 2.01 m/s. It deals with cap-bubbly and churn-turbulent flows.

- The **Smith1** model was developed for a 2 group configuration with 0.102 mm and 0.152 mm diameter pipes. It deals with bubbly, cap-bubbly and churn-turbulent flows.

- The **Schlegel1** model was developed for a 2 group configuration with large diameter channels. It deals with bubbly and cap-bubbly flows. Several constitutive relations and correlations were used to tune this model.

- The **Fu2002** model was developed for a 2 group configuration for small round pipe.

- **Dave2016a** proposed new Smith coefficient based on optimization with genetic algorithm on all TOPFLOW DN200 (pipe 195.3 mm).

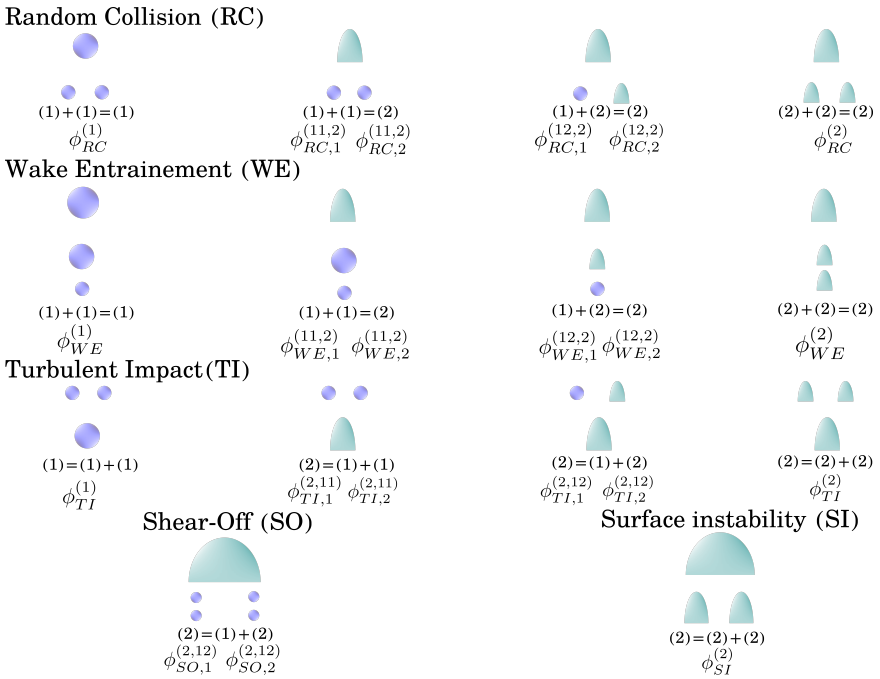


Figure 7.4: Representation of 2 group bubble mechanisms.

The coefficients of the previous models are summarized in Table 7.7. The model implemented is the one of Smith.
The source/sink terms of Random Collision (RC) are modeled as follows :

$$\phi_{RC}^{(1)} = -0.17 C_{RC}^{(1)} \lambda_{RC}^{(1)} \frac{\varepsilon^{1/3} \alpha_{g1} a_{i1}^{5/3}}{\alpha_{g1,max}^{1/3} \left(\alpha_{g1,max}^{1/3} - \alpha_{g1}^{1/3} \right)} \left[1 - \exp\left(-C_{RC1} \frac{\alpha_{g1,max}^{1/3} \alpha_{g1}^{1/3}}{\alpha_{g1,max}^{1/3} - \alpha_{g1}^{1/3}} \right) \right] \tag{7.168}$$

$$\phi_{RC,2}^{(11,2)} = 4.1 C_{RC}^{(1)} \lambda_{RC}^{(1)} \frac{\varepsilon^{1/3} \alpha_{g1} a_{i1}^{5/3}}{\alpha_{g1,max}^{2/3}} \left[1 - \exp\left(-C_{RC1} \frac{\alpha_{g1,max}^{1/3} \alpha_{g1}^{1/3}}{\alpha_{g1,max}^{1/3} - \alpha_{g1}^{1/3}} \right) \right] \left(1 - \frac{2}{3} D_{c1}^{*} \right). \tag{7.169}$$

Coefficient	Sun [**SUN2**]	Smith [**Smith1**]	Schlegel [**Schlegel1**]	Fu [**Fu2002**]	Dave [**Dave2016a**]
$C_{RC}^{(1)}$	0.005	0.01	0.01	0.0041	0.26
$C_{RC}^{(12,2)}$	0.005	0.01	0.05	0.005	0.41
$C_{RC}^{(2)}$	0.005	0.01	0.01	0.005	1.00
C_{RC0}	3.0	3.0	3.0	3.0	3.0
C_{RC1}	3.0	3.0	3.0	3.0	3.0
$\alpha_{g1,max}$	0.62	0.62	0.62	0.75	0.62
$C_{WE}^{(1)}$	0.002	0.002	0.002	0.002	0.001
$C_{WE}^{(12,2)}$	0.002	0.01	0.02	0.015	0.017
$C_{WE}^{(2)}$	0.005	0.01	0.05	10.	0.021
$C_{TI}^{(1)}$	0.1	0.05	0.05	0.0085	0.013
$C_{TI}^{(12,2)}$	0.02	0.04	0.02	0.02	0.006
$C_{TI}^{(2)}$	0.02	0.01	0.01	0.02	0.023
We_{cr1}	6.5	1.2	1.2	6.0	6.0
We_{cr2}	7.0	1.2	1.2	6.0	6.0
C_{SO}	3.8×10^{-5}	2.5×10^{-5}	5.0×10^{-5}	0.031	1.4×10^{-5}
$We_{c,SO}$	4500	4000	10	4500	4500

Table 7.7: Coefficients from Sun [**SUN2**], Smith [**Smith1**], Schlegel [**Schlegel1**], Fu [**Fu2002**], Dave [**Dave2016a**] for the Smith mechanistic modelling.

$$\phi_{RC,1}^{(12,2)} = -1.14 C_{RC}^{(12,2)} \lambda_{RC}^{(12,2)} \varepsilon^{1/3} \alpha_{g1}^{2/3} \alpha_{g2}^{4/3} a_{i1} a_{i2}^{2/3} \left[1 - \exp\left(-C_{RC1} \frac{\alpha_{g1,max}^{1/3} \alpha_{g1}^{1/3}}{\alpha_{g1,max}^{1/3} - \alpha_{g1}^{1/3}} \right) \right]. \tag{7.170}$$

$$\phi_{RC,2}^{(12,2)} = 1.80 C_{RC}^{(12,2)} \lambda_{RC}^{(12,2)} \varepsilon^{1/3} \alpha_{g1}^{5/3} \alpha_{g2}^{1/3} a_{i2}^{5/3} \left[1 - \exp\left(-C_{RC1} \frac{\alpha_{g1,max}^{1/3} \alpha_{g1}^{1/3}}{\alpha_{g1,max}^{1/3} - \alpha_{g1}^{1/3}} \right) \right]. \tag{7.171}$$

$$\phi_{RC}^{(2)} = -95.7 C_{RC}^{(2)} \lambda_{RC}^{(2)} \varepsilon^{1/3} \frac{\alpha_{g2}^{7/3}}{D_h^2} \frac{1}{a_{i2}^{1/3}} \left[1 - \exp\left(-C_{RC2} \alpha_{g2}^{1/2} \right) \right] \left(1 - 0.37 D_{c2}^{*3} \right). \tag{7.172}$$

$$\eta_{RC,2}^{(11,2)} = 3.15 C_{RC}^{(1)} \lambda_{RC}^{(1)} \frac{\varepsilon^{1/3} \alpha_{g1}^2 a_{i1}^{2/3}}{\alpha_{g1,max}^{2/3}} \left[1 - \exp\left(-C_{RC1} \frac{\alpha_{g1,max}^{1/3} \alpha_{g1}^{1/3}}{\alpha_{g1,max}^{1/3} - \alpha_{g1}^{1/3}} \right) \right] \left(1 - \frac{2}{3} D_{c1}^{*} \right). \tag{7.173}$$

$$\eta_{RC,2}^{(12,2)} = 1.44 C_{RC}^{(12,2)} \lambda_{RC}^{(12,2)} \varepsilon^{1/3} \alpha_{g1}^{5/3} \alpha_{g2}^{4/3} a_{i2}^{2/3} \left[1 - \exp\left(-C_{RC1} \frac{\alpha_{g1,max}^{1/3} \alpha_{g1}^{1/3}}{\alpha_{g1,max}^{1/3} - \alpha_{g1}^{1/3}} \right) \right]. \tag{7.174}$$

$\lambda_{RC}^{(1)}$, $\lambda_{RC}^{(12,2)}$, $\lambda_{RC}^{(2)}$ are defined as follows:

$$\lambda_{RC}^{(1)} = \exp\left(-C_{RC0} \frac{D_{sm1}^{5/6} \rho_l^{1/2} \varepsilon^{1/3}}{\sigma^{1/2}} \right), \tag{7.175}$$

$$\lambda_{RC}^{(2)} = \exp\left(-C_{RC0} \frac{D_{sm2}^{5/6} \rho_l^{1/2} \varepsilon^{1/3}}{\sigma^{1/2}} \right), \tag{7.176}$$

$$\lambda_{RC}^{(12,2)} = \lambda_{RC}^{(2)}. \tag{7.177}$$

In the above equations, $C_{RC}^{(1)}$, $C_{RC}^{(12,2)}$, $C_{RC}^{(2)}$ are three constant coefficients. C_{RC1}, C_{RC2} are coefficients accounting for effective range of influence of turbulent eddies. $\alpha_{g1,max}$ is the dense

packing limit for Group 1 bubbles. D_h is the hydraulic diameter. C_{RC0} is a constant coefficient. The source/sink terms of Wake Entrainment (WE) are modeled as follows :

$$\phi_{WE}^{(1)} = -0.17 C_{WE}^{(1)} C_{D1}^{1/3} U_{r1} a_{i1}^2. \tag{7.178}$$

$$\phi_{WE,2}^{(11,2)} = 2.57 C_{WE}^{(11,2)} C_{D1}^{1/3} U_{r1} a_{i1}^2 \left(1 - \frac{2}{3} D_{c1}^*\right). \tag{7.179}$$

$$\phi_{WE,l1}^{(12,2)} = -0.33 C_{WE}^{(12,2)} U_{w12} a_{i1} a_{i2}. \tag{7.180}$$

$$\phi_{WE,g2}^{(12,2)} = 0.922 C_{WE}^{(12,2)} U_{w12} \alpha_{g1} \frac{a_{i2}^2}{\alpha_{g2}}. \tag{7.181}$$

$$\phi_{WE}^{(2)} = -1.02 C_{WE}^{(2)} \left[1 - \exp(-0.7\alpha_{g2})\right] U_{rw2} \frac{a_{i2}^2}{\alpha_{g2}} \left(1 - 0.10 D_{c2}^{*2}\right). \tag{7.182}$$

$$\eta_{WE,2}^{(11,2)} = 3.85 C_{WE}^{(1)} C_{D1}^{1/3} U_{r1} \alpha_{g1} a_{i1} \left(1 - \frac{2}{3} D_{c1}^*\right). \tag{7.183}$$

$$\eta_{WE,2}^{(12,2)} = 0.33 C_{WE}^{(12,2)} U_{w12} \alpha_{g1} a_{i2}. \tag{7.184}$$

In the above equation

$$\begin{aligned} U_{rw2} &= 0.94 U_{r2} C_{D2}^{1/3}, \\ U_{w12} &= U_{rw2} + U_{r1} - U_{r2}, \\ D_{c2}^* &= \frac{D_c}{D_{sm2}}, \end{aligned}$$

and

$$\begin{aligned} C_{D1} &= \frac{2}{3} D_{sm1} \sqrt{\frac{g\Delta\rho}{\sigma}} \left(\frac{1 + 17.67[f(\alpha_{g1})]^{6/7}}{18.67 f(\alpha_{g1})}\right)^2 \text{ with } f(\alpha_{g1}) = (1 - \alpha_{g1})^{1.5}, \\ C_{D2} &= \frac{8}{3}(1 - \alpha_{g2})^2. \end{aligned}$$

In the above equations $C_{WE}^{(1)}$, $C_{WE}^{(11,2)}$, $C_{WE}^{(12,2)}$, $C_{WE}^{(2)}$ are constant coefficients.
The source/sink terms of Turbulent Impact (TI) are modeled as follows :

$$\phi_{TI}^{(1)} = 0.12 C_{TI}^{(1)} \varepsilon^{1/3} (1 - \alpha_g) \left(\frac{a_{i1}^{5/3}}{\alpha_{g1}^{2/3}}\right) \exp\left(-\frac{We_{cr1}}{We_1}\right) \sqrt{1 - \frac{We_{cr1}}{We_1}}. \tag{7.185}$$

$$\phi_{TI,1}^{(2,1)} = 6.165 C_{TI}^{(2,1)} \varepsilon^{1/3} (1 - \alpha_g) \left(\frac{a_{i2}^{5/3}}{\alpha_{g2}^{2/3}}\right) \exp\left(-\frac{We_{cr2}}{We_2}\right) \sqrt{1 - \frac{We_{cr2}}{We_2}} \left(0.212 D_{c2}^{*13/3} - 0.167 D_{c2}^{*5}\right). \tag{7.186}$$

$$\phi_{TI,2}^{(2)} = 0.378 C_{TI}^{(2)} \varepsilon^{1/3} (1 - \alpha_g) \left(\frac{a_{i2}^{5/3}}{\alpha_{g2}^{2/3}}\right) \exp\left(-\frac{We_{cr2}}{We_2}\right) \sqrt{1 - \frac{We_{cr2}}{We_2}} \left(1 - 0.212 D_{c2}^{*13/3}\right), \tag{7.187}$$

$$\eta_{TI,2}^{(2,1)} = -11.65 C_{TI}^{(2,1)} \varepsilon^{1/3} (1 - \alpha_g) \alpha_{g2}^{1/3} a_{i2}^{2/3} \exp\left(-\frac{We_{cr2}}{We_2}\right) \sqrt{1 - \frac{We_{cr2}}{We_2}} \left(0.15 D_{c2}^{*16/3} - 0.117 D_{c2}^{*6}\right). \tag{7.188}$$

$$\eta_{TI,1}^{(2,1)} = -\eta_{TI,2}^{(2,1)}, \tag{7.189}$$

with the following expressions for We_1 and We_2 :

$$We_1 = \frac{2\rho_l \varepsilon^{2/3} (D_{sm1})^{5/3}}{\sigma},$$
$$We_2 = \frac{2\rho_l \varepsilon^{2/3} (D_{sm2})^{5/3}}{\sigma}.$$

$C_{TI}^{(1)}$, $C_{TI}^{(2,1)}$, $C_{TI}^{(2)}$ are constant coefficients. We_{cr1}, We_{cr2} are critical Weber number for breakup due to turbulent impact.
The source/sink terms of Shearing-off (SO) are modeled as follows :

$$\phi_{SO,1}^{(2,12)} = 8.0 C_{SO} \frac{\rho_l^{3/5} U_{g2}^{1/5} \sigma^{2/5}}{\rho_g D_h^{2/5} We_c^{3/5}} \frac{a_{i2}^2}{\alpha_{g2}} \left[1 - \left(\frac{We_{c,SO}}{We_{m2}} \right)^4 \right]. \tag{7.190}$$

$$\phi_{SO,2}^{(2,12)} = -0.36 C_{SO} \left(\frac{\sigma}{\rho_g U_{g2}} \right) \frac{a_{i2}^3}{\alpha_{g2}^2} \left[1 - \left(\frac{We_{c,SO}}{We_{m2}} \right) \right]. \tag{7.191}$$

$$\eta_{SO,2}^{(2,12)} = -2.33 C_{SO} \left(\frac{\sigma}{\rho_g U_{g2}} \right) \frac{a_{i2}^2}{\alpha_{g2}} \left[1 - \left(\frac{We_{c,SO}}{We_{m2}} \right)^4 \right]. \tag{7.192}$$

$$\eta_{SO,1}^{(2,12)} = -\eta_{SO,2}^{(2,12)}. \tag{7.193}$$

C_{SO} is a constant coefficient. $We_{c,SO}$ is a critical weber number for shearing-off of small bubbles from large cap bubbles. We_{m2}, We_c, D_h.
The source/sink terms of Surface Instability (SI) are modeled as follows :

$$\begin{aligned} \phi_{SI}^{(2)} &= 2.616 \times 10^{-4} C_{RC}^{(2)} \varepsilon^{1/3} \frac{1}{D_h^2} \alpha_{g2}^2 \left(\frac{\sigma}{g \Delta \rho} \right)^{1/6} \left[1 - \exp\left(-C_{RC2} \alpha_{g2}^{1/2} \right) \right], \\ &+ 1.425 \times 10^{-7} C_{WE}^{(2)} (1 - \exp(-0.7 \alpha_{g2})) U_{rw2} \alpha_{g2}^2 \left(\frac{\sigma}{g \Delta \rho} \right)^{-1}, \end{aligned} \tag{7.194}$$

$C_{RC}^{(2)}$ and $C_{WE}^{(2)}$ are constant coefficients from Random collision and Wake Entrainment source terms. D_h is the hydraulic diameter.

Chapter 8

Turbulence

8.1 Code structure

Translate files and class names in English

All models are of the form: time derivative + advection + diffusion = production + dissipation + extra terms. This section describes the physical equations of the models, their implementation in the code, the constants values and a link to where they are defined. If their is any difference compared with the original model, it *must* be emphasized.

The turbulence model

All available single phase models are two-equations models involving the turbulent kinetic energy k equation. Thus all models share the same k equation implementation. The source files are

- Source_Production_energie_cin_turb.cpp
- Source_Dissipation_energie_cin_turb.cpp

No extra term to take buoyancy effect into account is available.

The base source

- Source_Production_echelle_temp_taux_diss_turb.cpp
- Source_Dissipation_echelle_temp_taux_diss_turb.cpp
- Source_Diffusion_croisee_echelle_temp_taux_diss_turb.cpp
- Source_Diffusion_supplementaire_echelle_temp_turb.cpp

8.2 Single-phase turbulence

8.2.1 Eddy-viscosity models

All of the constants used in the models are user-defined in the calculation dataset. This enables an easy transition from one turbulence model to another. The models that one can use to launch a calculation are the following.

1988 Wilcox $k-\omega$

The Wilcox version of the $k-\omega$ model is described in details by **Wilcox1988**. The turbulent viscosity is defined by $\nu_t = \frac{k}{\omega}$. The two equations are:

$$\partial_t \rho_l k + \nabla \cdot (\rho_l k \overrightarrow{u_l}) - \nabla \left(\rho_l (\nu_l + \sigma_k \nu_t) \underline{\nabla} k\right) = \underline{\underline{\tau_R}} :: \underline{\underline{\nabla}} \overrightarrow{u_l} - \beta_k \rho k \omega \tag{8.1}$$

$$\partial_t \rho_l \omega + \nabla \cdot (\alpha_l \rho_l \omega \overrightarrow{u_l}) - \nabla \left(\rho_l (\nu_l + \sigma_\omega \nu_t) \underline{\nabla} \omega\right) = \alpha_\omega \frac{\omega}{k} \underline{\underline{\tau_R}} :: \underline{\underline{\nabla}} \overrightarrow{u_l} - \beta_\omega \rho \omega^2 \tag{8.2}$$

The default values of the constants are

- $\alpha_\omega = 0.55$,
- $\beta_k = 0.09$,
- $\beta_\omega = 0.075$,
- $\sigma_k = 0.5$,
- $\sigma_\omega = 0.5$.

Kok $k - \omega$

This model [**Kok1999**] was introduced after the Menter SST $k - \omega$ model [**Menter1993**, **Menter2003**] showed the importance of cross-diffusion. The differences with the 1988 Wilcox model reside in the addition of a cross-diffusion term ($\sigma_d \frac{\rho_l}{\omega} \max\{\underline{\nabla} k \cdot \underline{\nabla} \omega, 0\}$) and a modification of the value of some constants. The turbulent equations are:

$$\partial_t \rho_l k + \nabla \cdot (\rho_l k \overrightarrow{u_l}) = \underline{\underline{\tau_R}} :: \underline{\underline{\nabla}} \overrightarrow{u_l} - \beta_k \rho k \omega + \nabla \left(\rho_l (\nu_l + \sigma_k \nu_t) \underline{\nabla} k\right) \tag{8.3}$$

$$\begin{aligned} \partial_t \rho_l \omega + \nabla \cdot (\alpha_l \rho_l \omega \overrightarrow{u_l}) = \alpha_\omega \frac{\omega}{k} \underline{\underline{\tau_R}} :: \underline{\underline{\nabla}} \overrightarrow{u_l} - \beta_\omega \rho \omega^2 + \nabla \left(\rho_l \left(\nu_l + \sigma_\omega \nu_t\right) \underline{\nabla} \omega\right) \\ + \sigma_d \frac{\rho_l}{\omega} \max\{\underline{\nabla} k \cdot \underline{\nabla} \omega,\ 0\} \end{aligned} \tag{8.4}$$

The values of the constants are $\alpha_\omega = 0.5$, $\beta_k = 0.09$, $\beta_\omega = 0.075$, $\sigma_k = 2/3$, $\sigma_\omega = 0.5$ and $\sigma_d = 0.5$.

Kok $k - \tau$

This is a variation of the 1999 Kok $k - \omega$. In this model [**Ktau2000**], the time scale $\tau = \frac{1}{\omega}$ is introduced. We therefore have $\nu_t = k\tau$. There is an additional diffusion term that comes out of the calculation ($-8\rho_l(\nu_l + \sigma_\omega \nu_t)||\underline{\nabla}\sqrt{\tau}||^2$). The turbulent equations become:

$$\partial_t \rho_l k + \nabla \cdot (\rho_l k \overrightarrow{u_l}) = \underline{\underline{\tau_R}} :: \underline{\underline{\nabla}} \overrightarrow{u_l} - \frac{\beta_k \rho k}{\tau} + \nabla(\rho_l (\nu_l + \sigma_k \nu_t) \underline{\nabla} k) \tag{8.5}$$

$$\begin{aligned} \partial_t \rho_l \tau + \nabla \cdot (\rho_l \tau \overrightarrow{u_l}) = -\alpha_\omega \frac{\tau}{k} \underline{\underline{\tau_R}} :: \underline{\underline{\nabla}} \overrightarrow{u_l} + \beta_\omega \rho + \nabla(\rho_l (\nu_l + \sigma_\omega \nu_t) \underline{\nabla} \tau) \\ + \sigma_d \rho_l \tau \times \min\{\underline{\nabla} k \cdot \underline{\nabla} \tau,\ 0\} - 8\rho_l \left(\nu_l + \sigma_\omega \nu_t\right) ||\underline{\nabla}\sqrt{\tau}||^2 \end{aligned} \tag{8.6}$$

The $-8\rho_l(\nu_l + \sigma_\omega \nu_t)||\underline{\nabla}\sqrt{\tau}||^2$ term presents important numerical difficulties close to the wall. In order to limit these issues, we have tried to implicit this term in 3 different ways. **The comparison between these methods and the determination of the most robust solution is ongoing**.

The constants are the same than in the 1999 Kok $k - \omega$ model: $\alpha_\omega = 0.5$, $\beta_k = 0.09$, $\beta_\omega = 0.075$, $\sigma_k = 2/3$, $\sigma_\omega = 0.5$, $\sigma_d = 0.5$.

2006 Wilcox $k-\omega$

The 2006 Wilcox k_ω model [**Wilcox2006**] is the same as the Kok $k-\omega$ with different coefficients. It is an update of the 1988 Wilcox $k-\omega$ model. The turbulent equations are the same as in equation 8.3. A notable difference is the introduction of a blending function for β_ω.

The values of the constants are: $\alpha_\omega = 0.52$, $\beta_k = 0.09$, $\beta_\omega = 0.0705 \cdot f(\Omega_{ij}, S_{ij})$, $\sigma_k = 0.6$, $\sigma_\omega = 0.5$, $\sigma_d = 0.125$.

- expliciter la fonction de blending
- ajouter implémentation de chaque terme
-

8.2.2 Large-Eddy Simulation

Large-Eddy simulation using Pb_Multiphase is a work in progress. This section will be filled later.

8.3 Two-phase turbulence

8.3.1 Eddy viscosity-like model

The Sato model [**Sato1981a**] is added in `Viscosite_turbulente_sato.cpp`. The original formula is

$$\epsilon'' = \left(1 - \exp\left(-\frac{y^+}{A^+}\right)\right)^2 k_1 \alpha \frac{D_b}{2} U_B. \tag{8.7}$$

with the coefficient $A^+ = 16$ and $k_1 = 1.2$. The bubble diameter D_b is modeled to take the deformation of the bubble into account at the wall. The velocity U_B, defined in the article, is the relative velocity.[1] The following expression is defined:

$$D_b = \begin{cases} 0 \text{ if } 0 < y < 20\ \mu m, \\ 4y\left(\widehat{D_B} - y\right)/\widehat{D_B} \text{ if } 20\ \mu m < y < \widehat{D_B}/2, \\ \widehat{D_B} \text{ if } \widehat{D_B}/2 < y < R. \end{cases} \tag{8.8}$$

with $\widehat{D_B}$ the cross-sectional mean diameter of the bubbles.

In TrioCFD, the squared coefficient depending on y^+ is not implemented. The bubble diameter is taken as is without the prescribed function.

8.3.2 Bubble-induced turbulence model in two-equations model

The HZDR model is described in the paper of **Rzehak2013a**, **Rzehak2013b**, **Rzehak2015**, **Colombo2021**. Their approach is to add a production term S_k^{BI} to the k equation and a dissipation term S_ω^{BI} to the ω equation. The general assumption is to consider that *all energy lost by the bubble to drag is converted to turbulent kinetic energy in the wake of the bubble* [**Rzehak2013b**].

In comparison with the current version of the code, they implemented those two additional terms in a $k-\omega$ SST turbulence model. In CMFD, only the production and dissipation terms have been extracted and implemented without (yet) the SST additional process. Thus the prescribed coefficient might not be well suited.

[1] Very poor notation, in our humble opinion.

The two terms are related by the expression

$$S_\omega^{\mathrm{BI}} = \frac{1}{C_\mu k_l}\mathcal{S}_\varepsilon^{\mathrm{BI}} - \frac{\omega_L}{k_L}S_k^{\mathrm{BI}} \tag{8.9}$$

with

$$\mathcal{S}_\varepsilon^{\mathrm{BI}} = C_{\varepsilon B}\frac{S_k^{\mathrm{BI}}}{\tau} = C_{\varepsilon B}\frac{S_k^{\mathrm{BI}}\varepsilon_l}{k_l} \tag{8.10}$$

In the code, the added dissipation term takes the following form

$$S_\omega^{\mathrm{BI}} = \frac{C_\varepsilon}{C_\nu D_b\sqrt{k}}\mathcal{S}_k^{\mathrm{BI}} \tag{8.11}$$

with

$$\mathcal{S}_k^{\mathrm{BI}} = \frac{3}{4}C_k\frac{C_d}{D_b}\frac{\alpha_g}{\alpha_l}u_r^3 \tag{8.12}$$

$$C_d = \max\left(\min\left(\frac{16}{Re_b}\left(1+0.15Re_b^{0.687}\right),\ \frac{48}{Re_b}\right),\ \frac{8Eo}{3\left(4+Eo\right)}\right) \tag{8.13}$$

$$Eo = \frac{g|\rho_l-\rho_g|D_b^2}{\sigma} \tag{8.14}$$

$$Re_b = \frac{D_b u_r}{\nu_l} \tag{8.15}$$

It is derived directly from Source_base and currently only implemented in PolyMAC.

The added production term in the turbulent kinetic energy equation is defined in Production_HZDR_PolyMAC_P0. The added term in the dissipation equation is defined in Source_Dissipation_HZDR_PolyMAC_P0. TODO: To avoid code duplication, the dissipation source term should call the production source term.

8.3.3 RSM-like model

By two-phase turbulence, we mean the effect of the bubbles on the turbulence in the liquid phase. To model this, we implemented the models developed during the PhD of Antoine du Cluzeau [**DuCluzeau2019**, **Cluzeau2019**, **Cluzeau2019a**]. Following the work of **Risso2018**, the authors divide the velocity fluctuations caused by the movement of bubbles in the fluid in two parts: wake-induced turbulence and wake-induced fluctuations. The total Reynolds stress tensor is the sum of all single-phase (calculated using 2-equation turbulence models) and two-phase turbulence.

$$\underline{\underline{\tau_R}} = \underline{\underline{\tau_R}}_{\text{single-phase}} + \underline{\underline{\tau_R}}_{\text{WIF}} + \underline{\underline{\tau_R}}_{\text{WIT}} \tag{8.16}$$

This model is only available in PolyMAC, for now.

Wake-induced fluctuations

Wake-induced fluctuations are the anisotropic effects of the average wake. These fluctuations are primarily in the direction of the liquid-gas velocity difference, i.e. in the vertical direction. No transport equation is necessary to model this term.

$$\underline{\underline{\tau_R}}_{\text{WIF}} = \alpha_v|\overrightarrow{u_v} - \overrightarrow{u_l}|^2\begin{bmatrix}3/20 & 0 & 0\\ 0 & 3/20 & 0\\ 0 & 0 & 1/5+3C_v/2\end{bmatrix} \tag{8.17}$$

with $C_v = 0.36$.

During his internship, Moncef El Moatamid defined a new formulation using the work of **Biesheuvel1984**

$$\underline{\underline{\tau_R}}_{\text{WIF}} = \alpha_v \left(\frac{3}{20} u_r^2 \underline{\underline{I}} + \left(\frac{1}{20} + 0.25 \times \frac{3}{2} \gamma^3 \right) \underline{u_r u_r} \right) \tag{8.18}$$

This formulation allows to have a tensor for the second part of the equation

Wake-induced turbulence

Wake-induced turbulence is the isotropic contribution of bubbles to the velocity fluctuations. It comes from the instabilities of bubble wakes. It takes the shape of an additional transport equation for a specific kinetic energy k_{WIT}.

$$\underline{\underline{\tau_R}}_{\text{WIT}} = k^{WIT} \frac{2}{3} \delta_{ij} \tag{8.19}$$

$$\frac{Dk^{WIT}}{Dt} = \underbrace{C_D \nabla^2 k^{WIT}}_{\text{Diffusion}} - \underbrace{\frac{2\nu_l C_D' Re_b}{C_\Lambda^2 d_b^2} k^{WIT}}_{\text{Dissipation}} + \underbrace{\alpha_v \frac{(\rho_l - \rho_v)}{\rho_l} g |\overrightarrow{u_v} - \overrightarrow{u_l}| \left(0.9 - exp\left(-\frac{Re_b}{Re_b^c} \right) \right)}_{\text{Production}} \tag{8.20}$$

where $C_\Lambda = 2.7$, $Re_b^c = 170$, C_D' is a user-inputted drag coefficient and C_D is a turbulent diffusion coefficient.

This model is implemented in `Energie_cinetique_turbulente_WIT.cpp` and inherits from `Convection_Diffusion_std`. The different terms of the right-hand side of the equations must be modeled. Thus, we have

- Viscosite_turbulente_WIT
- Dissipation_WIT_PolyMAC_P0
- Production_WIT_PolyMAC_P0

As specified above, it is currently only available with PolyMAC. To take the WIT into account in the momentum equation, one must specify its presence in the diffusion term using[2]

```
diffusion {
  turbulente multiple {
    k_omega k_omega { }
    WIT WIT { }
    WIF WIF { }
  }
}
```

Then, in the WIT equation bloc, the model for the turbulente diffusion of WIT is specified

```
diffusion { turbulente SGDH_WIT { } }
```

For the turbulent viscosity, an additional diffusion term must be specified in the data file to model the turbulent transport of WIT. Two models are available, a single gradient diffusion one and a generalized gradient diffusion one.

Production_WIT

$$\alpha g u_r \frac{\rho_l - \rho_g}{\rho_l} \left(0.9 - \exp\left(Re_b - Re_c\right)\right) \tag{8.21}$$

[2]This syntax might evolve to avoid repeating the names.

with

$$Re_b = \frac{D_b u_r}{\nu_l}. \tag{8.22}$$

The Reynolds number Re_c is a user parameter with a default value of 170 [**DuCluzeau2019**]. Only the `secmem` matrix is filled.

Dissipation_WIT Drag coefficient from Tomiyama, same than HZDR.

$$\frac{2\nu C_d Re_b k_{\mathrm{WIT}}}{C_\lambda^2 D_b^2} \tag{8.23}$$

with

$$C_d = \max\left(\min\left(\frac{16}{Re_b}\left(1+0.15Re_b^{0.687}\right), \frac{48}{Re_b}\right), \frac{8Eo}{3\left(4+Eo\right)}\right) \tag{8.24}$$

$$Eo = \frac{g|\rho_l - \rho_g|D_b^2}{\sigma} \tag{8.25}$$

$$Re_b = \frac{D_b u_r}{\nu_l} \tag{8.26}$$

Only the `secmem` matrix is filled.

Transport_turbulent_SGDH_WIT It comes from the paper of **Almeras2014**. It computes a characteristic time scale of the form

$$\tau = \frac{2}{3}\alpha_g u_r \frac{D_b}{\delta^3}\gamma^{2/3} \tag{8.27}$$

with δ the wake size, γ the bubble aspect ratio and C_s a constant Then it modifies the viscosity as

$$\nu = \frac{\mu_0}{\nu_0} C_s \tau \tag{8.28}$$

```
Param param(que_suis_je());
param.ajouter("Aspect_ratio", &gamma_); // rapport d'aspet des bulles
param.ajouter("Influence_area", &delta_); // paramtre modle d'Almras 2014 (
   taille du sillage)
param.ajouter("C_s", &C_s); // paramtre modle d'Almras 2014
```

Transport_turbulent_GGDH_WIT Same as SGDH but works with $\nu(i, \text{liq. idx}, \text{dim I}, \text{dim J})$ and R_{ij}

```
param.ajouter("Aspect_ratio", &gamma_); // rapport d'aspet des bulles
param.ajouter("Influence_area", &delta_); // paramtre modle d'Almras 2014 (
   taille du sillage)
param.ajouter("C_s", &C_s); // paramtre modle d'Almras 2014
//param.ajouter("vitesse_rel_attendue", &ur_user, Param::REQUIRED); //
   valeur de ur  prendre si u_r(i,0)=0
param.ajouter("Limiteur_alpha", &limiteur_alpha_, Param::REQUIRED); //
   valeur minimal de (1-alpha) pour utiliser le modle d'Almras
```

Viscosite_turbulente_WIT With the Reynolds_stress method, it fills the diagonal with $2/3 \times k_{\mathrm{WIT}}$.

```
param.ajouter("limiter|limiteur", &limiter_);
```

8.4 Boundary conditions

The algorithm for the boundary conditions is run independently in each near-wall cell. It's steps are:

1. Calculate $u_{\parallel}$
2. Calculate the friction viscosity u_τ using a Newton algorithm, to solve $\frac{u_{\parallel}}{u_\tau} = u_+(\frac{yu_\tau}{\nu})$
3. Calculate the shear stress $\tau_f = \rho u_\tau^2$
4. Obtain a friction coefficient at the wall $\alpha = \tau_f / u_{\parallel}$ that will be used to calculate the momentum flux boundary condition
5. If a thermal wall law is required, use the equation on the turbulent heat flux presented above to determine $q_{\text{wall}\to\text{phase } n}$
6. Calculate one of the following BC's on k:
   - Use $k = 0$ (if wall-resolved, i.e. $y_+ < 5$)
   - Use $\partial_y k = 0$ (if log-law region i.e. $y_+ > 30$)
   - Use equation a boundary condition that implements a blending between the two: Cond_lim_k_simple_flux_nul. The blending function is presented in equation (8.29).
   - No longer used: use the equations on the turbulent quantities presented above to determine $k(y/2)$ in the first half-cell and use it as Dirichlet BC: Cond_lim_k_complique_transition_flux_nul_demi
7. Either:
   - Calculate one of the following BC's on ω:
     - Use the equations on the turbulent quantities presented above to determine $\omega(y/2)$ in the first half-cell and use it as Dirichlet BC: Cond_lim_omega_demi
     - Use the equations on the turbulent quantities presented above to determine $\omega(y)$ in the first cell and use 10 times this value as Dirichlet BC at the wall: Cond_lim_omega_dix
   - Calculate one of the following BC's on τ:
     - Use $\tau = 0$
8. Iterate one time step of the complete system of equations (momentum, mass, energy, pressure, turbulent quantities)

8.4.1 Boundary condition on the turbulent kinetic energy

Basic boundary conditions

This boundary condition is implemented in `Cond_lim_k_simple_flux_nul.cpp` (and should get a new name).

It implements a seamless transition from a $k = 0$ at the wall for $y_+ < 5$ to a zero-flux condition for $y_+ > 30$ using a blending function.

The value on the wall is always zero, and the exchange coefficient between the wall and the first cell is:

$$h = \frac{\mu_{\text{tot}}}{y_{\text{loc}}} \times \left(1 + \tanh\left(\frac{1}{10}\frac{y_{\text{loc}} u_\tau}{{\nu_c}^2}\right)\right) \tag{8.29}$$

More complex boundary conditions

It is implemented in `Cond_lim_k_complique_transition_flux_nul_demi.cpp` but not used anymore. It is kept until an possible refactoring and cleaning operation.

$$h = \frac{2\mu_{\text{tot}}}{y_{\text{loc}}} \times \left(1 + \tanh\left(\frac{y_+^3}{50}\right)\right) \tag{8.30}$$

The turbulent kinetic energy is then computed by

$$k = u_\tau^2 \times \max\left[(1-b_1)\, f_1 + b_1 f_2, 0\right] \times (1-b_2) + b_2 \times f_3 \tag{8.31}$$

with

$$b_1 = \tanh\left[\left(\frac{y_+}{4}\right)^{10}\right] \tag{8.32}$$

$$b_2 = \tanh\left[\left(\frac{y_+}{2500}\right)^{1.4}\right] \tag{8.33}$$

$$f_1 = \frac{(y_+ - 1)^2}{30} \tag{8.34}$$

$$f_2 = \frac{1}{\sqrt{\beta_k}} - 0.08 \times \left(|4.6 - \log(y_+)|\right)^3 \tag{8.35}$$

$$f_3 = \frac{4}{\text{sqrt}\,(\beta_k)} \tag{8.36}$$

8.4.2 Boundary condition on the dissipation time scale equation

Advanced boundary condition

It is implemented in `Cond_lim_omega_demi.cpp`. It uses a virtual point at half the distance to the wall to impose the value of ω.

A blending function is used for the wall value of ω defined as

$$\omega = f_{\text{b}}\omega_1 + (1 - f_{\text{b}})\omega_2 \tag{8.37}$$

with

$$y_p = \frac{y}{u_\tau \nu}, \tag{8.38}$$

$$\omega_{\text{vis}} = \frac{6\nu}{\beta_\omega y_{\text{loc}}^2}, \tag{8.39}$$

$$\omega_{\log} = \frac{u_\tau}{\sqrt{\beta_k}\kappa y_{\text{loc}}}, \tag{8.40}$$

$$\omega_1 = \omega_{\text{vis}} + \omega_{\log}, \tag{8.41}$$

$$\omega_2 = \left((\omega_{\text{vis}})^{1.2} + (\omega_{\log})^{1.2}\right)^{1/1.2}, \tag{8.42}$$

$$f_{\text{b}} = \tanh\left(\left[\frac{y_+}{10}\right]^4\right). \tag{8.43}$$

The constants are the common ones of the $k-\omega$ models: $\beta_k = 0.09$ (called C_μ in $k-\varepsilon$ model), $\beta_\omega = 0.075$ and the VON KARMAN constant $\kappa = 0.41$.

Basic boundary condition

It is implemented in `Cond_lim_omega_dix`.

Set the wall value at $\omega_{\text{wall}} = C_{\text{wall}} \times 6\frac{\nu}{\beta_\omega y^2}$ with

- $C_{\text{wall}} = 10$, a user parameter. Its default value is prescribed by **Wilcox2008**.
- y the wall distance of the elem
- ν the viscosity

8.4.3 Wall functions

Adaptive wall function

Ramstorfer wall function

Adaptative wall flux

This function computes the wall heat flux based on Kader paper.

It computes a θ^+ quantity defined as

$$Pry^+ \exp(-\gamma) + \exp(-1/\gamma) \times \left(2.12 \log\left(\frac{1.5 \times (1+y^+)(2-y_D)}{(1+2(1-y_D)^2)}\right) + \beta\right) \tag{8.44}$$

with

$$y_D = 0, \tag{8.45}$$

$$Pr = \frac{\mu Cp}{\lambda}, \tag{8.46}$$

$$y^+ = \frac{\rho y u_\tau}{\mu}, \tag{8.47}$$

$$\beta = \left(3.85 Pr^{1/3} - 1.3\right)^2 + 2.12 \log(Pr), \tag{8.48}$$

$$\gamma = \frac{0.01 \times (Pry^+)^4}{1 + 5Pr^3 y^+}, \tag{8.49}$$

The value y_D was at first defined as y/D_h but lately set to zero.

Chapter 9

Homogeneous mixture modeling

This chapter focuses on the modeling of homogeneous mixtures through TrioCFD multiphase. The first part is a review of the use of the evanescence operator (section 9.1) before describing the equilibrium models dedicated to mixtures (section 9.2).

9.1 Homogeneous evanescence for mixture modeling

The `Pb_multiphase` framework allows to model a mixture by taking advantage of the evanescence operator described in section 4. By choosing the correct values of `alpha_res`=1 and `alpha_res_min`=0.5, one can reduce the system to three partial differential equations corresponding to the dynamics of a mixture. Without any model, this approach can be referred to as the Homogeneous Equilibrium Model as the two phases can be considered dynamically locked and at the same temperature. This can be the starting point of a homogeneous relaxation model approach by adding different models to describe the relation between the two phases.

Imposing the above-mentioned values lead to a simplification of Equation 4.1:

$$\begin{pmatrix} \mathcal{Q}_{\text{pred}} \\ \mathcal{Q}_{\text{mino}} \end{pmatrix} \Longrightarrow \begin{pmatrix} \mathcal{Q}_{\text{pred}} + \mathcal{Q}_{\text{mino}} \\ v_{\text{mino}} = v_{\text{pred}} + v_{\text{drift}} \end{pmatrix} \tag{9.1}$$

9.2 Dedicated mixture modeling

9.2.1 Drift velocity

The drift velocity of a gas phase u_{gj} is defined as the velocity of the gas phase u_g with respect to the volume center of the mixture j. The area average $\prec F \succ$ of a quantity F over the cross-sectional area A is defined by:

$$\prec F \succ = \frac{1}{A} \int_A F dA \tag{9.2}$$

The one-dimensional drift-flux model is defined as:

$$\frac{\prec j_g \succ}{\prec \alpha_g \succ} = C_0 \prec j \succ + V_{g0}, \tag{9.3}$$

with

$$C_0 = \frac{\prec \alpha_g j \succ}{\prec \alpha_g \succ \prec j \succ}, \tag{9.4}$$

$$V_{g0} = \frac{\prec \alpha_g u_{gj} \succ}{\prec \alpha_g \succ}. \tag{9.5}$$

The model is implemented in:

```
void Vitesse_derive_base::set_param(Param& param)
```

The drift velocity operator must fill vr and dvr tabs for each dimension d so that :

- $\texttt{vr}(k1, k2, d)$ Relative velocity in dimension d
- $\texttt{dvr}(k1, k2, d, \texttt{dimension} * k2 + d)$ Relative velocity derivative regarding the inlet superficial velocity $\texttt{v}(d, k2)$
- $\texttt{vr}(k2), k1, d) = -\texttt{output.vr}(k1, k2), d)$
- $\texttt{dvr}(k1, k2), d, dimension * k2 + d) = -\texttt{vr}(k1, k2), d, dimension * k2 + d)$

Model	Used	Validated	Test case
Constant bubble	✓	✓(100%)	TrioCFD/Drift flux, Trust/Canal bouillant drift
Ishii-Hibiki	✓	✓(100%)	TrioCFD/Drift flux, Trust/Canal bouillant drift
Spelt	✓	✗(0%)	
Forces	✓	✓(100%)	TrioCFD/Drift flux, Trust/Canal bouillant drift

Table 9.1: Availability of heat flux models in Trio_CFD.

Constant

The model is described in **ishii1977one**.
The model is implemented in:

```
void Vitesse_derive_constante::set_param(Param& param)
{
  param.ajouter("C0", &C0, Param::REQUIRED);
  param.ajouter("vg0_x", &vg0[0], Param::REQUIRED);
  param.ajouter("vg0_y", &vg0[1], Param::REQUIRED);
  if (dimension == 3) param.ajouter("vg0_z", &vg0[2], Param::REQUIRED);
}
```

Ishii-Hibiki : Bubbly flow

The model is described in **HIBIKI2002707**.
The model is implemented in:

```
void Vitesse_derive_Ishii::set_param(Param& param)
{
  param.ajouter("subcooled_boiling", &sb_, Param::REQUIRED);
}
```

Default values : `sb_` = 0 (0 : no, 1 : yes), `Cinf` = 1.2, `theta` = 1.75, `zeta` = 18.0.
The implemented model is:

$$C_0 = (\texttt{C\_inf} + (1 - \texttt{C\_inf})\sqrt{\frac{\rho_g}{\rho_l}})(1 - \texttt{sb\_}exp(-\texttt{zeta}\alpha_g)) \tag{9.6}$$

$$\overrightarrow{V_{g0}} = -\sqrt{2}(\frac{\rho_l - \rho_g)g\sigma}{\rho_l^2})^{1/4}(1 - \alpha_g)^{\texttt{theta}}\frac{\overrightarrow{g}}{|g|} \tag{9.7}$$

Spelt Biesheuvel

The model is described in **Spelt1997**.
The model is implemented in:

```
void Vitesse_derive_Spelt_Biesheuvel::set_param(Param& param)
```

Default values `Prt_` = 1.

$$\overrightarrow{V_{g0}} = (-(u_g - u_l)\frac{\overrightarrow{g}}{|g|} + \frac{\nu_{Spelt}\nabla\alpha_g}{max(\alpha_g, 0.0001)})(1 - C_0\alpha_g) \tag{9.8}$$

with

- $C_0 = 1.0$,
- $\nu_{Spelt} = \frac{\left(C_\mu^{1/4}k^{1/2}\right)^2 L}{u_g - u_l}\left(\frac{1}{2} + \frac{3}{8}\left(\frac{\tau^2}{\lambda_t}\right)^2\frac{\lambda_t}{L}\right)$,
- $L = C_\mu^{3/4}\frac{k^{3/2}}{\varepsilon}$,
- $\tau = \frac{u_g - u_l}{2g}$,
- $\lambda_t = \sqrt{10\nu_l\frac{k}{\varepsilon}}$,
- $\varepsilon = C_\mu\frac{k^2}{\nu_t}$.

Forces

The model is implemented in :

```
void Vitesse_derive_Forces::set_param(Param& param)
{
  param.ajouter("alpha_lim", &alpha_lim_);
}
```

Default values : `alpha_lim_` = 1.0×10^{-5}.

$$\overrightarrow{V_{g0}} = \left(-(u_g - u_l)\frac{\overrightarrow{g}}{|g|} + \frac{F^{\text{dispersion}} + F^{\text{lift}}}{f^D U_r}\right)(1 - C_0\alpha_g) \tag{9.9}$$

9.2.2 Two-phase frictional multiplier

The frictional pressure drop in gas–liquid flow can be expressed as a function of a two-phase friction multiplier [**FARAJI2022111863**], based on empirical correlations and both pure liquid

friction f_l and pure gas friction f_g.
The general expression of the two-phase frictional multiplier is :

$$\Phi_k^2 = \frac{(\frac{\partial p}{\partial z})|_{Two-phase}}{(\frac{\partial p}{\partial z})|_{Single\ phase\ k}} \tag{9.10}$$

```
void Multiplicateur_diphasique_base::set_param(Param& param)
```

The available input parameters are:

```
 const double alpha ; // Void fraction
const double rho ; // Density
const double v ; //Velocity
const double f ; // Darcy coefficient as if all the flow rate was in phase k
const double mu ; // Viscosity
const double Dh ; // Hydraulic diameter
const double gamma ; // Surface tension
```

The interfacial heat flux operator must fill $coeff$ tab so that :

- $coeff(k,0)$ multiplier for the single phase friction factor
- $coeff(k,1)$ multiplier for the mix friction factor

Homogeneous

The model is implemented in :

```
void Multiplicateur_diphasique_homogene::set_param(Param& param)
{
  param.ajouter("alpha_min", &alpha_min_);
  param.ajouter("alpha_max", &alpha_max_);
}
```

Default values : `alpha_min_` $= 0.9995$, `alpha_max_` $= 1$.
The model implemented is :

$$\Phi^2 = 1 + x(\frac{\rho_l}{\rho_g} - 1) \tag{9.11}$$

$$coeff(n_l,0) = Frag\_l\Phi^2,\ coeff(n_g,0) = \frac{Frag\_g}{\alpha_g^2} \tag{9.12}$$

With $Frag\_g = min(max(\frac{\alpha_g - \texttt{alpha\_min\_}}{\texttt{alpha\_max\_} - \texttt{alpha\_min\_}}, 0), 1)$,$Frag\_l = 1 - Frag\_g$,

Fridel: horizontal and vertical smooth tubes with $\mu_l/\mu_g < 1000$

The model is described in **friedel1979improved**.
The model is implemented in :

```
void Multiplicateur_diphasique_Friedel::set_param(Param& param)
{
  param.ajouter("alpha_min", &alpha_min_);
  param.ajouter("alpha_max", &alpha_max_);
  param.ajouter("min_lottes_flinn", &min_lottes_flinn_);
  param.ajouter("min_sensas", &min_sensas_);
}
```

Default values : `alpha_min_` $= 1$, `alpha_max_` $= 1.1$, `min_lottes_flinn_` $= 0$, `min_sensas_` $= 0$.
The model implemented is :

$$\Phi^2 = E + \frac{3.24FH}{Fr^{0.0454}We^{0.035}} \tag{9.13}$$

if $F_k min(1, 1.14429\alpha_l^{0.6492}) < \Phi^2 F_m \alpha_l^2$ and `min_sensas` $= 1$ and `min_lottes_flinn_` $= 1$:

$$coeff(n_l, 0) = \frac{Frac\_l}{\alpha_l^2},\ coeff(n_g, 0) = \frac{Frac\_g}{\alpha_l^2}; \tag{9.14}$$

else

$$coeff(n_l, 1) = Frag\_l\Phi^2,\ coeff(n_g, 1) = Frag\_g\Phi^2 \tag{9.15}$$

With $Frag\_g = min(max(\frac{\alpha_g - \texttt{alpha\_min\_}}{\texttt{alpha\_max\_} - \texttt{alpha\_min\_}}, 0), 1)$, $Frag\_l = 1 - Frag\_g$,

- $E = (1-x)^2 + x^2\frac{\rho_l f_g}{\rho_g f_l}$
- $F = x^{0.78}(1-x)^{0.224}$
- $G = \alpha_l\rho_l u_l + \alpha_g\rho_g u_g$
- $H = (\frac{\rho_l}{\rho_g})^{0.91}(\frac{\mu_g}{\mu_l})^{0.19}(1-\frac{\mu_g}{\mu_l})^{0.7}$
- $x = \frac{\alpha_g\rho_g u_g}{G}$
- $Fr = \frac{(\alpha_l\rho_l u_l + \alpha_g\rho_g u_g)^2}{9.81 D_h \rho_m^2}$
- $We = \frac{G^2 D_h}{\sigma\rho_m}$
- $\rho_m = \frac{1}{\frac{x}{\rho_g} + \frac{1-x}{\rho_l}}$

Lottes and Flinn: sodium two-phase pressure drop

The model is described in **lottes1956method**.
The model is implemented in :

```
void Multiplicateur_diphasique_Lottes_Flinn::set_param(Param& param)
{
  param.ajouter("alpha_min", &alpha_min_);
  param.ajouter("alpha_max", &alpha_max_);
}
```

Default values : `alpha_min_` $= 0.9$, `alpha_max_` $= 0.95$.
The model implemented is :

$$coeff(n_l, 0) = \begin{cases} \frac{max(\alpha_l - 1 + \texttt{alpha\_max\_}, 0)}{(1-\texttt{alpha\_min\_})^2}, if \alpha_l < 1 - \texttt{alpha\_min\_} \\ \frac{1}{\alpha_l^2}, otherwise. \end{cases}, \tag{9.16}$$

$$coeff(n_g, 0) = min(max(\frac{\alpha_g - \texttt{alpha\_min\_}}{\texttt{alpha\_max\_} - \texttt{alpha\_min\_}}, 0), 1) \tag{9.17}$$

Muller-Steinhagen: air–water, water-hydrocarbons and refrigerants in pipes

The model is described in **MULLERSTEINHAGEN1986297**.
The model is implemented in :

```
void Multiplicateur_diphasique_Muhler_Steinhagen::set_param(Param& param)
{
  param.ajouter("alpha_min", &alpha_min_);
  param.ajouter("alpha_max", &alpha_max_);
  param.ajouter("min_lottes_flinn", &min_lottes_flinn_);
  param.ajouter("min_sensas", &min_sensas_);
  param.ajouter("a", &a_);
  param.ajouter("b", &b_);
  param.ajouter("c", &c_);
}
```

Default values : `alpha_min_` $= 1$, `alpha_max_` $= 1.1$, `a_` $= 2$, `b_` $= 1$, `c_` $= 3$, `min_lottes_flinn_` $= 0$, `min_sensas_` $= 0$.
The model implemented is :
if $F_k min(1, 1.14429\alpha_l^{0.6492}) < \rho_l F_m \alpha_l^2 \frac{f_m^*}{f_l}$ and `min_sensas` $= 1$ and `min_lottes_flinn_` $= 1$:

$$coeff(n_l, 0) = \frac{min(1, 1.14429\alpha_l^{0.6492})}{\alpha_l^2}. \tag{9.18}$$

else

$$coeff(n_l, 1) = Frac\_l \rho_l \frac{f_m^*}{f_l},\ coeff(n_g, 1) = Frac\_g \rho_g \frac{f_m^*}{f_g} \tag{9.19}$$

With $Frac\_g = min(max(\frac{\alpha_g - \texttt{alpha\_min\_}}{\texttt{alpha\_max\_} - \texttt{alpha\_min\_}}, 0), 1)$, $Frac\_l = 1 - Frac\_g$,

- $G = \alpha_l \rho_l u_l + \alpha_g \rho_g u_g$
- $x = \frac{\alpha_g \rho_g u_g}{G}$
- $fm^* = \left(\frac{f_l}{\rho_l} + a x^b \left(\frac{f_g}{\rho_g} - \frac{f_l}{\rho_l}\right)\right)(1-x)^{1/c} + \frac{f_g}{\rho_g} x^c$.

Chapter 10

Post-processing

This chapter is an introduction or a reminder on the types of post-processing available in TrioCFD/TRUST (section 10.1) as well as the various variables accessible (section 10.2) for correctly visualizing the TrioCFD multiphase calculations.

10.1 Types of post-processing

Regarding the case under study and the variables of interest, `TRUST` offers a range of options for post-processing :

- Individual points of interest ('`Point`' keyword),
- Distributed points along a linear path ('`Segment`' keyword),
- Points arranged according to a predefined layout ('`Plan`' keyword),
- Points arranged within a parallelepiped structure ('`Volume`' keyword),
- Fields across the entire domain. The '`Fields`' keyword demands specifying the field's location on the mesh (faces, elements, or vertices), the field's name, the post-processing time, and a backup file,
- Statistical measurement can be applied to fields to compute the mean value, standard deviation, or correlation between two fields.The '`Statistics`' keyword requires defining a time window, time step, and the desired statistical methods.

10.2 Variables easily accessible

In order to ease computation post-processing, some variables are already accessible with keywords. They are summarized in the following tables.

Let's notice that physical properties (conductivity, diffusivity, etc) can also be post-processed. Furthermore, the `Definition_champs` keyword can be used to create new or more complex fields for advanced post-processing.

Name	Notation	Keyword	Unit
Cell volumes	V_{cell}	`Volume_maille`	m^3
Stability time steps	Δt	`Pas_de_temps`	s
Volumetric porosity	ϵ	`Porosite_volumique`	
Distance to the wall	y_w	`Distance_Paroi`	m
Cell Courant number (VDF only)	$C_o = \frac{u\Delta t}{\Delta x}$	`Courant_maille`	
Cell Reynolds number (VDF only)	$Re = \frac{u\Delta x}{\nu}$	`Reynolds_maille`	

Table 10.1: Cell keywords

Name	Notation	Keyword	Unit
Density	ρ	`masse_volumique`	$kg.m^{-3}$
Void fraction	α	`Alpha`	dimensionless
Mass balance on each cell	$\nabla \cdot u$	`Divergence_U`	$m^3.s^{-1}$

Table 10.2: Mass equation keywords

Name	Notation	Keyword	Unit
Velocity	u	`Vitesse` or `Velocity`	$m.s^{-1}$
Velocity residual	u_{res}	`Vitesse_residu`	$m.s^{-2}$
Kinetic energy per elements	$\frac{1}{2}\rho u^2$	`Energie_cinetique_elem`	$kg.m^{-1}.s^{-2}$
Total kinetic energy	$\frac{1}{2}\rho u^2$	`Energie_cinetique_totale`	$kg.m^{-1}.s^{-2}$
Vorticity	$w = rotu$	`Vorticite`	s^{-1}
Pressure in incompressible flow	$\frac{P}{\rho} + gz$	`Pression`	$Pa.m^3.kg^{-1}$
Pressure in incompressible flow	$P + \rho$ gz	`Pression_pa` or `Pressure`	Pa
Pressure in compressible flow	P	`Pression`	Pa
Hydrostatic pressure	ρgz	`Pression_hydrostatique`	Pa
Total pressure	P_{tot}	`Pression_tot`	Pa
Pressure gradient	$\nabla(\frac{P}{\rho} + gz)$	`Gradient_pression`	$m.s^{-2}$
Velocity gradient	∇u	`gradient_vitesse`	s^{-1}
Local shear strain rate	$\sqrt{2S_{ij}S_{Sij}}$	`Taux_cisaillement`	s^{-1}
Viscous force		`Viscous_force`	$kg.m^2.s^{-1}$
Pressure force		`Pressure_force`	$kg.m^2.s^{-1}$
Total force		`Total_force`	$kg.m^2.s^{-1}$
Viscous force along X		`Viscous_force_x`	$kg.m^2.s^{-1}$
Viscous force along Y		`Viscous_force_y`	$kg.m^2.s^{-1}$
Viscous force along Z		`Viscous_force_z`	$kg.m^2.s^{-1}$
Pressure force along X		`Pressure_force_x`	$kg.m^2.s^{-1}$
Pressure force along Y		`Pressure_force_y`	$kg.m^2.s^{-1}$
Pressure force along Z		`Pressure_force_z`	$kg.m^2.s^{-1}$
Total force along X		`Total_force_x`	$kg.m^2.s^{-1}$
Total force along Y		`Total_force_y`	$kg.m^2.s^{-1}$
Total force along Z		`Total_force_z`	$kg.m^2.s^{-1}$
Component velocity along X	u_X	`VitesseX`	$m.s^{-1}$
Component velocity along Y	u_Y	`VitesseY`	$m.s^{-1}$
Component velocity along Z	u_Z	`VitesseZ`	$m.s^{-1}$
Source term in non Galinean referential	a	`Acceleration_terme_source`	$m.s^{-2}$

Table 10.3: Momentum equation keywords

Name	Notation	Keyword	Unit
Temperature	T	`Temperature`	K
Temperature residual	T_{res}	`Temperature_residu`	$K.s^{-1}$
Temperature variance	$Var(T)$	`Variance_Temperature`	K^2
Temperature dissipation rate		`Taux_Dissipation_Temperature`	$K^2.s^{-1}$
Temperature gradient	∇T	`Gradient_temperature`	$K.m^{-1}$
Heat exchange coefficient	h	`H_echange_Tref`	$W.m^{-2}.K^{-1}$
Internal energy	U	`energie_interne`	J
Enthalpy	H	`enthalpie`	J
Irradiancy	I	`Irradiance`	$W.m^{-2}$
Volumic thermal power	P_w	`Puissance_volumique`	$W.m^{-3}$

Table 10.4: Energy equation keywords

Name	Notation	Keyword	Unit
Turbulent viscosity	ν_t	`Viscosite_turbulente`	$m^2.s^{-1}$
Turbulent dynamic viscosity	μ_t	`Viscosite_dynamique_turbulente`	$kg.m.s^{-1}$
Turbulent kinetic energy	ρk	`Energy`	$kg.m^2.s^{-2}$
Turbulent dissipation rate	ε	`Eps`	$m^2.s^{-3}$
Specific dissipation rate	ω	`omega`	s^{-1}
Specific dissipation time scale	τ	`tau`	s
Q-criteria	Q	`Critere_Q`	s^{-1}
Distance to the wall	y^+	`Y_plus`	
Friction velocity	u^*	`U_star`	$m.s^{-1}$
Turbulent heat flux		`Flux_Chaleur_Turbulente`	$m.K.s^{-1}$

Table 10.5: Turbulence equations keywords

Name	Notation	Keyword	Unit
Drag force	F_D	`Drag`	$N.m^{-3}$
Lift force	F_L	`Lift`	$N.m^{-3}$
Dispersion force	F_{Disp}	`Disp`	$N.m^{-3}$
Lubrication force	F_{Lub}	`Lub`	$N.m^{-3}$
Bubble diameter	d_b	`d_bulles`	m

Table 10.6: Two-phase models keywords

Chapter 11

Appendices

11.1 Matrix management

matrix(where it is derived, about which value it is derived. How is it stored?

For fields stored at the center of the element: index $= N \times e + n$ with N the number of phases of the equation unknown, n the phase number and e the element number. Be careful with turbulence, N might be different for turbulence.

For momentum equation, it depends on the numerical scheme:

- For VDF (velocity on faces): index $= N \times f + n$ with N the number of phases of the equation unknown, n the phase number and f the face number
- For PolyMAC_P0: same as VDF for faces unknown and $N \times (nf_{\text{tot}} + D \times e + d) + n$ for the element unknown with nf_{tot} the total number of faces, D the number of dimensions, e the element number, d the dimension and n the phase number
- For PolyVEF_P0, only unknown on faces, index $= N \times (D \times f + d) + n$

We sometimes use a magic index due to the triangular sup matrix:

$$\left(k(N-1) - (k-1)\frac{k}{2} + l - k - 1\right). \tag{11.1}$$

11.2 Suggestions and planned additions

11.2.1 Potential models for future integration

Physical models for bi-fluid modeling:

- Drag churn of Ishii
- Drag of droplets
- Lift of Hayashi et al.
- Added mass of Cai et Wallis
- Turbulent dispersion of Lavieville
- Particle model of Simmonin
- Large interface reconstruction of Coste

11.2.2 Turbulence

Planned additions:

- Add SST extension for the $k - \omega$ model.
- Add a Reynolds Stress Model

11.3 TODO for the documentation

11.3.1 Turbulence

- add modele_turbulence_longueur_melange
- add viscosite_turbulente_longueur_melange
- add Viscosite_turbulente_l_melange
- finish kader (flux parietal adaptatif)
- add clarification on the ω condition `cond lim demi`

11.3.2 Time Schemes

- Increase detail level in the SETS and ICE scheme presentation

11.4 Nomenclature

This nomenclature is intended to allow developers to understand the code naming choices. It is required to use the same names everywhere. This table will be expended with the standardisation of the multiphase module.

Variable name	Variable type	Variable value	Meaning
ne	const int	domaine.nb_elem()	Number of elements in the domain
D	const int	dimension	Dimension of the problem
N	const int	equation().inconnue()->valeurs().line_size()	Number of phases
pe	const DoubleVect	equation().milieu().porosite_elem()	Field of porosity
ve	const DoubleVect	domaine.volumes();	field of volumes

Table 11.1: Name, types values and meaning of common founded variables in the multiphase module.